Numerical Algorithms 4 (1993) 101–133

An efficient Total Least Squares algorithm based on a rank- revealing two-sided orthogonal decomposition

Sabine Van Huffel [1]

ESAT Laboratory, Department of Electrical Engineering, Katholieke Universiteit Leuven, Kardinaal Mercierlaan 94, 3001 Heverlee, Belgium

Hongyuan Zha

Department of Computer Science, The Pennsylvania State University, University Park, PA 16802, USA

Communicated by Å. Björck
Received 4 July 1991; revised 19 June 1992

Solving Total Least Squares (TLS) problems $AX \approx B$ requires the computation of the noise subspace of the data matrix $[A; B]$. The widely used tool for doing this is the Singular Value Decomposition (SVD). However, the SVD has the drawback that it is computationally expensive. Therefore, we consider here a different so-called rank-revealing two-sided orthogonal decomposition which decomposes the matrix into a product of a unitary matrix, a triangular matrix and another unitary matrix in such a way that the effective rank of the matrix is obvious and at the same time the noise subspace is exhibited explicitly. We show how this decomposition leads to an efficient and reliable TLS algorithm that can be parallelized in an efficient way.

1. Introduction

Solving TLS problems $AX \approx B$ requires the computation of the approximate null space of the data matrix $[A; B]$. The usual tool for doing this is via the Singular Value Decomposition (SVD), defined as follows [7, p. 71]. Given an $m \times n$ matrix, $C, m \geqslant n$, there exist unitary matrices $U \in \mathbb{C}^{m \times m}$ and $V \in \mathbb{C}^{n \times n}$ such that

$$U^H C V = \begin{bmatrix} \Sigma \\ 0 \end{bmatrix}, \tag{1}$$

with $\Sigma_{n \times n} = \operatorname{diag}(\sigma_1, \ldots, \sigma_n)$ the diagonal elements of which are the singular values and $\sigma_1 \geqslant \ldots \geqslant \sigma_n \geqslant 0$.

[1] Research Associate of the Belgian N.F.W.O. (National Fund for Scientific Research).

The procedure for computing approximate null spaces is to determine the numerical rank k of the matrix C such that σ_k is above the noise level while σ_{k+1} is below it. Appropriate noise level determinators have been given in [21,chap.3] and [13]. The columns $V_2 = [v_{k+1}, \ldots, v_n]$, corresponding to $\sigma_{k+1}, \ldots, \sigma_n$, then span the approximate null space. Once a basis of the range $R(V_2)$ is known, the TLS solution can be immediately computed, as outlined in [19, algor. 4.1] (see also [21]).

Although the SVD furnishes an elegant solution to the problem of calculating null spaces, it is computationally expensive. This serious drawback explains the renewed interest in Rank-Revealing (RR) QR decompositions [2,6,1,4,3], which compute a column permutation Π and a QR decomposition $C\Pi = \mathrm{QR}$ of an $m \times n$ matrix C, $m \geqslant n$, in such a way that the approximate rank k of the matrix is revealed in the upper triangular factor R having a small lower right block. As a by-product, this RR QR decomposition also furnishes an approximation W to the numerical null space $R(V_2)$ which can be further improved by means of inverse iteration [1,4]. As long as the gap between σ_k and σ_{k+1} is well-defined, it is shown in [3, theorem 4.1] that $R(W)$ is a good approximation to $R(V_2)$ in the sense that the subspace angle between these spaces is small. However, the approximate null space W is not unitary and is not represented explicitly in the RR QR formulation.

Therefore, Stewart presented an intermediary between the SVD and the QR decomposition, a rank-revealing two-sided orthogonal decomposition [13], that has the virtues of both. This decomposition stands for any URV or ULV decomposition, which reduce the matrix to upper and lower triangular form respectively by means of unitary transformations U and V, and is introduced in section 2. We will also show how to compute a RR URV and ULV decomposition of a matrix and prove some properties. In section 3, a new TLS algorithm based on this decomposition is described and implemented on a linear array of n processors in such a way that it runs in $O(n-k)$ time. Furthermore, it is shown how this algorithm can be extended to multidimensional TLS problems with more than one right-hand side vector and nongeneric TLS problems, and some general observations on updating are made. Finally, the newly presented TLS algorithm is evaluated: numerical results are given and its efficiency is compared with that of other TLS algorithms.

Before starting, we present some additional notation used throughout this paper.

- The SVD of a matrix C is denoted by (1). The smallest singular value is written as $\sigma_{\min}(C)$ and the corresponding left and right singular vectors are respectively denoted by $u_{\min}(C)$ and $v_{\min}(C)$. The set of singular values is denoted by $\sigma(C)$.
- $\| \cdot \|$ denotes the Euclidean vector norm and the Frobenius matrix norm, defined by $\|C\|^2 = \sum_{i,j} c_{ij}^2$.
- The $m \times m$ identity matrix is denoted by I_m or simply by I.
- The superscript H denotes the Hermitian conjugate, i.e. the complex conjugate and transpose of a matrix or vector.
- A matrix C is unitary if it satisfies $C^H C = I = CC^H$.

- est(x) stands for an estimate of x.
- A special notation is used for plane rotations which are the chief computational tool of this paper. Denote by $J_j^{(i)}$ a plane rotation of order j that is applied to columns i and $i+1$ of a premultiplied matrix or to rows i and $i+1$ of a postmultiplied matrix. The reader is assumed to be familiar with plane rotations which are discussed in most texts on numerical linear algebra, see e.g. [2,13,7].
- A flop is defined as in [7, p. 19] and is the amount of work associated with an addition, a multiplication or a square root, e.g. an inner product $x^H y$ of 2 vectors x and y of length n involves $2n$ flops. For real matrices a flop equals a floating point operation.
- The distance between any two subspaces S_1 and S_2 is denoted by $\text{dist}(S_1, S_2)$ (see [7, sec. 2.6.3] for a definition).

2. Rank-revealing two-sided orthogonal decompositions

2.1. DEFINITIONS

The URV and ULV decompositions of a matrix, as presented in [13,15], [7, sec. 5.4.2], are defined as follows.

DEFINITION 1: URV AND ULV DECOMPOSITION

Given an $m \times n$ matrix C, $m \geqslant n$, of rank k, there exist unitary matrices $U \in \mathbb{C}^{m \times n}$ and $V \in \mathbb{C}^{n \times n}$ such that

$$U^H C V = T, \tag{2}$$

where T is either of the form

$$T = \begin{bmatrix} R & 0 \\ 0 & 0 \end{bmatrix} \begin{matrix} k \\ m-k \end{matrix} \qquad \text{URV decomposition}$$
$$\qquad\quad k \quad n-k$$

with R an upper triangular matrix or

$$T = \begin{bmatrix} L & 0 \\ 0 & 0 \end{bmatrix} \begin{matrix} k \\ m-k \end{matrix} \qquad \text{ULV decomposition}$$
$$\qquad\quad k \quad n-k$$

with L a lower triangular matrix.

In practice, and certainly in TLS applications, C will not have rank k because of superimposed noise or numerical errors. Therefore, the exact URV and ULV decompositions, as defined above, are only of theoretical interest. In practice, we merely deal with the rank-revealing variants, as defined below.

DEFINITION 2: RANK-REVEALING (RR) URV AND ULV DECOMPOSITION

Given an $m \times n$ matrix C, $m \geqslant n$, of approximate rank k in the sense that its singular values satisfy

$$\sigma_1 \geqslant \ldots \geqslant \sigma_k > \sigma_{k+1} \geqslant \ldots \geqslant \sigma_n,$$

where σ_k is large compared to σ_{k+1}, there exist unitary matrices $U \in \mathbb{C}^{m \times m}$ and $V \in \mathbb{C}^{n \times n}$ such that

$$U^H C V = T, \tag{3}$$

where T is either of the form

$$T = \begin{bmatrix} R & F \\ 0 & G \\ 0 & 0 \end{bmatrix} \begin{matrix} k \\ n-k \\ m-n \end{matrix} \quad \text{rank-revealing URV decomposition}$$

$$\qquad\quad k \quad n-k$$

with R, G upper triangular, $\sigma_{\min}(R) \cong \sigma_k$ and $\sqrt{\|F\|^2 + \|G\|^2} \cong \sqrt{\sigma_{k+1}^2 + \ldots + \sigma_n^2}$ or

$$T = \begin{bmatrix} L & 0 \\ H & E \\ 0 & 0 \end{bmatrix} \begin{matrix} k \\ n-k \\ m-n \end{matrix} \quad \text{rank-revealing ULV decomposition}$$

$$\qquad\quad k \quad n-k$$

with L, E lower triangular, $\sigma_{\min}(L) \cong \sigma_k$ and $\sqrt{\|H\|^2 + \|E\|^2} \cong \sqrt{\sigma_{k+1}^2 + \ldots + \sigma_n^2}$.

Observe that these decompositions are not unique. In particular, the SVD is itself a special (rank-revealing) URV and ULV decomposition. From such a decomposition we can extract the approximate null space, just as we did with the SVD. However, RR URV and ULV decompositions are much cheaper to compute, as shown below.

2.2. ALGORITHMS

In [13–15] it is shown how to compute a RR URV and ULV decomposition, but no detailed algorithm is given. For clarity of exposition, we outline the algorithms here since they will also be used in the TLS computation (see section 3). In addition, the presentations given here also introduce some of the notations used later.

In essence, the RR URV algorithm computes a sequence of upper triangular matrices R_i of decreasing order i. The matrix R_i is judged to be rank-deficient if there exists a vector w_i of unit length such that $\|R_i w_i\|$ is less than some prespecified noise level. Such vectors can be reliably found by means of a condition estimator [8] or

by applying inverse iteration [12]. Having found such a vector w_i, the matrix R_i is deflated as follows. Unitary matrices P_i and Q_i are constructed such that

(1) $Q_i^H w_i = e_i = [0, \ldots, 0, 1]^H$ where e_i is the ith unit vector,
(2) $\hat{R}_i = P_i^H R_i Q_i$ is upper triangular.

It then follows that

$$\omega_i = \|R_i w_i\| = \|P_i^H R_i Q_i Q_i^H w_i\| = \|\hat{R}_i e_i\|,$$

i.e. the last column r_i of $\hat{R}_i$ has small norm ω_i such that

$$\hat{R}_i = \begin{bmatrix} R_{i-1} & f_i \\ 0 & g_i \end{bmatrix} = \left[\begin{pmatrix} R_{i-1} \\ 0 \end{pmatrix}, \quad r_i \right] \quad \text{with } \|r_i\|^2 = \|f_i\|^2 + g_i^2 = \omega_i^2$$

has the (partially) RR form (3). The process is stopped at $i = k$ since $\sigma_{\min}(R_k) \cong \|R_k w_k\|$ is considered to be above the noise level. Overall, we have the following algorithm.

ALGORITHM 1: RANK-REVEALING URV DECOMPOSITION

Given: • $C_{m\times n}$, $m \geqslant n$,
- a noise level determinator *tol* and bounds $k_{\min}$, $k_{\max}$ on the computed rank k such that $k_{\min} \leqslant k \leqslant k_{\max}$,
- logical variables *wantf*, *wantu* and *wantv* indicating the need for computing respectively the matrix F and the unitary matrices U and V explicitly.

Step 1: Compute $C = Q_C \begin{bmatrix} R_n \\ 0 \end{bmatrix} \begin{matrix} n \\ \end{matrix}$, $Q_C^H Q_C = I_m$ and R_n upper triangular {*QRfactor.*}

Step 2: Initialize: if (*wantu*) then $U \leftarrow Q_C$; if (*wantv*) then $V \leftarrow I_n$;
$i \leftarrow n$; $w_n \leftarrow \text{est}(v_{\min}(R_n))$ with $\|w_n\| = 1$

Step 3: While ($\|R_i w_i\| < tol$ or $i > k_{\max}$) and $i > k_{\min}$ do

(a) compute plane rotations $Q_i = J_i^{(1)} \ldots J_i^{(i-1)}$ such that

$$Q_i^H w_i = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{matrix} i-1 \\ 1 \end{matrix}$$

(b) if (*wantu*) then accumulate Q_i into V:

$$V \leftarrow V \begin{bmatrix} Q_i & 0 \\ 0 & I_{n-i} \end{bmatrix}$$

(c) compute plane rotations $P_i = I_i^{(1)} \ldots I_i^{(i-1)}$ such that $\hat{R}_i = P_i^H R_i Q_i$ is upper triangular

(d) if (*wantu*) then accumulate P_i into U:

$$U \leftarrow U \begin{bmatrix} P_i & 0 \\ 0 & I_{m-i} \end{bmatrix}$$

(e) partition $\hat{R}_i = {}_{1}\left[\begin{pmatrix} R_{i-1} \\ 0 \end{pmatrix}, \begin{matrix} r_i \\ 1 \end{matrix}\right]$ $\begin{matrix} \\ i-1 \end{matrix}$

(f) if (*wantf*) then set

$$\begin{bmatrix} F_{i-1} \\ G_{i-1} \end{bmatrix} \begin{matrix} i-1 \\ n-i+1 \end{matrix} \leftarrow \left[\begin{pmatrix} r_i \\ 0 \end{pmatrix}, \begin{bmatrix} P_i^H F_i \\ G_i \end{bmatrix} \begin{matrix} i \\ n-i \end{matrix}\right]$$
$$\begin{matrix} n-i+1 \end{matrix} \qquad\qquad \begin{matrix} n-i \end{matrix}$$

(g) compute $w_{i-1} = \text{est}(v_{\min}(R_{i-1}))$ with $\|w_{i-1}\| = 1$

(h) $i \leftarrow i-1$

end while

Step 4: $k \leftarrow i$

END

Submatrices with zero row or column dimension may be annihilated, e.g. F_n and G_n in step (3f). The RR ULV decomposition is computed analogously but now a sequence of lower triangular matrices L_i of decreasing order i is generated, as outlined below.

ALGORITHM 2: RANK-REVEALING ULV DECOMPOSITION

Given: • $C_{m\times n}, m \geqslant n$,

- a noise level determinator *tol* and bounds $k_{\min}$, $k_{\max}$ on the computed rank k such that $k_{\min} \leqslant k \leqslant k_{\max}$,
- logical variables *wantf*, *wantu* and *wantv* indicating the need for computing respectively the matrix H and the unitary matrices U and V explicitly.

Step 1: Compute $C = Q_C \begin{bmatrix} L_n \\ 0 \end{bmatrix} \begin{matrix} n \\ m-n \end{matrix}$, $Q_C^H Q_C = I_m$ and L_n lower triangular $\{QLfactor.\}$

Step 2: Initialize: if (*wantu*) then $U \leftarrow Q_C$; if (*wantv*) then $V \leftarrow I_n$;
$i \leftarrow n$; $w_n \leftarrow \text{est}(u_{\min}(L_n))$ with $\|w_n\| = 1$

Step 3: While ($\|L_i^H w_i\| < tol$ or $i > k_{\max}$) and $i > k_{\min}$ do

(a) compute plane rotations $P_i = I_i^{(1)} \ldots I_i^{(i-1)}$ such that

$$P_i^H w_i = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{matrix} i-1 \\ 1 \end{matrix}$$

(b) if (*wantu*) then accumulate P_i into U:

$$U \leftarrow U \begin{bmatrix} P_i & 0 \\ 0 & I_{m-i} \end{bmatrix}$$

(c) compute plane rotations $Q_i = J_i^{(1)} \ldots J_i^{(i-1)}$ such that
$\hat{L}_i = P_i^H L_i Q_i$ is lower triangular

(d) if (*wantv*) then accumulate Q_i into V:

$$V \leftarrow V \begin{bmatrix} Q_i & 0 \\ 0 & I_{n-i} \end{bmatrix}$$

(e) partition $\hat{L}_i = \begin{bmatrix} (L_{i-1}, & 0) \\ l_i^H & \\ & 1 \end{bmatrix} \begin{matrix} i-1 \\ 1 \end{matrix}$

(f) if (*wantf*) then set

$$\begin{matrix} [H_{i-1}; & E_{i-1}] \\ i-1 & n-i+1 \end{matrix} \leftarrow \begin{bmatrix} (l_i^H, & 0) \\ [H_i Q_i; & E_i] \end{bmatrix} \begin{matrix} \\ i \quad n-i \end{matrix}$$

(g) compute $w_{i-1} = \text{est}(u_{\min}(L_{i-1}))$ with $\|w_{i-1}\| = 1$

(h) $i \leftarrow i-1$

end while

Step 4: $k \leftarrow i$

END

Applying algorithm 1 (resp., 2) yields the desired RR URV (resp., ULV) decomposition (3) of matrix C with computed rank k where

$$U = Q_C \begin{bmatrix} P_n & 0 \\ 0 & I_{m-n} \end{bmatrix} \cdots \begin{bmatrix} P_{k+1} & 0 \\ 0 & I_{m-k-1} \end{bmatrix},$$

$$V = Q_n \begin{bmatrix} Q_{n-1} & 0 \\ 0 & 1 \end{bmatrix} \cdots \begin{bmatrix} Q_{k+1} & 0 \\ 0 & I_{n-k-1} \end{bmatrix},$$

$$R = R_k \text{ (resp., } L = L_k),$$

$$\begin{bmatrix} F \\ G \end{bmatrix} \begin{matrix} k \\ n-k \end{matrix} = \begin{bmatrix} F_k \\ G_k \end{bmatrix} \text{ (resp., } \begin{matrix} [H; & E] \\ k & n-k \end{matrix} = [H_k; E_k]).$$

The RR URV decomposition can be refined, as described by Stewart [13,14], in order to produce for each i a better approximate null vector of $\hat{R}_i$. This refinement step, which is designed to reduce $\|r_i\|$ and can be repeated iteratively, must be performed after step 3(f) within the while-loop. Denote by L the number of desired refinement steps to be carried out, then this refinement step proceeds as follows.

set $l \leftarrow L$

while $l > 0$ do

compute a unitary matrix $Q_i^{(l)}$ such that

$$\hat{R}_i Q_i^{(l)} = \begin{bmatrix} \tilde{R}_{i-1} & 0 \\ h_i^H & \epsilon_i \end{bmatrix} \begin{matrix} i-1 \\ 1 \end{matrix}$$
$$\qquad\quad i-1 \quad 1$$

where $\tilde{R}_{i-1}$ is upper triangular

if (*wantv*) then accumulate $Q_i^{(l)}$ into V: $V \leftarrow V \begin{bmatrix} Q_i^{(l)} & 0 \\ 0 & I_{n-i} \end{bmatrix}$

compute a unitary matrix $P_i^{(l)}$ such that $P_i^{(l)H} \hat{R}_i Q_i^{(l)}$ is upper triangular

if (*wantu*) then accumulate $P_i^{(l)}$ into U: $U \leftarrow U \begin{bmatrix} P_i^{(l)} & 0 \\ 0 & I_{m-i} \end{bmatrix}$

set $\hat{R}_i = \left[\begin{pmatrix} R_{i-1} \\ 0 \end{pmatrix}, r_i \right] \leftarrow P_i^{(l)H} \hat{R}_i Q_i^{(l)}$

if (*wantf*) then set

$$\begin{bmatrix} F_{i-1} \\ G_{i-1} \end{bmatrix} = \left[\begin{pmatrix} r_i \\ 0 \end{pmatrix}, \begin{bmatrix} FF \\ GG \end{bmatrix} \right] \leftarrow \left[\begin{pmatrix} r_i \\ 0 \end{pmatrix}, \begin{bmatrix} P_i^{(l)H} FF \\ GG \end{bmatrix} \right]$$
$$\qquad\quad 1 \qquad n-i$$

$l \leftarrow l+1$
end while

Each refinement step requires $6i^2 + O(i)$ flops, assuming that F, U and V are not needed and ordinary Givens rotations are used (or $4i^2 + O(i)$ flops, if fast Givens rotations are used). These refinement steps are not needed in the RR ULV algorithm and were the main reason for introducing this decomposition [15]. Although a ULV decomposition can be expected to give a higher quality approximate null space, it is not yet clear under what circumstances one decomposition should be preferred to the other. Alternatively, the RR URV decomposition can be refined by starting from a better estimate of $v_{\min}(R_i)$, e.g. by performing more inverse iteration steps such that w_i converges more closely to $v_{\min}(R_i)$. This requires $2i^2 + O(i)$ flops per step implying that we can perform 3 inverse iteration steps at the cost of one refinement step. Whether the refinement is worth its cost and should be preferred to refinement of the estimate by inverse iteration is to be determined experimentally. However, this study is beyond the scope of this paper, merely because this iterative refinement procedure can not be used in conjunction with the TLS algorithm presented in section 3.

Let us now investigate the amount of work involved in performing the RR URV algorithm 1. Throughout this paper, we only consider the highest order terms and throw away the low order terms, so our flop counts are valid for large n. Assum-

ing that F is not needed explicitly and no accumulation into U or V is required, algorithm 1 requires

$$2n^2\left(m-\frac{n}{3}\right)+\left(\frac{2}{3}K+2+2L\right)(n^3-k^3)+O(n^2) \text{ flops},$$

where L is the number of refinement steps carried out for refining each $v_{\min}(R_i)$. K depends on the method used for estimating $v_{\min}(R_i)$. If computed by inverse iteration applied implicitly to $R_i^H R_i$ [12], K denotes the number of performed iteration steps. Usually, $K=2$ is sufficient in practice, especially if an efficient condition estimator is used to select the initial vector. If one of the existing condition estimators, as surveyed in [8], is used then K is given by [8, table 8.1] depending on the chosen algorithm, e.g. $K=5$ if the 2-norm look-behind algorithm is used. This computed estimate can further be refined by I inverse iteration steps. If this is the case, K must be augmented with I.

The RR ULV algorithm 2 requires the same amount of work but no refinement steps are needed ($L=0$). So we have

$$2n^2\left(m-\frac{n}{3}\right)+\left(\frac{2}{3}K+2\right)(n^3-k^3)+O(n^2) \text{ flops}.$$

If fast Givens rotations are used, the flop counts are given by:

$$2n^2\left(m-\frac{n}{3}\right)+\left(\frac{2}{3}K+\frac{4}{3}+\frac{4}{3}L\right)(n^3-k^3)+O(n^2)$$

flops for RR URV Algorithm 1,

$$2n^2\left(m-\frac{n}{3}\right)+\left(\frac{2}{3}K+\frac{4}{3}\right)(n^3-k^3)+O(n^2)$$

flops for RR ULV Algorithm 2.

The flop counts given above hold when F is not needed explicitly. This is the case in many applications, in particular when the RR URV or ULV decomposition is used for computing the null space of a matrix, e.g. in all TLS applications. Nevertheless, if F is needed then

$$(1+L)[n^3-k^3(3n-2k)]+O(n^2) \text{ flops}$$

$$\left(\text{resp., } \frac{2}{3}(1+L)[n^3-k^3(3n-2k)]+O(n^2) \text{ flops}\right)$$

should be added to the above counts if ordinary (resp., fast) Givens rotations are used. Note that G is always explicitly available without performing additional computations.

It is important to note that the extra work performed after the initial QR (or QL) factorization is of low order and negligible if $k \approx n$, especially when $m \gg n$. Even in the extreme case of $k=0$, we have to perform only $2n^2(m+\frac{4}{3}n)$ flops (as-

suming that $K = 2$ and $L = 0$) which is considerably smaller than the cost of $2n^2(2m + 2n)$ flops for computing the SVD [7, sec. 5.4.5]. Also observe that the RR URV or ULV decomposition actually requires slightly more computational work than the RR QR factorization [2, sec. 5]: approximately $\frac{2}{3}(n^3 - k^3)$ more flops if F is not needed (pay attention: the flop counts given in [2, sec. 5] must be doubled to be in accordance with our definition here; also note that these counts are too much overestimated for small k).

2.3. PROPERTIES

If C has exactly rank k, i.e. $\sigma_k > \sigma_{k+1} = \ldots = \sigma_n = 0$, then there exists a two-sided orthogonal decomposition of the form (2). In practice, we always deal with matrices of approximate rank, k, i.e. $\sigma_{k+1}, \ldots, \sigma_n$ are negligibly small compared to σ_k. For these matrices the approximate null space is given by the right singular subspace corresponding to its smallest singular values $\sigma_{k+1}, \ldots, \sigma_n$. For TLS applications we must compute this subspace up to a data-dependent precision determined by the noise added to the data. In this section we prove that the quality of the estimated null space vectors $\text{est}(v_{\min}(R_i))$ or $\text{est}(u_{\min}(L_i))$ determines the closeness between the approximate null space $V_2 = [v_{k+1}, \ldots, v_n]$, computed by means of the RR URV or ULV algorithm, and the exact right singular subspace of the given matrix corresponding to its smallest singular values $\sigma_{k+1}, \ldots, \sigma_n$. More specifically, choosing all estimates w_i exactly equal to $v_{\min}(R_i)$ leads to a RR URV decomposition (3) where $F = 0$, $G = \text{diag}(\sigma_{k+1}, \ldots, \sigma_n)$ and both spaces coincide, as proven in the following theorem.

THEOREM 1: EXACT RR URV DECOMPOSITION

Given an $m \times n$ matrix C, $m \geq n$, of approximate rank k whose SVD is given by:

$$U^H C V = \begin{bmatrix} \Sigma_k & 0 \\ 0 & \Sigma_0 \\ 0 & 0 \end{bmatrix} \begin{matrix} k \\ n-k \\ m-n \end{matrix}$$
$$\qquad\quad k \qquad n-k$$

with

$$U = [U_k; \quad U_0; \quad U_\perp \quad]$$
$$\qquad k \quad n-k \quad m-n$$

and

$$V = [V_k; \quad V_0 \quad] \tag{4}$$
$$\qquad k \quad n-k$$

$\Sigma_k = \text{diag}(\sigma_1, \ldots, \sigma_k)$ and $\Sigma_0 = \text{diag}(\sigma_{k+1}, \ldots, \sigma_n)$.

Assume that $w_i = v_{\min}(R_i)$, $i = k+1, \ldots, n$, where R_i is the $i \times i$ upper triangular matrix computed in the RR URV algorithm 1. Then the RR URV decomposition (3) of C reduces to:

$$U_R^H C V_R = \begin{bmatrix} R_k & 0 \\ 0 & \Sigma_0 \\ 0 & 0 \end{bmatrix} \begin{matrix} k \\ n-k \\ m-n \end{matrix}$$
$$\qquad\quad k \qquad n-k$$

with

$$U_R = [U_1; \quad U_2; \quad U_\perp \ ]$$
$$\qquad\quad k \quad n-k \quad m-n$$

and

$$V_R = [V_1; \quad V_2 \ ], \tag{5}$$
$$\qquad\quad k \quad n-k$$

where σ_i is the ith singular value of C and the ith column of U_R (resp., V_R) is the corresponding left (resp., right) singular vector for $i = k+1, \ldots, n$, i.e. the columns of U_2 and V_2 span exactly the left and right singular subspace respectively of C corresponding to its smallest singular values $\sigma_{k+1}, \ldots, \sigma_n$.

Proof

We use the notations used in the RR URV algorithm 1. In order to prove (5), we need to prove that for each i

$$P_i^H R_i Q_i = \begin{bmatrix} R_{i-1} & 0 \\ 0 & \sigma_i \end{bmatrix},$$

with σ_i the ith singular value of C.

Let $R_i = \sum_{j=1}^{i} \sigma_j^{(i)} u_j^{(i)} v_j^{(i)H}$ be the SVD of R_i, then $w_i = v_i^{(i)}$ by assumption and we have:

$$R_i v_i^{(i)} = \sigma_i^{(i)} u_i^{(i)} \quad \text{and} \quad R_i^H u_i^{(i)} = \sigma_i^{(i)} v_i^{(i)}.$$

Now, $R_i Q_i (Q_i^H v_i^{(i)}) = \sigma_i^{(i)} u_i^{(i)}$ where Q_i is such that

$$Q_i^H v_i^{(i)} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{matrix} i-1 \\ 1 \end{matrix}$$

and

$$P_i^H R_i Q_i \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \sigma_i^{(i)} P_i^H u_i^{(i)},$$

with P_i such that

$$P_i^H R_i Q_i = \begin{bmatrix} R_{i-1} & f_i \\ 0 & g_i \end{bmatrix}$$

is upper triangular. Thus,

$$\begin{bmatrix} R_{i-1} & f_i \\ 0 & g_i \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{matrix} i-1 \\ 1 \end{matrix} = \begin{bmatrix} f_i \\ g_i \end{bmatrix} = \sigma_i^{(i)} P_i^H u_i^{(i)}, \tag{6}$$

implying that the last column of $P_i^H R_i Q_i$ has norm $\sigma_i^{(i)}$ and is proportional to $P_i^H u_i^{(i)}$. To prove that

$$P_i^H R_i Q_i = \begin{bmatrix} R_{i-1} & 0 \\ 0 & \sigma_i \end{bmatrix},$$

we must prove that

$$\begin{bmatrix} f_i \\ g_i \end{bmatrix} = \begin{bmatrix} 0 \\ \sigma_i \end{bmatrix},$$

i.e. $g_i = \sigma_i^{(i)} = \sigma_i$ and

$$P_i^H u_i^{(i)} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{matrix} i-1 \\ 1 \end{matrix}.$$

We first prove the last assertion.
Consider the set $R_i^H u_i^{(i)} = \sigma_i^{(i)} v_i^{(i)}$ and perform P_i and Q_i:

$$Q_i^H R_i^H P_i P_i^H u_i^{(i)} = \sigma_i^{(i)} Q_i^H v_i^{(i)} = \sigma_i^{(i)} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{matrix} i-1 \\ 1 \end{matrix}$$

$$\Downarrow$$

$$(P_i^H R_i Q_i)^H (P_i^H u_i^{(i)}) = \begin{bmatrix} 0 \\ \sigma_i^{(i)} \end{bmatrix}. \tag{7}$$

Since $\hat{R}_i = P_i^H R_i Q_i$ is upper triangular, $\hat{R}_i^H = \hat{L}_i$ must be lower triangular. Using this notation, (7) implies:

$$\hat{R}_i^H (P_i^{(i)H} u_i^{(i)}) = \hat{L}_i P_i^H u_i^{(i)} = \begin{bmatrix} 0 \\ \sigma_i^{(i)} \end{bmatrix} \begin{matrix} i-1 \\ 1 \end{matrix} \Rightarrow P_i^H u_i^{(i)} = \hat{L}_i^{-1} \begin{bmatrix} 0 \\ \sigma_i^{(i)} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

The inverse $\tilde{L}_i = \hat{L}_i^{-1}$ of a lower triangular matrix $\hat{L}_i$ is lower triangular. So,

$$P_i^H u_i^{(i)} = \tilde{L}_i \begin{bmatrix} 0 \\ \sigma_i^{(i)} \end{bmatrix}$$

is zero except for its last entry. Since $\|P_i^H u_i^{(i)}\| = 1$, it follows that

$$P_i^H u_i^{(i)} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{matrix} i-1 \\ 1 \end{matrix}.$$

Substituting this into (6) gives

$$\begin{bmatrix} f_i \\ g_i \end{bmatrix} = \sigma_i^{(i)} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ \sigma_i^{(i)} \end{bmatrix}.$$

It remains to prove that $\sigma_i^{(i)} = \sigma_i$ the ith singular value of C, which can be done by induction.

(1) For $i = n$ we have that $\sigma(R_n) = \sigma(C) = \{\sigma_j | j = 1, \ldots, n\}$. Indeed, for $i = n$ we have $R_i = R_n$ with $C = Q_C \begin{bmatrix} R_n \\ 0 \end{bmatrix}$ the QR decomposition of C. The unitarity of Q_C implies that $\sigma(R_n) = \sigma(C)$ and hence $\sigma_n^{(n)} = \sigma_{\min}(R_n) = \sigma_{\min}(C) = \sigma_n$.

(2) If $\sigma(R_i) = \{\sigma_j | j = 1, \ldots, i\}$ then we must prove that $\sigma(R_{i-1}) = \{\sigma_j | j = 1, \ldots, i-1\}$. Indeed, we have

$$P_i^H R_i Q_i = \begin{bmatrix} R_{i-1} & 0 \\ 0 & \sigma_i^{(i)} \end{bmatrix},$$

with $\sigma_i^{(i)} = \sigma_{\min}(R_i) = \sigma_i$ by assumption. The unitarity of P_i, Q_i imply that $\sigma(R_i) = \{\sigma(R_{i-1}, \sigma_i\}$. Since $\sigma(R_i) = \{\sigma_1, \ldots, \sigma_i\}$ we must have $\sigma(R_{i-1}) = \sigma(R_i) \backslash \{\sigma_i\} = \{\sigma_1, \ldots, \sigma_{i-1}\}$. Hence, $\sigma_{i-1}^{(i-1)} = \sigma_{\min}(R_{i-1}) = \sigma_{i-1}$.

Moreover, U_2 and V_2 are left and right singular vectors of C, corresponding with the singular values Σ_2, since they satisfy the singular vector equations:

$$Cv_i = \sigma_i u_i \text{ and } C^H u_i = \sigma_i v_i \text{ for each column } v_i \in V_2 \text{ and } u_i \in U_2, i = k+1, \ldots, n,$$

i.e. the noise subspaces spanned by the columns of U_2 and V_2, as computed by the RR URV algorithm, coincide exactly with the exact left and right singular subspaces of C corresponding to its smallest singular values. □

Of course, if all $\sigma_i, i = k, \ldots, n$ are different, then $U_2 = U_0$ and $V_2 = V_0$ up to a sign change in its columns. Under the assumptions of theorem 1, (5) reveals exactly the rank of C and is therefore called here the "exact" RR URV decomposition. A similar theorem can be proven for the RR ULV decomposition under the assumptions that $w_i = u_{\min}(L_i)$, $i = k+1, \ldots, n$. When $k = 0$, the "exact" RR URV or ULV decomposition equals the SVD of the matrix but it is obtained in a much cheaper way (as shown in section 2.2) provided the number of required inverse iteration steps K to obtain the exact $v_{\min}(R_i)$ is not too large.

Let (1) be the SVD of C and assume for the rest of this paragraph that C has approximate rank $k = n - 1$ implying that $\sigma_{n-1} > \sigma_n$. Then, provided $w_n = v_{\min}(C) = v_n$, theorem 1 states that the RR URV decomposition of C is given by:

$$[U_1; u_n; U_\perp]^H C [V_1; v_n] = \begin{bmatrix} R_{n-1} & 0 \\ 0 & \sigma_n \\ 0 & 0 \end{bmatrix}. \tag{8}$$

Usually, $w_n = \sum_{i=1}^n \alpha_i v_i = V\alpha \neq v_{\min}(C)$ with $\alpha = [\alpha_1 \ldots \alpha_n]^T$. Applying algorithm 1, we end up with a RR URV decomposition of the form:

$$U_R^H C V_R = \begin{bmatrix} R_{n-1} & f_n \\ 0 & g_n \\ 0 & 0 \end{bmatrix}, \tag{9}$$

where

$$\|f_n\|^2 + g_n^2 = \|r_n\|^2 = \|\hat{R}_n(Q_n^H w_n)\|^2 = w_n^H Q_n^H \hat{R}_n^H \hat{R}_n Q_n w_n . \tag{10}$$

Since $\hat{R}_n = P_n^H R_n Q_n$ and $R_n^H R_n = V\Sigma^H \Sigma V^H$, (10) yields:

$$\|r_n\|^2 = \|f_n\|^2 + g_n^2 = w_n^H V\Sigma^H \Sigma V^H w_n = \alpha^H \Sigma^H \Sigma \alpha = \sum_{i=1}^{n} \alpha_i^2 \sigma_i^2 . \tag{11}$$

Also, since

$$Q_n^H w_n = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{matrix} n-1 \\ 1 \end{matrix},$$

it follows that

$$w_n = Q_n \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{matrix} n-1 \\ 1 \end{matrix},$$

i.e. w_n must equal the last column q_n of Q_n and thus $w_n = \sum_{i=1}^{n} \alpha_i v_i = q_n$. The better w_n estimates $v_{\min}(R_n) = v_{\min}(C)$, the smaller $\alpha_1, \ldots, \alpha_{n-1}$ will be and the closer the RR URV decomposition (9) will approach (8).

Ideally $w_n = v_n = q_n$ exactly. Then

$$\alpha = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \text{ and } r_n = \begin{bmatrix} f_n \\ g_n \end{bmatrix} = \begin{bmatrix} 0 \\ \sigma_n \end{bmatrix}.$$

In this case, our approximate null space vector w_n coincides exactly with $v_{\min}(C)$, the right singular vector of C corresponding with its smallest singular value. If w_n is not exactly equal to v_n but $w_n = V\alpha$, $\|w_n\| = 1$, then f_n is no longer zero. The smaller $\|f_n\|$, the closer w_n approximates the desired null space vector $v_{\min}(C)$, as proven more generally in the following theorem.

THEOREM 2: BOUNDS ON THE RR URV AND ULV DECOMPOSITION

Let (3) be the RR URV and ULV decomposition of a given $m \times n$ matrix C, $m \geq n$, of approximate rank k. Partition

$$U = \begin{matrix} [U_1; & U_2; & U_\perp \ ] \\ k & n-k & m-n \end{matrix} \text{ and } V = \begin{matrix} [V_1; & V_2 \ ]. \\ k & n-k \end{matrix}$$

Let (4) be the SVD of C, then the distance between $R(V_2)$ and $R(V_0)$ (resp., $R(U_2)$ and $R(U_0)$) is bounded by:

$$\begin{aligned} \operatorname{dist}(R(U_2), R(U_0)) &= \|\Sigma_k^{-1} V_k^H V_2 G\|_2 \\ &\leq \sigma_k^{-1} \|G\|_2 \operatorname{dist}(R(V_2), R(V_0)) \leq \sigma_k^{-1} \|G\|_2 , \\ \operatorname{dist}(R(V_2), R(V_0)) &= \|\Sigma_k^{-1} U_k^H (U_1 F + U_2 G)\|_2 \\ &\leq \sigma_k^{-1} (\|F\|_2 \operatorname{dist}(R(U_1), R(U_0)) \\ &\quad + \|G\|_2 \operatorname{dist}(R(U_2), R(U_0))) \leq \sigma_k^{-1} \left\| \begin{bmatrix} F \\ G \end{bmatrix} \right\|_2 \end{aligned}$$

for the RR URV decomposition and

$$\begin{aligned}\operatorname{dist}(R(U_2), R(U_0)) &= \|\Sigma_k^{-1} V_k^H (V_1 H^H + V_2 E^H)\|_2 \\ &\leqslant \sigma_k^{-1}(\|H\|_2 \operatorname{dist}(R(V_1), R(V_0)) \\ &\quad + \|E\|_2 \operatorname{dist}(R(V_2), R(V_0))) \leqslant \sigma_k^{-1}\|[H; E]\|_2 , \\ \operatorname{dist}(R(V_2), R(V_0)) &= \|\Sigma_k^{-1} U_k^H U_2 E\|_2 \\ &\leqslant \sigma_k^{-1}\|E\|_2 \operatorname{dist}(R(U_2), R(U_0)) \leqslant \sigma_k^{-1}\|E\|_2\end{aligned}$$

for the RR ULV decomposition.

Proof

Since $C = U_k \Sigma_k V_k^H + U_0 \Sigma_0 V_0^H$, it follows that $CV_2 = U_k \Sigma_k V_k^H V_2 + U_0 \Sigma_0 V_0^H V_2$. Using this and [7, cor. 2.4], we have:

$$\begin{aligned}\operatorname{dist}(R(V_2), R(V_0)) &= \|V_k^H V_2\|_2 = \|\Sigma_k^{-1} U_k^H U_k \Sigma_k V_k^H V_2\|_2 \\ &= \|\Sigma_k^{-1} U_k^H (CV_2 - U_0 \Sigma_0 V_0^H V_2)\|_2 \\ &= \|\Sigma_k^{-1} U_k^H CV_2\|_2 . \qquad (12)\end{aligned}$$

Substituting the RR URV decomposition (3) of C into (12) yields:

$$\begin{aligned}\operatorname{dist}(R(V_2), R(V_0)) &= \|\Sigma_k^{-1} U_k^H [U_1; U_2] \begin{bmatrix} R & F \\ 0 & G \end{bmatrix} \begin{bmatrix} V_1^H \\ V_2^H \end{bmatrix} V_2\|_2 \\ &= \|\Sigma_k^{-1} U_k^H [U_1; U_2] \begin{bmatrix} F \\ G \end{bmatrix}\|_2 = \|\Sigma_k^{-1} U_k^H (U_1 F + U_2 G)\|_2 \\ &\leqslant \|\Sigma_k^{-1} U_k^H U_1 F\|_2 + \|\Sigma_k^{-1} U_k^H U_2 G\|_2 \\ &\leqslant \sigma_k^{-1}(\|F\|_2 \operatorname{dist}(R(U_1), R(U_0)) + \|G\|_2 \operatorname{dist}(R(U_2), R(U_0))) \\ &\leqslant \sigma_k^{-1}\|\begin{bmatrix} F \\ G \end{bmatrix}\|_2 .\end{aligned}$$

To prove the second bound, we use the equality $C^H U_2 = V_k \Sigma_k U_k^H U_2 + V_0 \Sigma_0 U_0^H U_2$ and [7, cor. 2.4]. This yields:

$$\begin{aligned}\operatorname{dist}(R(U_2), R(U_0)) &= \|U_k^H U_2\|_2 = \|\Sigma_k^{-1} V_k^H V_k \Sigma_k U_k^H U_2\|_2 \\ &= \|\Sigma_k^{-1} V_k^H (C^H U_2 - V_0 \Sigma_0 U_0^H U_2)\|_2 \\ &= \|\Sigma_k^{-1} V_k^H C^H U_2\|_2 . \qquad (13)\end{aligned}$$

Substituting the RR URV decomposition (3) of C into (13) yields:

$$\begin{aligned}\operatorname{dist}(R(U_2), R(U_0)) &= \|\Sigma_k^{-1} V_k^H [V_1; V_2] \begin{bmatrix} R^H & 0 \\ F^H & G^H \end{bmatrix} \begin{bmatrix} U_1^H \\ U_2^H \end{bmatrix} U_2 \|_2 \\ &= \|\Sigma_k^{-1} V_k^H [V_1; V_2] \begin{bmatrix} 0 \\ G^H \end{bmatrix} \|_2 = \|\Sigma_k^{-1} V_k^H V_2 G^H\|_2 \\ &\leqslant \sigma_k^{-1} \|G\|_2 \operatorname{dist}(R(V_2), R(V_0)) \leqslant \sigma_k^{-1} \|G\|_2 .\end{aligned}$$

The bounds for the RR ULV decomposition are proven similarly, by substituting the RR ULV decomposition (3) of C into (12) and (13). □

As shown in theorem 2, the distance between $R(U_2)$ and $R(U_0)$ and between $R(V_2)$ and $R(V_0)$ depends on the quality of the estimates w_i, which is reflected by the size of the elements of F and H, and on the gap σ_{k+1}/σ_k between the signal singular values $\sigma_1, \ldots, \sigma_k$ and the noise singular values $\sigma_{k+1}, \ldots, \sigma_n$. Indeed, applying the interlacing theorem for singular values [7, cor. 8.3.3], [21, thm. 2.4], we have that $\|[F^H; G^H]^H\|_2$, $\|E\|_2$, $\|G\|_2$ and $\|[H; E]\|_2 \geqslant \sigma_{k+1}$ so that $\sigma_k^{-1}\|[F^H; G^H]^H\|_2$, $\sigma_k^{-1}\|E\|_2$, $\sigma_k^{-1}\|G\|_2$ and $\sigma_k^{-1}\|[H; E]\|_2 \geqslant \sigma_{k+1}/\sigma_k$. Under the assumptions of theorem 1, $V_k^H V_2 = 0$, $F = 0$, $U_k^H U_1 = 0$ and $H = 0$ so that all distances in theorem 2 are zero, proving that $R(U_2) = R(U_0)$ and $R(V_2) = R(V_0)$.

The perturbation bounds for the singular subspaces derived in theorem 2 are similar to the ones given in [3] for the RR QR factorization. Observe however that the latter factorization can be considered as a special (suboptimal) RR URV decomposition since you can write the RR QR factorization $C\Pi = QR$ of a matrix $C_{m\times n}$ as:

$$C = QR_{n\times n}\Pi^H = U \begin{bmatrix} R & F \\ 0 & G \end{bmatrix} V^H .$$

In RR QR decompositions the unitary matrix V is "restricted" to be a permutation matrix Π and is therefore suboptimal. This strongly suggests that the newly proposed decomposition should be a more useful alternative for the SVD (compared to RR QR) resulting in tighter upper and lower bounds for the singular values than the ones given in theorem 2. Up to now, these theorems have not yet been proved.

3. TLS algorithm

3.1. THE ONE-DIMENSIONAL TLS PROBLEM

Consider

$$Ax \approx b, \quad A \in \mathbb{C}^{m\times(n-1)}, b \in \mathbb{C}^m, m \geqslant n, \tag{14}$$

where k is the numerical rank of $[A; b]$ and $k<n$. Then, if $V_2 = [v_{k+1}, \ldots, v_n]$ is a basis of the right singular subspace of $[A; b]$ corresponding to its $n-k$ smallest singular values, the minimum norm TLS solution is computed as follows (see [19]). Compute a unitary matrix Q such that

$$V_2 Q = [v_{k+1}, \ldots, v_n] Q = \begin{bmatrix} Y & z \\ 0 & \gamma \end{bmatrix}_1 . \tag{15}$$

If $\gamma \neq 0$ then the TLS solution $\hat{x}$ is given by:

$$\hat{x} = -\frac{z}{\gamma} . \tag{16}$$

Using the RR URV or ULV algorithm this solution can be computed more efficiently by avoiding the *explicit* computation of the approximate null space basis V_2. Indeed, V_2 is given by:

$$\begin{aligned} V_2 &= V \begin{bmatrix} 0 \\ I_{n-k} \end{bmatrix} = \prod_{j=k+1}^{n} \hat{Q}_j \begin{bmatrix} 0 \\ I_{n-k} \end{bmatrix} \\ &= \left[\prod_{j=k+1}^{n} \hat{Q}_j \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \begin{matrix} k \\ 1 \\ n-k-1 \end{matrix}, \ldots, \hat{Q}_n \hat{Q}_{n-1} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \begin{matrix} n-2 \\ 1 \\ 0 \end{matrix}, \hat{Q}_n \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{matrix} n-1 \\ 1 \end{matrix} \right], \end{aligned} \tag{17}$$

with

$$\hat{Q}_i = \begin{bmatrix} Q_i & 0 \\ 0 & I_{n-i} \end{bmatrix}$$

and Q_i as specified by the RR URV or ULV algorithm. Defining

$$w_i = Q_i \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{matrix} i-1 \\ 1 \end{matrix},$$

eq. (17) yields:

$$V_2 = \left[\prod_{j=k+2}^{n} \hat{Q}_j \begin{bmatrix} w_{k+1} \\ 0 \end{bmatrix} \begin{matrix} k+1 \\ n-k-1 \end{matrix}, \ldots, \hat{Q}_n \begin{bmatrix} w_{n-1} \\ 0 \end{bmatrix} \begin{matrix} n-1 \\ 1 \end{matrix}, w_n \right]. \tag{18}$$

Observe that $w_i = \mathrm{est}(v_{\min}(R_i))$, as computed in the RR URV algorithm, so that these vectors need not be recomputed here provided they have been stored in memory for later use.

Now, the trick is that we do not need to compute first all columns of V_2 explicitly for obtaining the TLS solution. What we try to do is to compute immediately a basis of the form $\begin{bmatrix} Y & z \\ 0 & \gamma \end{bmatrix}$, then we do not need to compute the vectors $\begin{bmatrix} Y \\ 0 \end{bmatrix}$ but only the last vector $\begin{bmatrix} z \\ \gamma \end{bmatrix}$ in order to obtain the TLS solution of the one-dimensional TLS problem. This is done as follows. Consider the first two columns of the basis (18):

$$\prod_{j=k+2}^{n} \hat{Q}_j \begin{bmatrix} w_{k+1} \\ 0 \end{bmatrix} \begin{matrix} k+1 \\ n-k-1 \end{matrix}, \quad \prod_{j=k+3}^{n} \hat{Q}_j \begin{bmatrix} w_{k+2} \\ 0 \end{bmatrix} \begin{matrix} k+2 \\ n-k-2 \end{matrix}.$$

Now computing:

$$\hat{Q}_{k+2}\begin{bmatrix} w_{k+1} \\ 0 \end{bmatrix}\begin{matrix} k+1 \\ n-k-1 \end{matrix} = \begin{bmatrix} w_{k+1}^{(1)} \\ * \\ 0 \end{bmatrix}\begin{matrix} k+1 \\ 1 \\ n-k-2 \end{matrix}$$

implies that one possibly nonzero element is created at position $k+2$. Apply a plane rotation $G_2^{(1)}$ to make this element again zero by operating on the modified column and the next selected one:

$$\begin{matrix} k+1 \\ 1 \\ n-k-2 \end{matrix}\begin{bmatrix} \begin{pmatrix} w_{k+1}^{(1)} \\ * \end{pmatrix} & w_{k+2} \\ 0 & 0 \end{bmatrix} G_2^{(1)} = \begin{matrix} k+1 \\ 1 \\ n-k-2 \end{matrix}\begin{bmatrix} \begin{pmatrix} w_{k+1}^{(2)} \\ 0 \end{pmatrix} & w_{k+2}^{(1)} \\ 0 & 0 \end{bmatrix}. \quad (19)$$

Since each Q_i is defined by the product of plane rotations $Q_i = J_i^{(1)} \ldots J_i^{(i-1)}$, we can prove the following. If we would compute $\prod_{j=k+3}^{n} \hat{Q}_j \begin{bmatrix} w_{k+1}^{(2)} \\ 0 \end{bmatrix}$ explicitly, we will obtain a vector of the form $\begin{bmatrix} * \\ 0 \end{bmatrix}_1^{n-1}$. So, a zero will be maintained in the last entry. Such a basis vector is of no value in computing the TLS solution (see (15–16)) and therefore need not be computed saving a lot of computation time. Indeed, computing

$$\begin{bmatrix} w_{k+1}^{(3)} \\ 0 \end{bmatrix}\begin{matrix} k+2 \\ n-k-2 \end{matrix} = \hat{Q}_{k+3}\begin{bmatrix} w_{k+1}^{(2)} \\ 0 \end{bmatrix}$$

makes the $(k+2)$th entry nonzero. Computing then

$$\begin{bmatrix} w_{k+1}^{(4)} \\ 0 \end{bmatrix}\begin{matrix} k+3 \end{matrix} = \hat{Q}_{k+4}\begin{bmatrix} w_{k+1}^{(3)} \\ 0 \end{bmatrix}$$

makes the $(k+3)$th entry nonzero, etc. So, computing finally

$$\begin{bmatrix} w_{k+1}^{(n-k)} \\ 0 \end{bmatrix}_1 = \hat{Q}_n \begin{bmatrix} w_{k+1}^{(n-k+1)} \\ 0 \\ 0 \end{bmatrix}$$

makes the $(n-1)$th entry nonzero and implies that the nth entry remains zero.

Now consider the third column in (18) and the second one, modified by applying $G_2^{(1)}$ on it (see (19)), and apply the same trick, i.e., compute:

$$\hat{Q}_{k+3}\begin{bmatrix} w_{k+2}^{(1)} \\ 0 \end{bmatrix}\begin{matrix} k+2 \\ n-k-2 \end{matrix} = \begin{bmatrix} w_{k+2}^{(2)} \\ * \\ 0 \end{bmatrix}\begin{matrix} k+2 \\ 1 \\ n-k-3 \end{matrix},$$

creating a possibly nonzero element in the $(k+3)$th entry. Restore the zero by applying a plane rotation $G_2^{(2)}$:

$$\begin{matrix} k+2 \\ 1 \\ n-k-3 \end{matrix}\begin{bmatrix} \begin{pmatrix} w_{k+2}^{(2)} \\ * \end{pmatrix} & w_{k+3} \\ 0 & 0 \end{bmatrix} G_2^{(2)} = \begin{bmatrix} \begin{pmatrix} w_{k+2}^{(3)} \\ 0 \end{pmatrix} & w_{k+3}^{(1)} \\ 0 & 0 \end{bmatrix}.$$

Since the premultiplied matrices $\hat{Q}_n \ldots \hat{Q}_{k+4}$ are the same for both vectors

$$\begin{bmatrix} \begin{pmatrix} w_{k+2}^{(2)} \\ * \\ 0 \end{pmatrix} & w_{k+3} \\ 0 & 0 \end{bmatrix},$$

we are allowed to perform a basis transformation because we are working in the same transformed subspace.

Continue in the same way and consider the last two columns of V_2 in (18): $\begin{bmatrix} w_{n-1} \\ 0 \end{bmatrix}$ is changed to $\begin{bmatrix} w_{n-1}^{(1)} \\ 0 \end{bmatrix}$, due to the applied plane rotation $G_2^{(n-k-2)}$. Compute now

$$\begin{bmatrix} w_{n-1}^{(2)} \\ * \end{bmatrix} = \hat{Q}_n \begin{bmatrix} w_{n-1}^{(1)} \\ 0 \end{bmatrix}$$

and apply the plane rotation $G_2^{(n-k-1)}$ to restore the zero in the nth entry:

$$\left[\begin{pmatrix} w_{n-1}^{(2)} \\ * \end{pmatrix}, w_n \right] G_2^{(n-k-1)} = \left[\begin{pmatrix} w_{n-1}^{(3)} \\ 0 \end{pmatrix}, w_n^{(1)} \right]. \tag{20}$$

Provided $w_{n,n}^{(1)} \neq 0$, we obtain directly the minimum norm TLS solution $\hat{x}$ of (14) from (20):

$$\begin{bmatrix} \hat{x} \\ -1 \end{bmatrix} = -\frac{1}{w_{n,n}^{(1)}} w_n^{(1)}. \tag{21}$$

3.2. THE MULTIDIMENSIONAL TLS PROBLEM

If the TLS problem had d right-hand sides, i.e.

$$AX \approx B \quad A \in \mathbb{C}^{m \times (n-d)}, B \in \mathbb{C}^{m \times d}, m \geqslant n, \tag{22}$$

where k is the numerical rank of $[A; B]$ and $k \leqslant n-d$, the procedure explained in the previous section should be applied d times. At each round $i = 1, \ldots, d$ the last $i-1$ columns remain unchanged and only the first $(n-k)-i+1$ modified columns of V_2 need to be considered. Hence, having applied the procedure of section 3.1 once, we now apply the procedure to the first $n-k-1$ modified basis vectors:

$$\prod_{j=k+3}^{n} \hat{Q}_j \begin{bmatrix} w_{k+1}^{(2)} \\ 0 \end{bmatrix}, \ldots, \hat{Q}_n \begin{bmatrix} w_{n-2}^{(3)} \\ 0 \end{bmatrix}, \begin{bmatrix} w_{n-1}^{(3)} \\ 0 \end{bmatrix}.$$

Observe that the last elements of the above vectors are zero. Now compute:

$$\hat{Q}_{k+3} \begin{bmatrix} w_{k+1}^{(2)} \\ 0 \end{bmatrix} = J_{k+3}^{(1)} \ldots J_{k+3}^{(k+2)} \begin{bmatrix} w_{k+1}^{(2)} \\ 0 \end{bmatrix} \begin{matrix} k+1 \\ n-k-1 \end{matrix} = \begin{bmatrix} w_{k+1}^{(3)} \\ * \\ 0 \end{bmatrix} \begin{matrix} k+1 \\ 1 \\ n-k-2 \end{matrix}.$$

Since $J_{k+3}^{(k+2)}$ operates on rows $k+2$ and $k+3$, the result operation is zero since these elements are zero. $J_{k+3}^{(k+1)}$ operates on rows $k+1$ and $k+2$ creating a nonzero in position $k+2$ if the $(k+1)$th element is nonzero. So, the result operation is again that one element is made nonzero. By applying an appropriate plane rotation $G_2^{(1,2)}$ (the second superscript refers here to the number of times the procedure of section 3.1 is applied), the zero pattern can be restored:

$$\begin{matrix} k+1 \\ 1 \\ n-k-2 \end{matrix}\begin{bmatrix} \begin{pmatrix} w_{k+1}^{(3)} \\ * \end{pmatrix} & w_{k+2}^{(3)} \\ 0 & 0 \end{bmatrix} G_2^{(1,2)} = \begin{bmatrix} \begin{pmatrix} w_{k+1}^{(4)} \\ 0 \end{pmatrix} & w_{k+2}^{(4)} \\ 0 & 0 \end{bmatrix}.$$

Continue in the same way until the computation of:

$$\hat{Q}_n \begin{bmatrix} w_{n-2}^{(4)} \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} w_{n-2}^{(5)} \\ * \\ 0 \end{bmatrix}$$

($w_{n-2}^{(3)}$ was modified to $w_{n-2}^{(4)}$ by a previously applied plane rotation $G_2^{(n-k-3,2)}$). Finally, applying an appropriate plane rotation restores the zero in the $(n-1)$th entry:

$$\begin{bmatrix} \begin{pmatrix} w_{n-2}^{(5)} \\ * \end{pmatrix} & w_{n-2}^{(3)} \\ 0 & 0 \end{bmatrix} G_2^{(n-k-2,2)} = \begin{bmatrix} \begin{pmatrix} w_{n-2}^{(6)} \\ 0 \end{pmatrix} & w_{n-1}^{(4)} \\ 0 & 0 \end{bmatrix}.$$

If $d = 2$, the minimum norm TLS solution $\hat{X}$ of (22) is immediately obtained from:

$$\begin{bmatrix} \begin{pmatrix} w_{n-1}^{(4)} \\ 0 \end{pmatrix}, w_n^{(1)} \end{bmatrix} = \begin{matrix} Z \\ \Gamma \end{matrix}\begin{matrix} n-2 \\ 2 \end{matrix} \Rightarrow \hat{X} = -Z\Gamma^{-1},$$

where Γ is already in upper triangular form. For larger d, the same procedure is repeated.

3.3. OUTLINE OF THE ALGORITHM

Overall we have the following algorithm for solving TLS problems of the form (22) (we now no longer consider vectors $w_j^{(i)}$ but transformations of the same vector w_j which are stored in the same location):

ALGORITHM 3: MULTIDIMENSIONAL TLS SOLUTION $\hat{X}$ OF $AX \approx B$

Given: • $[A; B] \in \mathbb{C}^{m \times n}$, $m \geq n$ and $B \in \mathbb{C}^{m \times d}$
• *tol*

Step 1: $C \leftarrow [A; B]$; *wantf* = .false.; *wantu* = .false.; *wantv* = .false.; $k_{\min} = 0$; $k_{\max} = n - d$

Step 2: (a) Apply RR URV algorithm 1 or RR ULV algorithm 2.

(b) If $\begin{cases} \text{est}(v_{\min}(R_i)), i = k+1, \ldots, n \text{ are not available} \\ \text{or} \\ \text{RR ULV algorithm 2 is used} \end{cases}$ then

For $i = k+1, \ldots, n$ do

$$w_i \leftarrow Q_i \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{matrix} i-1 \\ 1 \end{matrix}$$

end for

end if

Step 3: For $j = 1, \ldots, d$ do

If $k < n-j$ then

For $i = k+1, \ldots, n-j$ do

(a) transform: $\begin{bmatrix} w_i \\ * \\ 0 \end{bmatrix} \begin{matrix} i \\ 1 \\ j-1 \end{matrix} \leftarrow Q_{i+j} \begin{bmatrix} w_i \\ 0 \end{bmatrix} \begin{matrix} i \\ j \end{matrix}$

(b) compute a plane rotation $G_2^{(i,j)}$ such that

$$\left[\begin{pmatrix} w_i \\ 0 \end{pmatrix}, w_{i+1}\right] \leftarrow \left[\begin{pmatrix} w_i \\ * \end{pmatrix}, w_{i+1}\right] G_2^{(i,j)}$$

end for;

end if;

end for

Step 4: Set $\begin{bmatrix} Z \\ \Gamma \end{bmatrix}_d \leftarrow \left[\begin{pmatrix} w_{n-d+1} \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \cdots, \begin{pmatrix} w_{n-1} \\ \\ \\ 0 \end{pmatrix}, w_n\right]$ where Γ is upper triangular.

Step 5: If Γ is singular then quit {no generic TLS solution}

Step 6: Solve $\hat{X}\Gamma = -Z$

END

By definition, $Q_{i+j} = J_{i+j}^{(1)} \ldots J_{i+j}^{(i+j-1)}$ but in computing step 3(a) the plane rotations $J_{i+j}^{(i+1)}, \ldots, J_{i+j}^{(i+j-1)}$ need not be performed since they operate on zero elements, so the result operation is zero. $J_{i+j}^{(i)}$ operates on rows i and $i+1$ creating a nonzero $*$ in position $i+1$ (which is then zeroed out in the next step). So, the result $Q_{i+j}\begin{bmatrix} w_i \\ 0 \end{bmatrix}$ is obtained by applying effectively i plane rotations $J_{i+j}^{(1)} \ldots J_{i+j}^{(i)}$, operating on consecutive elements of the postmultiplied vector, so that at most one zero element will be made nonzero. Our procedure discussed in sections 3.1 and 3.2 exploits this structure. However, if L *refinement* steps are performed in the RR URV algorithm, then each Q_{i+j} consists of:

$$Q_{i+j} = J_{i+j}^{(1)} \ldots J_{i+j}^{(i+j-1)} Q_{i+j}^{(1)} \ldots Q_{i+j}^{(L)}.$$

Here, $Q_{i+j}^{(l)}$ has the form $S_{i+j}^{(i+j-1,i+j)} \ldots S_{i+j}^{(1,i+j)}$ where $S_{i+j}^{(s,t)}$ denotes a plane rotation of order $i+j$ applied to the elements at positions s and t of a postmultiplied vector $\begin{bmatrix} w_i \\ 0 \end{bmatrix}$. These plane rotations do not operate on consecutive elements and will make all elements with index $> i$ nonzero so that our procedure explained in sections 3.1 and 3.2 is no longer applicable. Therefore, the RR URV algorithm, as used in step 2 of algorithm 3, must be applied without refinement steps. Remember that we can always refine our RR URV decomposition by starting from a better estimate of $v_{\min}(R_i)$ in each step of algorithm 1, e.g. by performing more inverse iteration steps.

The $\sum_{i=k}^{n-1} i = \frac{1}{2}(n(n-1) - k(k-1))$ plane rotations $Q_n = J_n^{(1)} \ldots J_n^{(n-1)}, \ldots, Q_{k+1} = J_{k+1}^{(1)} \ldots J_{k+1}^{(k)}$, as computed in the RR URV or ULV algorithm, need to be *stored compactly* in memory for further use in step 3 of the TLS algorithm 3. Therefore, the lower triangular part of the data matrix C can be used, i.e. all elements below the main diagonal (the upper triangular part stores the lastly computed triangular matrix $\hat{R}_{k+1}$ or $\hat{L}_{k+1}$ of the RR URV or ULV algorithm). Usually, each plane rotation is represented by 2 numbers c and s [7, section 5.1.9] but, if storage is critical, each plane rotation can be represented by only one number, as described by Stewart [16]. Of course, this compact way of storage involves some extra computational work to recompute c and s from the stored number when this plane rotation is to be applied.

If the RR URV algorithm is used, we also need to store the $\text{est}(v_{\min}(R_i))$. If not, we can recompute these in step 2(b) but this requires of course extra flops. In the RR ULV algorithm, the $\text{est}(u_{\min}(L_i))$ are computed but these estimates are not needed to compute the TLS solution. Hence they need not be stored but here, extra flops are required to perform step 2(b).

Finally, we investigate the computational work of algorithm 3. Assuming that ordinary Givens rotations are used, the total amount of flops is

$$2n^2\left(m - \frac{n}{3}\right) + \left(\frac{2}{3}K + 2\right)(n^3 - k^3) + J(n-k)(n+k) - 5nd^2 + 2d^3 \text{ flops} . \tag{23}$$

K depends on the method used for estimating $v_{\min}(R_i)$ (resp., $u_{\min}(L_i)$) at each i of algorithm 1 (resp., 2) (see section 2.2). For instance, if inverse iteration is used then K is the number of iteration steps carried out. $J = 6d$ if the RR URV algorithm is used provided all $\text{est}(v_{\min}(R_i))$ are stored, else $J = 6d + 3$ since also step 2(b) is to be performed. If fast Givens rotations are used, algorithm 3 requires

$$2n^2\left(m - \frac{n}{3}\right) + \left(\frac{2}{3}K + \frac{4}{3}\right)(n^3 - k^3) + J_F(n-k)(n+k) - 3nd^2 + \frac{4}{3}d^3 \text{ flops} , \tag{24}$$

with $J_F = 4d$ (resp., $4d + 2$) if the RR URV (resp., ULV) algorithm is used. For close-to-maximal rank TLS problems in which $d \ll n$ and $k \approx n - d$, the last terms are negligible and the first term, the initial QR (resp., QL) factorization, is dominant.

3.4. NONGENERIC TLS PROBLEMS

If it turns out that Γ in step 5 of algorithm 3 is *singular*, then the TLS problem fails to have a finite solution. These problems are called *nongeneric* (see [19,17,21] for more details). In order to make these problems solvable, additional constraints are imposed on the TLS solution and a nongeneric TLS solution is computed. Therefore, the numerical rank k of $[A; B]$ need be lowered such that the noise subspace is enlarged and additional basis vectors $y_{k+1}, \ldots, y_{k_0}$ (where k is the newly computed rank and k_0 is the previous value of the numerical rank k) are added. In order to solve these nongeneric TLS problems, replace step 5 in algorithm 3 by the following:

Step 5: While Γ singular do

(a) $tol \leftarrow \|R_{k_0} w_{k_0}\| + \epsilon$ or $\|L_{k_0}^H w_{k_0}\| + \epsilon$; $i \leftarrow k_0$

(b) apply RR algorithm 1 or 2, steps 3 and 4.

(c) apply TLS algorithm 3, steps 2(b) and 3, with n replaced by k_0.

(d) set $h \leftarrow \min(d, k_0 - k)$;

for $i = k_0 - h + 1, \ldots, k_0$ do

$$y_i \leftarrow \hat{Q}_n \ldots \hat{Q}_{k_0+1} \begin{bmatrix} w_i \\ 0 \end{bmatrix} \begin{matrix} i \\ n-i \end{matrix} \text{ with } \hat{Q}_i = \begin{bmatrix} Q_i & 0 \\ 0 & I_{n-i} \end{bmatrix}$$

end for

(e) compute a unitary matrix $S^{(k)}$ such that

$$\left[\begin{pmatrix} y_{k_0-h+1} \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \cdots, \begin{pmatrix} y_{k_0-1} \\ \\ \\ 0 \end{pmatrix}, y_{k_0}, \begin{pmatrix} Z \\ \\ \Gamma \\ \\ \end{pmatrix} \right] S^{(k)} = \begin{bmatrix} \hat{Y} & \hat{Z} \\ 0 & \hat{\Gamma} \end{bmatrix} \begin{matrix} n-d \\ d \end{matrix}$$
$$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad h \quad d$$

and $\hat{\Gamma}$ upper triangular

(f) set $\begin{bmatrix} Z \\ \Gamma \end{bmatrix} \leftarrow \begin{bmatrix} \hat{Z} \\ \hat{\Gamma} \end{bmatrix}$

end while

After updating the RR URV (or ULV) decomposition by lowering the rank from k_0 to k, we apply again our procedure of sections 3.1 and 3.2, summarized in the steps 2(b) and 3 of algorithm 3, to the newly computed estimates $w_i = \text{est}(v_{\min}(R_i))$ or $\text{est}(u_{\min}(L_i))$, $i = k+1, \ldots, k_0$ for computing the additional

basis vectors $y_{k_0-h+1}, \ldots, y_{k_0}$. These vectors are then transformed to basis vectors of the enlarged noise subspace of $[A; B]$ (step 5(d)) and then combined with the d previously computed basis vectors $\begin{bmatrix} Z \\ \Gamma \end{bmatrix}$ in order to obtain a new basis $\begin{bmatrix} \hat{Z} \\ \hat{\Gamma} \end{bmatrix}$ until $\hat{\Gamma}$ is nonsingular (step 5(e)). Here, we proceed as described in the classical TLS algorithm [19, algor. 4.1], steps 3 and 4, but, since the two sets of basis vectors have a triangular trailing matrix, the unitary transformation $S^{(k)}$ can be optimized by making use of the zero pattern.

ϵ is a parameter that takes into account the numerical multiplicity of the minimal singular value of R_{k_0} or L_{k_0}: all singular values of R_{k_0} or L_{k_0} which approach its minimal singular value to a distance $\leqslant \epsilon$ are considered to coincide with this value.

3.5. UPDATING

In certain TLS problems $AX \approx B$, the data matrix $[A; B]$ changes at each time instant such that the solution of the corresponding problem must be updated correspondingly. This happens for instance in signal processing applications and in the estimation of parameters of nonstationary systems that slowly vary with time, space or frequency. In these problems good initial estimates of $v_{\min}(R_i)$ or $u_{\min}(L_i)$ are available, namely the estimates computed in the previous problem, which can be further improved by inverse iteration or by using an adaptive condition estimator, see e.g. [9,5,10]. Therefore, the use of algorithm 3 is recommended for solving these slowly varying TLS problems, especially when the changes in the data matrix are of small norm but still of full rank, e.g. when the data are processed in non-overlapping blocks. This implies that in each problem the re-factorization of the new data matrix must be started from scratch.

Only in situations where the changes in the data matrix are of rank one, e.g. when a new row is appended successively, efficient rank one updating RR URV or ULV algorithms, as described by Stewart [13,15,16], can be applied for computing the new RR decomposition from the previous one and speed up the computation time. These algorithms can be used for downdating as well, e.g. when an existing row is deleted, provided the ordinary Givens rotations, which reduce the new data matrix to triangular form, are replaced by appropriate hyperbolic rotations and the test criterion [13, eq. (5.1)] is replaced by the test $\sqrt{v^2 - \|y\|^2} \leqslant tol$. However, these rank one updating RR algorithms can not be used in combination with algorithm 3 since then our procedure, described in sections 3.1 and 3.2, is no longer applicable. Moreover, these algorithms require the computation of

$$z^H V,$$

where z represents either the appended or deleted row and V equals the right unitary matrix in the RR decomposition (3) of $C = [A; B]$. This implies that V must be computed explicitly (or else, all unitary transformations Q_i, used during the entire up- or downdating process, must be kept in memory which is of course an impossible task). Since $V = [V_1; V_2]$ is known, a basis V_2 of the approximate $(n-k)$-dimen-
k

sional null space is also available so that we better compute here the TLS solution $\hat{X}$ in the classical way, i.e. compute a unitary matrix Q such that

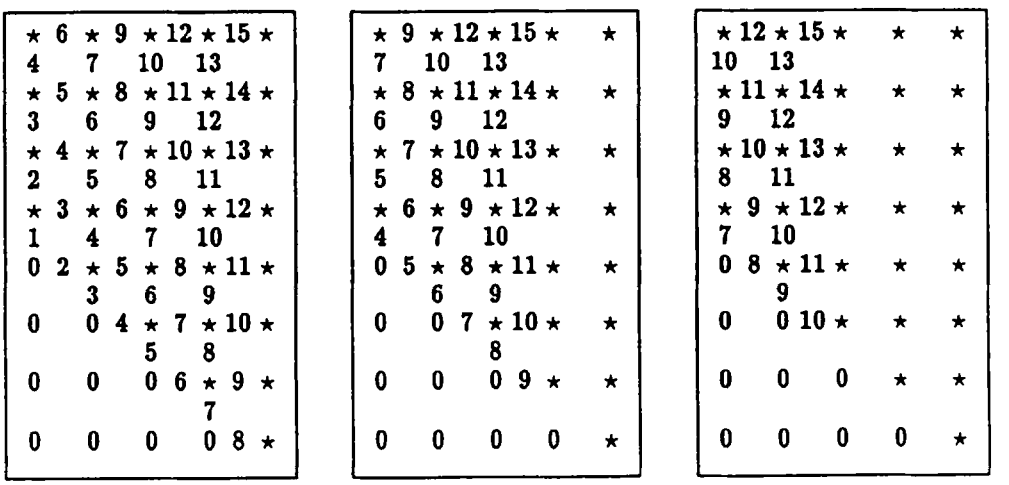
$$V_2 Q = \begin{bmatrix} Y & Z \\ 0 & \Gamma \end{bmatrix} \begin{matrix} n-d \\ d \end{matrix} \quad \text{and } \Gamma\, d \times d \text{ upper triangular .} \tag{25}$$

$$\text{If } \Gamma \text{ is nonsingular then } \hat{X} = -Z\Gamma^{-1} . \tag{26}$$

3.6. PARALLEL IMPLEMENTATION

In this section we show how steps 3–4 of algorithm 3 can be parallelized in order to obtain the TLS solution very efficiently. To describe these parallel algorithms, we will use the method of *precedence diagrams*, as described in [13, section 6]. These diagrams show the order in which operations can be performed consistently so that operations can be assigned to processors in such a way that no two simultaneous operations are performed by the same processor. In our case, the assignments will amount to making each processor responsible for a diagonal of the matrix and its neighbours.

We begin with applying plane rotations on the vector w_{k+1} to the left. Specifically we desire to compute $J_{i+j}^{(1)} \ldots J_{i+j}^{(i)} \begin{bmatrix} w_l \\ 0 \end{bmatrix}_j^l$ where $J_{i+j}^{(l)}$ combines the entries l and $l+1$ of $\begin{bmatrix} w_l \\ 0 \end{bmatrix}$. Figure 1 shows a precedence diagram of this computation for a TLS problem $A_{8\times(8-d)}X \approx B_{8\times d}$ where the computed rank k of $[A;B]$ is 3 and $d \leqslant 5$. The stars represent elements of the vectors $w_{k+1}, \ldots, w_n$. The bottom triangle is filled with zeros. The numbers between any two rows represent the time at which a rota-



(a) $j = 1$ (b) $j = 2$ (c) $j = 3$

Fig. 1. Precedence diagram for solving a TLS problem $A_{8\times(8-d)}X \approx B_{8\times d}$, where the computed rank k of $[A;B]$ is 3 and $d \leqslant 5$, with TLS algoritm 3, steps 3–4. (a) $j = 1, d \geqslant 1$ (b) $j = 2, d \geqslant 2$ and (c) $j = 3, d \geqslant 3$.

tion can be applied to the two corresponding elements. These numbers are called *ticks* (see also [13]). The increasing ticks in each column, from bottom to top, reflect the fact that $J_{i+j}^{(l)}$ must be applied before $J_{i+j}^{(l-1)}$.

Consider now the application of the plane rotation $G_2^{(i,j)}$ to the right which combines two consecutive column vectors w_i and w_{i+1}. These computations can be represented in the same precedence diagram and be combined with the left plane rotations in a suitable way, as shown in fig. 1. A number between two elements in a row represents a plane rotation to the right; between two stars in a column a rotation to the left. From this diagram it is seen that steps 3–4 of the TLS algorithm 3 can be carried out in $3(n-k-1)+k$ ticks. This number is independent of the number of right-hand sides d, as can be seen in fig. 1. Indeed, when performing the algorithm for $j=1$ in $3(n-k-1)+k$ ticks we need to wait 3 time units before we can start the operations for $j=2$. For $j=2$, we need no longer to consider the last column, as well as the last row, of our diagram. In fact, we now operate on an $(n-1)\times(n-k-1)$ matrix. This means that the operations for $j=2$ are finished after

$$3+3(n-k-2)+k=3(n-k-1)+k \text{ ticks} .$$

More generally, for $j=l$, $1\leqslant l\leqslant d$, we need to wait $3(l-1)$ time units before we can start the operations. Since we do not have to consider the last $(l-1)$ rows and columns of our diagram, the operations for $j=l$ will be finished after

$$3(l-1)+3(n-k-l-2)+k=3(n-k-1)+k \text{ ticks} .$$

Observe that the ticks increase as we go up a column (and as we go down along a diagonal). On a shared memory system there will be contention along a column as the processors attempt to access the same rotation. In a distributed system the rotation must be passed up the column.

Consider first the one-dimensional TLS problem (14) ($d=1$). Figure 2(a) shows a possible assignment of operations to processors for solving the TLS problem $A_{8\times 7}x\approx b_{8\times 1}$ where the computed rank k of $[A;b]$ is 3. Operations between two diagonal lines are assigned to the same processor. This diagram shows that the solution of the one- dimensional TLS problem can be computed with n processors which perform the $n^2-(k+1)^2$ operations in $3(n-k-1)+k$ ticks. In this configuration, each processor performs about $2(n-k-1)$ operations.

This configuration could be used as well for solving multidimensional TLS problems (22) where $d>1$. If only one processor is assigned to each diagonal, the algorithm can not be performed in $3(n-k-1)+k$ ticks due to the fact that each processor can only perform one operation per time unit. As illustrated in figs. 2(b)–(c) for a TLS problem $A_{8\times(8-d)}X\approx B_{8\times d}$ with $k=3$ and $d\leqslant 5$, this assignment results in longer execution times, which increase linearly with the number of right-hand sides d. Since each processor now performs about $\sum_{j=1}^{d}2(n-k-j)$ operations, the algorithm needs $\sum_{j=1}^{d}2(n-k-j)+(n-d)=2d(n-k-d)+n$ time units to perform its task.

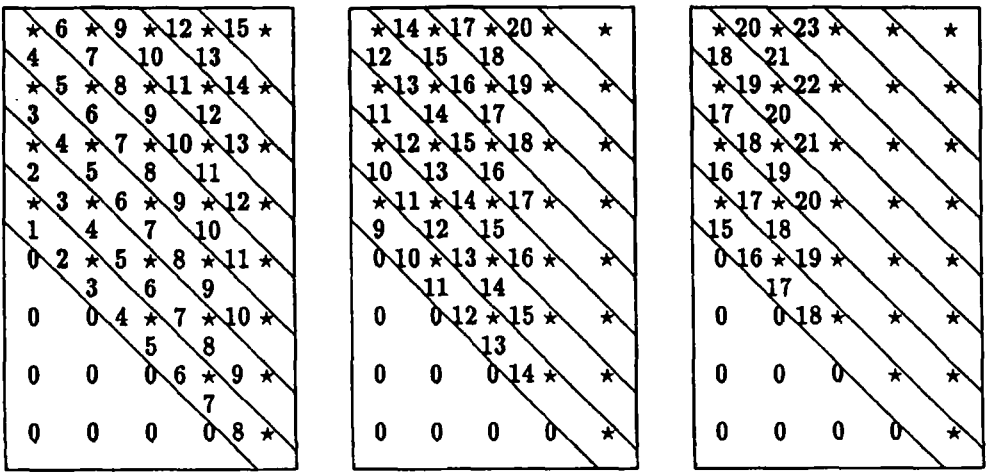


Fig. 2. Assignment of operations to processors for solving a TLS problem $A_{8\times(8-d)}X \approx B_{8\times d}$, where the computed rank k of $[A; B]$ is 3 and $d \leqslant 5$, with TLS algoritm 3, steps 3–4. Operations between two diagonal lines are assigned to the same processor. (a) $j = 1, d \geqslant 1$ (b) $j = 2, d \geqslant 2$ and (c) $j = 3, d \geqslant 3$.

The best execution times are obtained when a configuration is used that computes the TLS solution in $3(n - k - 1) + k$ time units. Therefore, only 3 operations for each j, $1 \leqslant j \leqslant d$, can be assigned to one processor. One operation is either the application of one left plane rotation to 2 consecutive elements of the same vector w_i or the application of one right plane rotation between two elements of the same

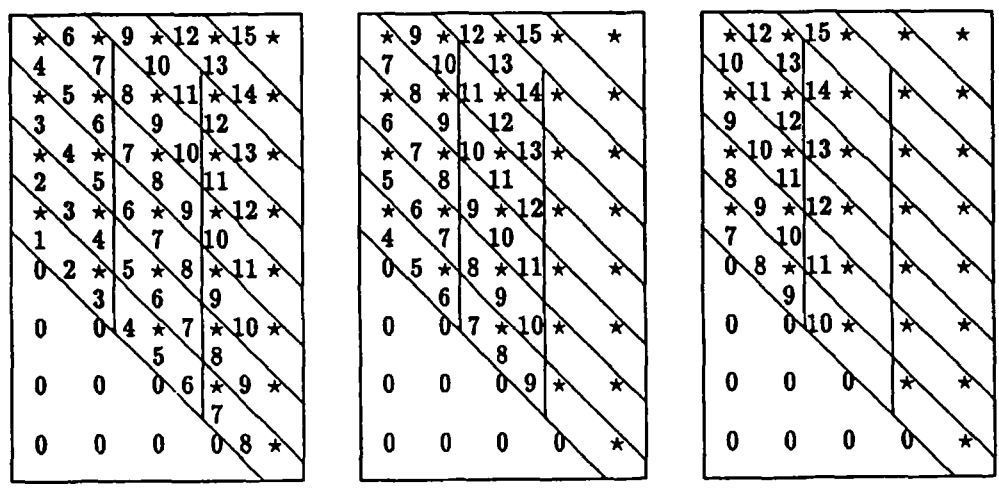


Fig. 3. Assignment of operations to processors for solving a TLS problem $A_{8\times(8-d)}X \approx B_{8\times d}$, where the computed rank k of $[A; B]$ is 3 and $d \leqslant 5$, with TLS algoritm 3, steps 3–4. Operations within the same box are assigned to the same processor. (a) $j = 1, d \geqslant 1$ (b) $j = 2, d \geqslant 2$ and (c) $j = 3, d \geqslant 3$.

row of 2 consecutive vectors w_i and w_{i+1}. Hence, we need about $(n^2-(k+1)^2)/3$ processors. Figure 3 shows a possible assignment of operations to processors that can be used for solving multidimensional TLS problems (22) (where $d>1$) in $3(n-k-1)+k$ ticks. Here, about $\frac{2}{3}(n-k-1)$ processors are to be assigned to each diagonal instead of one in fig. 2. Depending on what is critical, the number of processors used or the execution time of the algorithm, the configuration shown in fig. 2 or 3 will be preferred.

3.7. EVALUATION

3.7.1. Numerical examples

In this section we give a few examples using Pro-Matlab to illustrate the properties of the RR URV decomposition and the application of our RR URV based

Table 1
Comparison of the SVD based TLS solution $\hat{x}$ [7, algor. 12.3.1] and the RR URV based TLS solution $\check{x}$ (algorithm 3) of different sets $A_{25\times 9}x \approx b_{25\times 1}$ for varying quality of the estimates w_i obtained by adding Gaussian noise with varying standard deviation σ_v to the exact $v_{\min}(R_i)$. $m=25$, $n=10$, $d=1$, $\operatorname{rank}([A;b])=7$ and $\sigma([A;b])=\{1, 0.5, 0.2, 0.1, 0.05, 0.02, 0.01, \sigma_8, \sigma_9, \sigma_{10}\}$. $\{\sigma_8, \sigma_9, \sigma_{10}\}$ vary from case to case and are respectively: (a) $\{9\times 10^{-18}, 7\times 10^{-18}, 4\times 10^{-18}\}$, (b) $\{10^{-5}, 10^{-6}, 10^{-7}\}$, (c) $\{10^{-3}, 10^{-4}, 10^{-5}\}$, (d) $\{5\times 10^{-3}, 2\times 10^{-3}, 10^{-3}\}$, (e) $\{9.9\times 10^{-3}, 9.8\times 10^{-3}, 9.7\times 10^{-3}\}$. V_2, V_0, Σ_0, F, G are as defined in theorem 2 for $C=[A;b]$ and $|\cdot|$ denotes the absolute values.

Case	σ_v	$\operatorname{dist}(R(V_2), R(V_0))$	$\sigma_k^{-1}\|[F^H; G^H]^H\|_2$	$\|F\|_2$	$\|\|G\|-\Sigma_0\|_2$	$\|\check{x}-\hat{x}\|/\|\hat{x}\|$
(a)	0.0	2.71e−15	2.58e−14	2.54e−16	4.20e−17	1.33e−15
	e−14	4.47e−14	1.99e−12	1.99e−14	4.20e−17	3.63e−14
	e−10	4.95e−10	1.66e−08	1.66e−10	4.20e−17	5.19e−10
	e−06	4.62e−06	1.77e−04	1.77e−06	4.20e−17	2.45e−06
	e−02	2.87e−02	1.86e+00	1.86e−02	4.20e−17	4.39e−02
(b)	0.0	2.60e−15	1.00e−03	2.05e−16	1.89e−17	2.89e−15
	e−14	4.24e−14	1.00e−03	1.82e−14	1.89e−17	6.64e−14
	e−10	4.00e−10	1.00e−03	2.09e−10	4.98e−16	5.62e−10
	e−06	3.06e−06	1.00e−03	1.88e−06	9.35e−12	2.54e−06
	e−02	2.79e−02	8.51e−01	8.51e−03	9.04e−08	3.52e−02
(c)	0.0	3.88e−15	1.00e−01	1.34e−16	1.76e−17	3.57e−15
	e−14	2.49e−14	1.00e−01	1.12e−14	1.85e−17	2.87e−14
	e−10	3.92e−10	1.00e−01	1.99e−10	1.25e−13	3.28e−10
	e−06	2.75e−06	1.00e−01	1.12e−06	7.90e−10	4.28e−06
	e−02	3.11e−02	2.23e+00	2.23e−02	1.17e−05	3.65e−02
(d)	0.0	5.60e−15	5.00e−01	1.59e−16	1.67e−17	2.73e−15
	e−14	3.30e−14	5.00e−01	1.94e−14	5.31e−17	3.32e−14
	e−10	3.80e−10	5.00e−01	2.27e−10	5.90e−13	2.68e−10
	e−06	2.87e−06	5.00e−01	1.14e−06	5.82e−09	3.61e−06
	e−02	3.05e−02	2.59e+00	2.59e−02	4.64e−05	3.89e−02
(e)	0.0	1.60e−13	9.90e−01	1.37e−16	2.81e−17	1.68e−13
	e−14	2.51e−13	9.90e−01	1.94e−14	2.54e−17	2.44e−13
	e−10	3.11e−10	9.90e−01	1.32e−10	4.00e−14	4.62e−10
	e−06	2.89e−06	9.90e−01	1.04e−06	1.85e−10	3.31e−06
	e−02	4.96e−02	3.26e+00	3.11e−02	1.17e−05	4.11e−02

TLS algorithm. The results are tabulated in table 1. The examples considered are approximately the same as those given in [3, section 7]. All matrices were generated by replacing the singular values σ_i in the SVD of randomly generated matrices. Columns 3 and 4 compare the distance between the approximate null space $R(V_2)$, as computed by the RR URV algorithm, and the exact one $R(V_0)$ (as computed by the SVD) with the bound given in theorem 2. Notice that the quality of the approximation V_2 only depends on the quality of the estimates $\text{est}(v_{\min}(R_i))$ and not on the gap σ_k/σ_{k+1}. This suggests that the bounds given in theorem 2 are too pessimistic and that tighter bounds can be derived that do not depend on the gap. This is in contrast to the RR QR factorization based results shown in [3] that do depend on the gap showing that the RR URV based results are clearly more accurate that the RR QR based ones. The next two columns yield the norm of F and $|G| - \Sigma_0$ and show how closely the RR URV decomposition approaches the exact RR URV decomposition as defined in theorem 1. As expected, the closeness depends directly on the quality of the estimates $\text{est}(v_{\min}(R_i))$. Observe that $\||G| - \Sigma_0\|_2$ is much smaller that $\|F\|_2$ and both are even much smaller than the errors on the estimates of $v_{\min}(R_i)$ even for very small gaps σ_k/σ_{k+1}. The last column compares the RR URV based TLS solution $\check{x}$ with the true TLS solution $\hat{x}$ computed by means of the SVD (see e.g. [7, algor. 12.3.1]). Again, note that the relative accuracy of $\check{x}$ only depends on the quality of the estimates of $v_{\min}(R_i)$ (same order of magnitude) independently of the gap.

From these numerical examples it can be concluded that our algorithm performs very well and provides a TLS solution of $Ax \approx b$ having the same relative accuracy as the accuracy of the given estimates of the null space vectors $v_{\min}(R_i)$ independently of the gap between the signal singular values $\sigma_1, \ldots, \sigma_k$ and the noise singular values $\sigma_{k+1}, \ldots, \sigma_n$ of $[A; b]$.

3.7.2. Comparison with other TLS algorithms

In this section, we compare the flop counts of the TLS algorithm 3 with the currently used TLS algorithms in function of the dimensions m and n of the data matrix $C_{m\times n} = [A; B_{m\times d}]$, the number of right-hand sides d and the computed rank k of C. In particular, the classical TLS algorithm [7, section 12.3], [19], the partial TLS algorithm [18,21] and the block iterative TLS algorithm, based on inverse iteration applied implicitly to $R_n^H R_n$ (where $C = Q_C \begin{bmatrix} R_n \\ 0 \end{bmatrix}$ is the QR factorization of the data matrix C) [20,21], are considered. Since these algorithms operate on real matrices we restrict ourselves to this class in this section. The results are summarized in table 2 for the TLS algorithm 3 based on the RR URV algorithm 1. Whenever the number of right-hand side vectors d is small the last term $\mathcal{F}(d)$ can be neglected. Similar results hold for the RR ULV based algorithm provided $3(n^2 - k^2)$ is subtracted from each expression. Only the highest order terms are considered so that our results are valid for large m, n. The flop counts for the classical TLS, partial TLS and iterative TLS algorithm respectively are obtained from the flop counts of the classi-

Table 2
Comparison in flop counts of TLS algorithm 3 (based on the RR URV algorithm 1 and using ordinary Givens rotations) with the classical TLS, the partial TLS and the iterative TLS algorithm based on inverse iteration with blocksize $n-k$. Here, diff represents the highest order difference in flop counts so that diff $\geqslant 0$ whenever algorithm 3 is more efficient than the considered TLS algorithm. An $m \times n$ real matrix $C = [A; B_{m\times d}]$ is considered where k denotes the computed rank of $C (k \leqslant n-d)$. s is the average number of required QR / QL iteration steps for convergence to one basis vector with the considered algorithm. K_i is the number of required inverse iteration steps for the block iterative TLS algorithm and K depends on the method used for estimating each $v_{\min}(R_i)$ in the RR URV algorithm 1.

TLS algorithm	diff
classical TLS ($m < \frac{5}{3}n$)	$2mn^2 + (6s - K - 2)\frac{2n^3}{3} + (K+3)\frac{2k^3}{3} + \mathcal{F}(d)$
classical TLS ($m \geqslant \frac{5}{3}n$)	$(6s - K + 3)\frac{2n^3}{3} + (K+3)\frac{2k^3}{3} + \mathcal{F}(d)$
partial TLS ($m < \frac{5}{3}n$)	$2mn^2 + (6s - K - 1)\frac{2n^3}{3} + (K+3)\frac{2k^3}{3} - 2kn^2 - 4sk^2n + \mathcal{F}(d)$
partial TLS ($m \geqslant \frac{5}{3}n$)	$(6s - K + 4)\frac{2n^3}{3} + (K+3)\frac{2k^3}{3} - 2kn^2 - 4sk^2n + \mathcal{F}(d)$
block iterative TLS	$(13K_i - K - 3)\frac{2n^3}{3} + (2K_i + K + 3)\frac{2k^3}{3} - 14K_ikn^2 + 4K_ik^2n + \mathcal{F}(d)$
	with $\mathcal{F}(d) = -2dn^2 + 6dk^2 + 2nd^2 - \frac{2}{3}d^3 + 2kd^2 - 4knd$

cal SVD, partial SVD [22,21] and the inverse iteration algorithm [20,21] respectively (these counts are doubled here since multiplications, as well as additions, are considered here) by adding

$$4dn^2 + (2k - 3n)d^2 + \frac{4}{3}d^3 - 4knd \text{ flops},$$

i.e., the amount of flops needed for computing the TLS solution from the computed basis $V_2 = [v_{k+1}, \ldots, v_n]$ of the approximate null space of the matrix C as described by (25)–(26).

For reasonable values of the rank k of C and the number K of required inverse iteration steps performed for estimating the $v_{\min}(R_i)$, the expressions "diff" in table 2 are positive for any m, n indicating that TLS algorithm 3 is most efficient. This is certainly the case for the partial and the classical TLS algorithms but also for the block iterative TLS algorithm provided k is sufficiently small compared to n. This happens for instance in signal processing applications [23] and in solving TLS problems with multiple right-hand sides (d large).

Filling in typical values of m, n, k, d, as done in table 3, shows that the newly proposed TLS algorithm 3 is up to 7 times more efficient than the other TLS algorithms depending on the algorithm and the parameters used. Especially, the ratio m/n and the rank k influence the relative efficiency. The smaller the ratio m/n, the more efficient TLS algorithm 3 is compared to the other ones under discussion. Decreasing the number of right-hand sides d or decreasing the number K of inverse iterations performed for estimating $v_{\min}(R_i)$ slightly improves the relative efficiency of TLS algorithm 3. Observe that the influence of the rank k differs from one algorithm to the other. In particular, the lower rank k is, the better is the efficiency of TLS algorithm 3 compared to the partial TLS and block iterative TLS algorithms in contrast to the classical TLS algorithm (of which the computational efficiency does not depend on k).

Table 3
Number of times that the RR URV based TLS algorithm 3 is more efficient than the classical TLS, the partial TLS and the block iterative TLS algorithm for different values of m, n, k, d, s, K and K_i as defined in table 2. This quantity is given by $(T + \text{diff})/T$, where diff is defined in table 2 and T by expression (23), and indicates that algorithm 3 is more efficient whenever the value is > 1.

m	n	k	d	s	K	K_i	class. TLS	part. TLS	it. TLS
30	28	17	1	2	2	2	2.94	2.11	1.29
30	28	17	1	2	3	4	2.62	1.88	1.97
30	28	17	1	2	10	10	1.48	1.06	2.51
50	30	15	1	2	2	2	2.57	2.16	1.47
50	30	15	4	2	2	2	2.44	2.06	1.40
50	30	5	1	2	2	2	2.39	2.41	2.54
60	50	48	1	2	2	2	6.02	1.97	0.90
60	50	30	1	2	2	2	2.90	2.13	1.30
60	50	5	1	2	2	2	2.48	2.56	3.17
100	50	48	1	2	2	2	4.12	1.80	0.95
100	50	30	1	2	2	2	2.55	1.98	1.22
100	50	5	1	2	2	2	2.27	2.33	2.66
110	100	98	1	2	2	3	7.16	1.98	0.96
500	100	98	1	2	2	3	2.24	1.30	0.99
900	100	98	1	2	2	3	1.67	1.16	1.00
110	100	5	2	2	2	3	2.50	2.61	5.12
500	100	5	2	2	2	3	1.67	1.72	2.61
900	100	5	2	2	2	3	1.41	1.44	1.99

Moreover, the iterative TLS algorithm requires a priori knowledge of the rank k of the data matrix $C = [A; B]$ since the dimension $n - k$ of the computed approximate null space is fixed during the execution of the algorithm. In contrast, TLS algorithm 3 computes the rank of C during its execution and can easily lower the rank, if necessary, which is of course one of its main advantages.

Finally, let us compare our TLS algorithm 3 based on the RR URV or ULV decomposition with the TLS algorithm outlined in [3] and based on the RR QR factorization. As shown in section 2.2, the RR QR factorization actually requires slightly less computations than the RR URV or ULV decomposition. But on the other hand, the RR QR based TLS algorithm of [3] needs to compute explicitly a unitary basis of the approximate null space of $[A; b]$ in order to deduce the minimum norm TLS solution. Moreover, some more inverse iteration steps are to be performed in order to improve this basis especially in the presence of a small gap σ_k/σ_{k+1}. A more detailed comparison is needed to exhibit clearly the (dis)advantages of one method to the other.

4. Conclusions

This paper shows how the use of a Rank-Revealing (RR) two-sided orthogonal decomposition (instead of the SVD) leads to an efficient algorithm for solving TLS

problems $A_{m\times(n-d)}X \approx B_{m\times d}$. Two different decompositions, called URV and ULV respectively, are presented decomposing the matrix into a product of a unitary matrix, an upper and lower triangular matrix respectively, and another unitary matrix in such a way that the effective rank k of the matrix is obvious and at the same time the noise subspace is exhibited explicitly. Two algorithms are outlined. If the singular vectors corresponding to the minimal singular value of the triangular factors of decreasing dimension, generated during execution of the algorithm, can be estimated exactly, then the RR two-sided orthogonal decomposition generates exactly the smallest singular values and corresponding singular vectors of the noise subspace. If not, appropriate bounds can be derived describing the closeness between the computed noise subspaces and the exact ones.

Based on this decomposition, a new TLS algorithm is presented and implemented on a linear array of n processors in such a way that it runs in $3(n-k-1)+k$ time units. Its high efficiency is mainly due to the fact that this algorithm need not compute an explicit basis of the approximate null space of the data matrix $[A;B]$ in order to compute the minimum norm TLS solution. Additionally, the algorithm is extended to multidimensional TLS problems and nongeneric TLS problems and two parallel implementations are discussed. If implemented on an array of n processors, $2d(n-k-d)+n$ time units are needed but better execution times, namely $3(n-k-1)+k$ time units, can be achieved provided more processors are used, at least $n^2-(k+1)^2)/3$. This algorithm is also recommended for updating slowly varying TLS problems since good estimates of the approximate null space, computed in the previous problem, can be used to compute the solution of the present problem. Only in problems where the changes in the data matrix $[A;B]$ are of rank one, more efficient rank one updating RR URV and ULV algorithms can be used. Since these algorithms require the explicit computation of the approximate null space basis, they can not be used in combination with the newly presented TLS algorithm but the TLS solution should be updated or downdated in the classical way. Finally, the newly presented TLS algorithm is shown to be more efficient than the classical and partial TLS algorithm and also more efficient than the iterative TLS algorithm, based on inverse iteration, provided the rank k of $[A;B]$ is sufficiently small compared to n.

References

[1] C.H. Bischof and P.C. Hansen, Structure- preserving and rank-revealing QR factorizations, SIAM J. Sci. Stat. Comput. 12 (1991) 1332-1350.

[2] T.F. Chan, Rank revealing QR factorizations, Lin. Alg. Appl. 88/89 (1987) 67-82.

[3] T.F. Chan and P.C. Hansen, Computing truncated singular value decomposition least squares solutions by rank revealing QR-factorizations, SIAM J. Sci. Stat. Comput. 11 (1990) 519-530.

[4] T.F. Chan and P.C. Hansen, Some applications of the rank revealing QR factorization, SIAM J. Sci. Stat. Comput. 13 (1992) 727–741.

[5] W.R. Ferng, G.H. Golub and R.J. Plemmons, Adaptive Lanczos methods for recursive condition estimation, Numer. Alg. 1 (1991) 1-19.
[6] L.V. Foster, Rank and null space calculations using matrix decomposition without column interchanges, Lin. Alg. Appl. 74 (1986) 47-71.
[7] G.H. Golub and C.F. van Loan, *Matrix Computations*, 2nd ed. (The Johns Hopkins University Press, Baltimore, MD, 1989).
[8] N.J. Higham, A survey of condition number estimation for triangular matrices, SIAM Rev. 29 (1987) 575-596.
[9] D.J. Pierce and R.J. Plemmons, Fast adaptive condition estimation, SIAM J. Matrix Anal. Appl. 13 (1992) 274-291.
[10] G.T. Shroff and C.H. Bischof, Adaptive condition estimation for rank-one updates of QR factorizations, SIAM J. Matrix Anal. Appl. 13, no. 4 (1992) 1264–1278.
[11] G.W. Stewart, The economical storage of plane rotations, Numer. Math. 25 (1976) 137-138.
[12] G.W. Stewart, On the implicit deflation of nearly singular systems of linear equations, SIAM J. Sci. Stat. Comput. 2 (1981) 136-140.
[13] G.W. Stewart, An updating algorithm for subspace tracking, IEEE Trans. Signal Proc. 40 (1992) 1535–1541.
[14] R. Mathias and G.W. Stewart, A block QR algorithm and the singular value decomposition, Lin. Alg. Appl. (1992), to appear.
[15] G.W. Stewart, Updating a rank-revealing ULV decomposition, Techn. Report UMIACS-TR 91-46, CS-TR 2627, Dept. of Computer Sciences, Univ. of Maryland, MD (1991), to appear in SIAM J. Matrix Anal. Appl.
[16] G.W. Stewart, Updating URV decompositions in parallel, Techn. Report UMIACS-TR 92-44, CS-TR 2880, Department of Computer Sciences, Univ. of Maryland, MD (April 1992).
[17] S. van Huffel, On the significance of nongeneric total least squares problems, SIAM J. Matrix Anal. Appl. 13 (1992) 20-35.
[18] S. van Huffel and J. Vandewalle, The partial total least squares algorithm, J. Comput. Appl. Math. 21 (1988) 333-341.
[19] S. van Huffel and J. Vandewalle, Analysis and solution of the nongeneric total least squares problem, SIAM J. Matrix Anal. Appl. 9 (1988) 360-372.
[20] S. van Huffel and J. Vandewalle, Iterative speed improvement for solving slowly varying total least squares problems, Mech. Syst. Sign. Proc. 2 (1988) 327-348.
[21] S. van Huffel and J. Vandewalle, *The Total Least Squares Problem: Computational Aspects and Analysis*, Frontiers in Applied Mathematics series, Vol. 9 (SIAM, Philadelphia, 1991).
[22] S. van Huffel, J. Vandewalle and A. Haegemans, An efficient and reliable algorithm for computing the singular subspace of a matrix, associated with its smallest singular values, J. Comput. Appl. Math. 19 (1987) 313-330.
[23] M.D. Zoltowski, Signal processing applications of the method of total least squares, *Proc. 21st Asilomar Conf. on Signals, Systems and Computers*, Pacific Grove, CA (November 1987) pp. 290-296.