Signal Processing 25 (1991) 203–213
Elsevier

A systolic algorithm for QSVD updating

Marc Moonen
ESAT Katholieke Universiteit Leuven, K. Mercierlaan 94, 3001 Heverlee, Belgium

Paul Van Dooren
University of Illinois at Urbana-Champain, Department of Electrical and Computer Engineering, 1101 West Springfield Avenue, Urbana, IL 61801, USA

Joos Vandewalle
ESAT Katholieke Universiteit Leuven, K. Mercierlaan 94, 3001 Heverlee, Belgium

Received 13 June 1991

Abstract. In earlier reports, a Jacobi-type algorithm for SVD updating has been developed and implemented on a systolic array. Here, this is extended to a generalized decomposition for a matrix pair, viz. the quotient singular value decomposition (QSVD). Updating problems are considered where new rows are appended to either one or both of the matrices involved. Systolic arrays as well as square root free implementations are described.

Zusammenfassung. In vorausgehenden Berichten wurde ein Jacobi-ähnlicher Algorithmus zur fortlaufenden Aktualisierung der SVD einer Matrix und die zugehörige Realisierung auf einem systolischen Array vorgestellt. In diesem Beitrag wird eine Erweiterung auf die verallgemeinerte Zerlegung eines Matrizenpaares, die sogenannten Quotient SVD (QSVD), behandelt. Es werden Aufgabenstellungen betrachtet, bei denen entweder nur eine oder beide beteiligten Matrizen durch Hinzunahme neuer Zeilen aktualisiert werden. Außerdem werden wurzelfreie Realisierungen sowie systolische Arrays beschrieben.

Résumé. Un algorithme de type Jacobi pour la mise à jour de la décomposition en valeurs singulières (SVD) a été développé et implanté sur un réseau systolique dans des articles antérieurs. Nous l'étendons ici à la décomposition généralisée pour une paire de matrices, c'est à dire pour la décomposition en valeurs singulières quotient (QSVD). Nous considérons les problèmes de mise à jour impliqués par l'ajout de nouvelles lignes à une ou aux deux matrices considerées. Nous décrivons des implantations sur réseau systolique et des implantations sans calcul de racine carrée.

Keywords. Singular value decomposition, parallel algorithms, recursive algorithms.

1. Introduction

The problem of continuously updating matrix decompositions as new rows are appended, frequently occurs in signal processing applications such as adaptive beamforming, direction finding and spectral analysis. An efficient technique for updating the singular value decomposition (SVD), amenable to systolic implementation, has been developed in [10, 11]. In the present paper, this approach is extended to a generalized decomposition for a matrix pair, viz. the quotient singular value decomposition (QSVD).[1] We consider updating problems where new rows are appended to either one or both of the matrices involved. As far as we know, these problems had not been looked at so far.

The approach to SVD updating consists in combining QR-updating with a Jacobi-type algorithm for the SVD, applied to the triangular factor. In each step, only $O(n^2)$ operations are performed,

[1] Formerly known as generalized singular value decomposition. See [1] for a standardized nomenclature.

0165-1684/91/$03.50

where n is the problem size. The outcome is a seemingly non-iterative algorithm that at any time step produces an approximate decomposition computed from a previous approximation, with a tracking error of the order of magnitude of the time variation in $O\{n\}$ time steps [10]. In [11], it is shown how to combine the different operations on one and the same systolic array with $O\{n^2\}$ processors. The obtained throughput is $O\{n^0\}$, i.e. independent of the problem size. The idea of combining different computational steps on the same array – with intermingled computational fronts and a small computational overhead for an increased throughput – is apparently new here.

For the QSVD updating problem one can similarly combine QR-updating with a Jacobi-type QSVD algorithm. Here we focus on such an updating algorithm and its parallel implementation. The performance and error analysis results for SVD updating [10] thereby largely carry over to the QSVD case, and are left out for the sake of brevity. In Section 2 the QSVD updating algorithm is derived. Its systolic implementation is developed in Section 3, and in Section 4 it is shown how to obtain a square root free updating algorithm.

Finally, let us mention that a similar updating algorithm is readily developed for the product SVD (PSVD) of a matrix pair [1, 4]. However, while QSVD updating definitely has practical relevance, this cannot be said yet about the PSVD updating problem. Therefore these results were left out here, and we refer to [8] for the details.

2. QSVD updating

The QSVD is a simultaneous orthogonal reduction of two matrices A and B to triangular matrices, the corresponding rows of which are parallel. As further on we consider the updating problem when new rows are appended sequentially, we can assume that the number of rows in A and B is larger than the number of columns. The QSVD then reads as follows [14]:

$$\underbrace{A}_{m\times n} = \underbrace{U_A}_{m\times n}\underbrace{\Sigma_A R}_{n\times n}\underbrace{Q^{\mathrm{T}}}_{n\times n},$$

$$\underbrace{B}_{p\times n} = \underbrace{U_B}_{p\times n}\underbrace{\Sigma_B R}_{n\times n}\underbrace{Q^{\mathrm{T}}}_{n\times n},$$

$$\Sigma_A = \mathrm{diag}\{\alpha_1, \ldots, \alpha_n\},$$

$$\Sigma_B = \mathrm{diag}\{\beta_1, \ldots, \beta_n\}.$$

Here $R_{n\times n}$ is an upper triangular matrix, the matrices U_A and U_B have orthonormal columns and Q is an orthogonal matrix.[2] The pairs $\{\alpha_i, \beta_i\}$ are called the quotient singular value pairs of $\{A, B\}$, while α_i/β_i are the quotient singular values. QSVD applications include generalized least squares problems associated with the general Gauss–Markov linear model [12], generalized total least squares problems [15], generalized state space identification [9], etc. Most of the time A is a data matrix while B corresponds to a known error covariance matrix. In the white noise case, where $B = I_{n\times n}$, such QSVD methods reduce to corresponding ordinary SVD methods.

Briefly, the QSVD is computed as follows [5, 13]. First, the matrices A and B are reduced to triangular form by preliminary QRDs:

$$A = \underbrace{Q_A}_{m\times n} R_A, \qquad B = \underbrace{Q_B}_{p\times n} R_B,$$

where R_A and R_B are square upper triangular, and Q_A and Q_B have orthonormal columns. The QSVD of $\{A, B\}$ readily follows from the QSVD of $\{R_A, R_B\}$. The QSVD of $\{R_A, R_B\}$ is then computed by carrying out an iterative procedure, where a series of Givens transformations is applied to R_A and R_B in order to yield triangular factors with parallel rows $\Sigma_A R$ and $\Sigma_B R$. Each iteration essentially reduces to a QSVD of a 2×2 block on the main diagonal, parallelizing the rows of $\{R_A\}_{i,i+1}$ and $\{R_B\}_{i,i+1}$, where $\{M\}_{i,i+1}$ denotes the 2×2 matrix on the intersection of rows i, $i+1$ and columns i, $i+1$ of M. We refer to [5, 13] for the

[2] $U_A^{\mathrm{T}}U_A = U_B^{\mathrm{T}}U_B = Q^{\mathrm{T}}Q = I_{n\times n}$.

details. By making use of either inner rotations + permutations or outer rotations, an elegant pipelined implementation is obtained, wherewith the pivot index repeatedly takes up all possible values

$$i = 1, 2, \ldots, n-1.$$

Again we refer to [5, 6] for the details.

The updating problem considered here consists in computing the QSVD of the modified matrices, after appending new rows to A and/or B:

$$\underbrace{\underline{A}}_{(m+1)\times n} = \begin{bmatrix} \lambda A \\ a^{\mathrm{T}} \end{bmatrix} = \underbrace{\underline{U}_A}_{(m+1)\times n} \underline{\Sigma}_A \underline{R} \underline{Q}^{\mathrm{T}},$$

$$\underbrace{\underline{B}}_{(p+1)\times n} = \begin{bmatrix} \lambda B \\ b^{\mathrm{T}} \end{bmatrix} = \underbrace{\underline{U}_B}_{(p+1)\times n} \underline{\Sigma}_B \underline{R} \underline{Q}^{\mathrm{T}}.$$

We consider constant weighting factors λ, for the sake of brevity. In on-line applications a new updating needs to be performed after each sampling, such that $A_{[k]}$ and $B_{[k]}$ are defined in a recursive manner:

$$A_{[k]} = \begin{bmatrix} \lambda A_{[k-1]} \\ a_{[k]}^{\mathrm{T}} \end{bmatrix}, \qquad B_{[k]} = \begin{bmatrix} \lambda B_{[k-1]} \\ b_{[k]}^{\mathrm{T}} \end{bmatrix}.$$

In most cases, the U-matrices – with growing matrix dimensions! – need not be computed explicitly, only R, Q and Σ_A, Σ_B are of interest.

New data vectors are worked in as follows. Suppose that at time $k-1$, the triangular factors are reduced to $R_{A[k-1]}$ and $R_{B[k-1]}$ (with approximately parallel rows, say) and the corresponding Q-matrix is denoted as $Q_{[k-1]}$:

$$A_{[k-1]} = U_{A[k-1]} R_{A[k-1]} Q_{[k-1]}^{\mathrm{T}},$$

$$B_{[k-1]} = U_{B[k-1]} R_{B[k-1]} Q_{[k-1]}^{\mathrm{T}}$$

(only $R_{A[k-1]}$, $R_{B[k-1]}$ and $Q_{[k-1]}$ are stored). After appending new rows $a_{[k]}^{\mathrm{T}}$ and/or $b_{[k]}^{\mathrm{T}}$, one has a decomposition of the type

$$A_{[k]} = \begin{bmatrix} U_{A[k-1]} & \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} \\ \begin{matrix} 0 & \cdots & 0 \end{matrix} & 1 \end{bmatrix} \begin{bmatrix} \lambda R_{A[k-1]} \\ a_{[k]}^{\mathrm{T}} Q_{[k-1]} \end{bmatrix} Q_{[k-1]}^{\mathrm{T}},$$

$$B_{[k]} = \begin{bmatrix} U_{B[k-1]} & \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} \\ \begin{matrix} 0 & \cdots & 0 \end{matrix} & 1 \end{bmatrix} \begin{bmatrix} \lambda R_{B[k-1]} \\ b_{[k]}^{\mathrm{T}} Q_{[k-1]} \end{bmatrix} Q_{[k-1]}^{\mathrm{T}}.$$

First, the triangular factors are restored by performing QR factorizations, or more specifically a QR updating [3] with the transformed input vectors $\tilde{a}_{[k]}^{\mathrm{T}} = a_{[k]}^{\mathrm{T}} Q_{[k-1]}$ and $\tilde{b}_{[k]}^{\mathrm{T}} = b_{[k]}^{\mathrm{T}} Q_{[k-1]}$:

$$A_{[k]} = \underbrace{\begin{bmatrix} U_{A[k-1]} & \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} \\ \begin{matrix} 0 & \cdots & 0 \end{matrix} & 1 \end{bmatrix} Q_{A[k]}}_{U_{A[k]}} \tilde{R}_{A[k]} Q_{[k-1]}^{\mathrm{T}},$$

$$B_{[k]} = \underbrace{\begin{bmatrix} U_{B[k-1]} & \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} \\ \begin{matrix} 0 & \cdots & 0 \end{matrix} & 1 \end{bmatrix} Q_{B[k]}}_{U_{B[k]}} \tilde{R}_{B[k]} Q_{[k-1]}^{\mathrm{T}}.$$

Here $Q_{A[k]}$ and $Q_{B[k]}$ are $(n+1)\times n$ matrices with othogonal columns, which need not be computed explicitly. The QR updatings do not alter the matrix $Q_{[k-1]}$. New rows are thus worked in by performing matrix–vector multiplications and QR updates. After each such update the QSVD process is resumed. Similar to the SVD updating procedure [10], a QSVD updating procedure can then be devised as follows, with after each QR update a fixed number of QSVD steps, say r. We make an arbitrary choice and set r equal to $n-1$, so that after each QR update the pivot index takes up all values $i = 1, \ldots, n-1$ only once. In this case, both the QR updating and the rotations following the update take $O(n^2)$ operations, which in particular results in an elegant parallel implementation, see

Section 3. All of our further results are straightforwardly recast for other choices for r. The tracking error of course largely depends on the choice for r, see [10].

Initialization

$$Q \Leftarrow I_{n\times n}$$

$$R_A \Leftarrow O_{n\times n}$$

$$R_B \Leftarrow O_{n\times n}$$

Loop

for $k = 1, \ldots, \infty$

input new measurement vectors $a_{[k]}$, $b_{[k]}$

1. matrix–vector multiplications

$$\tilde{a}_{[k]} = a_{[k]}^{\mathrm{T}} Q$$

$$\tilde{b}_{[k]} = b_{[k]}^{\mathrm{T}} Q$$

2. QR updates

$$R_A \Leftarrow Q_{A[k]}^{\mathrm{T}} \begin{bmatrix} \lambda R_A \\ \tilde{a}_{[k]}^{\mathrm{T}} \end{bmatrix}$$

$$R_B \Leftarrow Q_{B[k]}^{\mathrm{T}} \begin{bmatrix} \lambda R_B \\ \tilde{b}_{[k]}^{\mathrm{T}} \end{bmatrix}$$

3. QSVD steps

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

$$R_A \Leftarrow \Theta_{A[k,i]}^{\mathrm{T}} R_A Q_{[k,i]}$$

$$R_B \Leftarrow \Theta_{B[k,i]}^{\mathrm{T}} R_B Q_{[k,i]}$$

$$Q \Leftarrow Q Q_{[k,i]}$$

end

end

Here time indices are sometimes omitted for brevity. Matrices $\Theta_{A[k,i]}$, $\Theta_{B[k,i]}$ and $Q_{[k,i]}$ are embeddings of plane rotations, in the $\{i, i+1\}$ plane, corresponding to the ith iteration in time step k.

NOTE. a dual QSVD updating problem. So far, we considered the updating problem where new *rows* are appended to the matrices A and/or B. Alternatively, one could also append *columns* to A and B, and try to devise a similar updating algorithm. However, this dual QSVD updating problem seems to be much more difficult to solve. The updating problem now consists in computing the modified QSVD

$$\underline{A} = [\lambda A \quad a] = \underline{U}_A \underline{\Sigma}_A R Q^{\mathrm{T}},$$

$$\underline{B} = [\lambda B \quad b] = \underline{U}_B \underline{\Sigma}_B R Q^{\mathrm{T}}$$

by making use of the original QSVD of A and B. This time U_A, U_B, Σ_A, Σ_B and R are important. The matrix Q has growing dimensions and is not computed explicitly. Suppose that A and B are stored in triangular form. The main difficulty appears to be the QR updating of the triangular factors R_A and R_B after appending new columns. As the orthogonal updating transformation to the right necessarily has to be the same for both R_A and R_B (Q-matrix!), this updating actually requires both left and right transformations. The available approximations for U_A and U_B can then change significantly, or in other words the approximate collinearity of R_A and R_B can get lost, so that many more QSVD steps are needed afterwards in order to restore the approximation.

Fortunately, any updating algorithm of this kind would apparently be of little use in practice. In general, the QSVD for two matrices $A_{m\times n}$ and $B_{p\times n}$, with more columns than rows ($n > p, m$) looks as follows:

$$\underbrace{A}_{m\times n} = \underbrace{U_A}_{m\times m} \underbrace{\Sigma_A}_{m\times k} \left[\underbrace{R}_{k\times k} \quad \underbrace{0}_{k\times(n-k)} \right] Q^{\mathrm{T}},$$

$$\underbrace{B}_{p\times n} = \underbrace{U_B}_{p\times p} \underbrace{\Sigma_B}_{p\times k} \left[\underbrace{R}_{k\times k} \quad \underbrace{0}_{k\times(n-k)} \right] Q^{\mathrm{T}},$$

where $k = \mathrm{rank}\{A^{\mathrm{T}}, B^{\mathrm{T}}\}$. If A and B contain measured data, with a signal-to-noise ratio of 100 say, then – due to the noise – A and B as well as $[A^{\mathrm{T}} \quad B^{\mathrm{T}}]$ generically have full row rank, $k = m + p$,

with a fairly small condition number of approximately 100. The QSVD then follows somewhat trivially from an RQ factorization:

$$\begin{bmatrix} A \\ B \end{bmatrix} = \underbrace{R}_{m+p \times m+p} \; Q^{\mathrm{T}},$$

which can be written in a QSVD form

$$A = \underbrace{I}_{m \times m} [I \quad 0] R Q^{\mathrm{T}},$$

$$B = \underbrace{I}_{p \times p} [0 \quad I] R Q^{\mathrm{T}}.$$

For practical applications – e.g. systems identification from measured I/O-data – the most valuable information that can be drawn from such a QSVD would be the intersection of the row spaces of A and B [9]. Due to the noise, this intersection is immediately lost. The QSVD is clearly not able to point out an 'approximate intersection', but becomes trivial instead. A one stage QSVD procedure thus turns out to be useless for these types of problems. Alternatively, one can make use of a two stage QSVD algorithm, like in [14]. In a first step, the SVD of $[A^{\mathrm{T}} \; B^{\mathrm{T}}]^{\mathrm{T}}$ is computed, and in a second step, the QSVD follows from the CS-decomposition of certain submatrices computed in the first step. The difference between these methods turns out to be an intermediate rank decision after the first SVD in the latter approach, that e.g. fixes the dimension of the approximate intersection, by implicitly setting the smaller singular values of $[A^{\mathrm{T}} \; B^{\mathrm{T}}]^{\mathrm{T}}$ equal to zero. Although this – possibly difficult – intermediate rank decision has been a main motive for developing a one stage QSVD algorithm [13], in practice it is apparently inevitable.

3. Systolic implementation

The QSVD array in [5] is quite similar to the SVD array of [6]. The triangular part contains two triangular factors instead of one. The processors on the main diagonal perform 2×2 QSVDs instead of SVDs, and the off-diagonal processors apply transformations to both R_A-elements and R_B-elements.[3] Similar to the SVD updating array in [11], one can therefore straightforwardly design a QSVD updating array, as displayed in Fig. 1. It consists of a square part – containing the Q matrix – which also computes the matrix–vector products, and a triangular part – containing the triangular factors – that performs the QR updating and the QSVD reduction. All these operations are carried out simultaneously as is detailed next. The correctness of the array has been verified by software simulation.

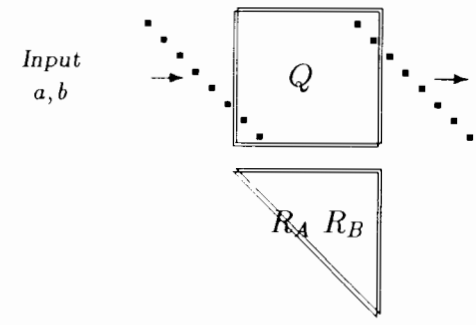

Fig. 1. Outline QSVD updating array.

The array operations are as follows, Fig. 2. Processors on the main diagonal perform 2×2 QSVDs [5]. Row transformation parameters are passed on to the right, while column transformation parameters are passed on upwards. Off-diagonal processors only apply and propagate these transformations to the blocks next outward. Column transformations are also propagated through the upper square part, containing the Q matrix (Q's first row in the top row, etc.). The 2×2 QSVDs that are performed in parallel on the main diagonal correspond to different pipelined sequences of $n-1$ rotations, where in each sequence the pivot index successively takes up the values $i = 1, \ldots, n-1$. Each such sequence corresponds to a time update. As pointed out in the algorithmic description, the QR updatings should

[3] Note that the row transformations applied to R_A and R_B are different.

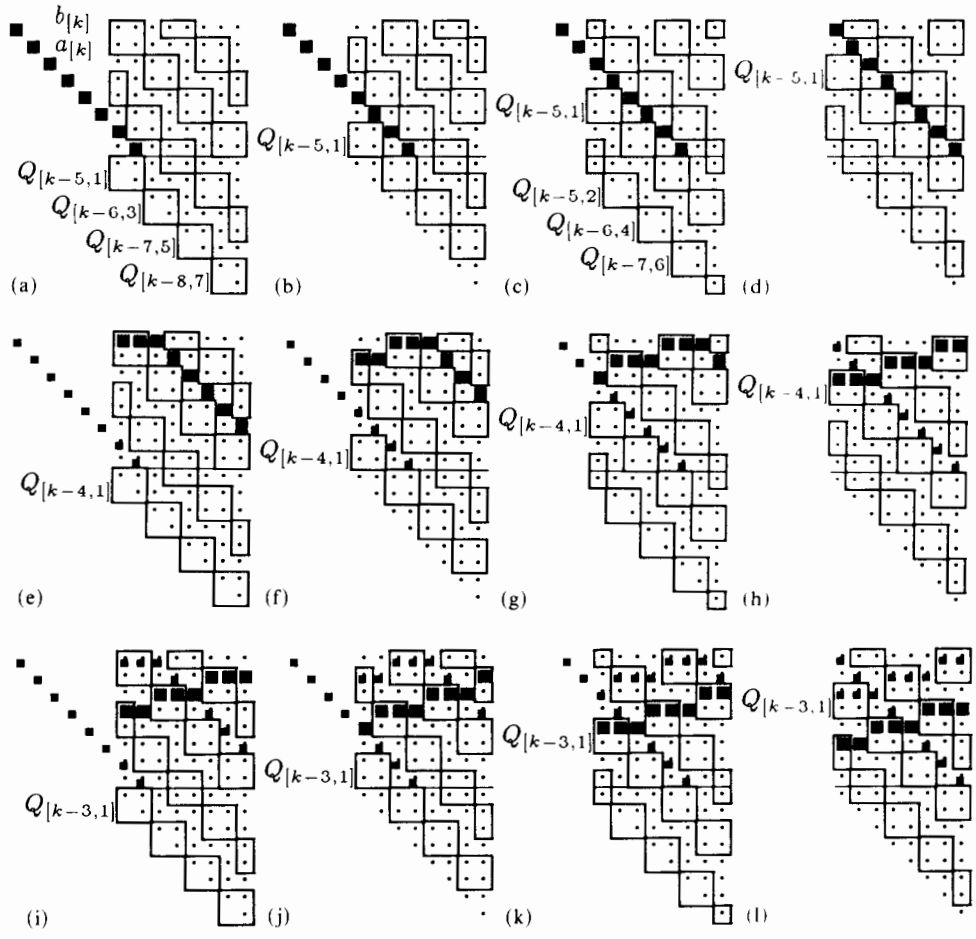

Fig. 2. Array operations (24 snapshots).

be inserted in between two such sequences. For clarity, the location of some of the column transformations is indicated explicitly in Fig. 2.

If both R_A and R_B have to be updated with new rows, the filled boxes in Fig. 2 actually represent two data vectors, viz. a and b. The $a_{[k]}$ and $b_{[k]}$ are indicated with the ■'s, while subsequent vectors are indicated with the ■'s. The data vectors a and b are fed in a skewed fashion as indicated, and are propagated to the right, in between two rotation fronts corresponding to the QSVD reduction (frames). Meanwhile, the matrix–vector products are accumulated from bottom to top in the usual way. Each processor in the square array receives a, b-components from its left neighbour, and intermediate products from its lower neighbour. The intermediate products are then updated and passed on to the upper neighbour, while the a, b-components are passed on to the right. The resulting matrix–vector products become available at the top end of the square array, and are then reflected and propagated downwards, towards the triangular array. The obtained matrix–vector products equal $\tilde{a}_{[k]}$, $\tilde{b}_{[k]}$ up to a number of transformations $Q_{[\ldots]}$, as clearly some older version of Q has been used for computing the products. While going downwards, the ■'s cross these upgoing rotations $Q_{[\ldots]}$, which are then applied not only to the Q matrix, but also to the ■'s, in order to obtain consistent results, see [11]. As an example,

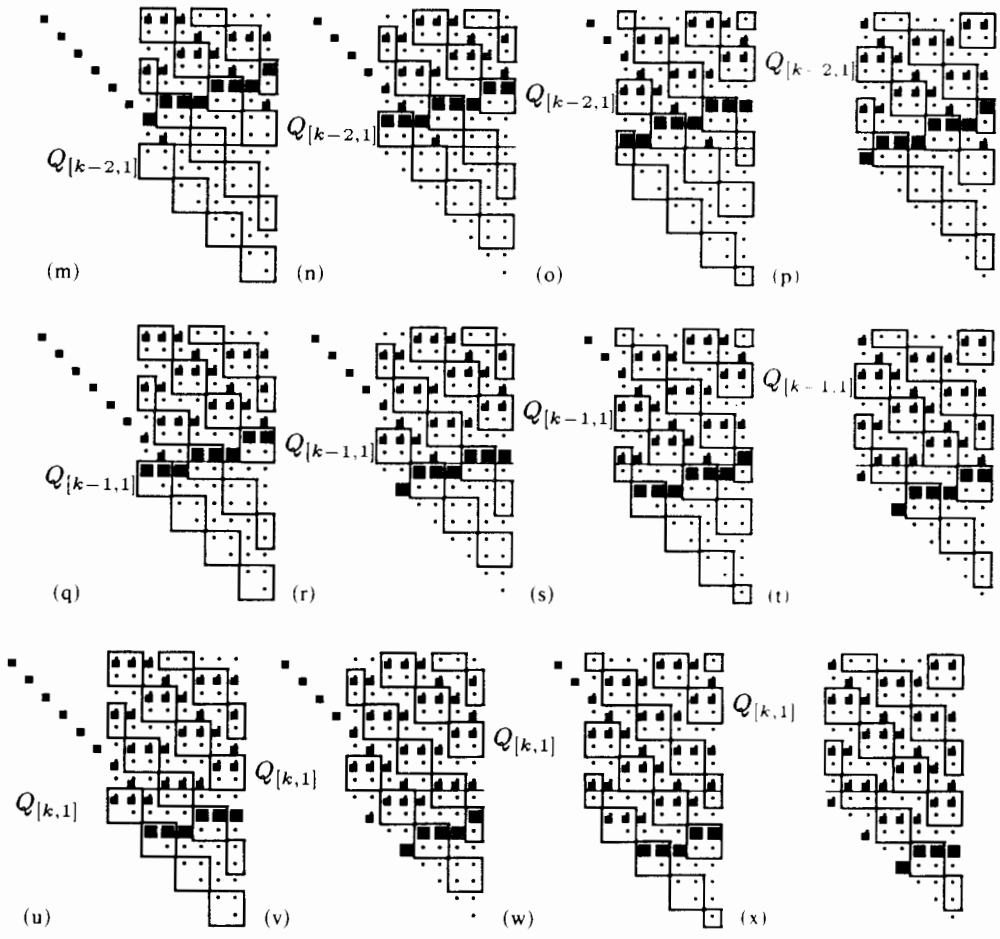

Fig. 2. Continued.

transformations $Q_{[k-5,1]}$, $Q_{[k-4,1]}$, $Q_{[k-3,1]}$, $Q_{[k-2,1]}$, $Q_{[k-1,1]}$ and $Q_{[k,1]}$ are seen to be applied to the ■'s in Fig. 2(e, h, k, n, q). The complete matrix–vector products are thus computed in several stages. This represents a computational overhead, but on the other hand it allows for a perfect pipelining of the different operations, such that the obtained throughput is $O\{n^0\}$ instead of $O\{n^{-1}\}$.

Finally, the QR updating is similarly interlaced with the QSVD steps in the triangular array (starting in Fig. 2(q)). In each frame, column and row transformations corresponding to the QSVD reduction are performed first, while in a second step, only row transformations are performed corresponding to the QR updating (affecting the ■-components and the upper part of the 2×2-blocks). Again, column transformations in the first step should be applied to the ■-components as well (overhead). For more details, we refer to [11].

The QSVD is an efficient tool for computing e.g. the generalized total least squares (GTLS) solution for systems of linear equations, when the covariance matrix of errors in the rows is known up to a scale factor [15]. Here only one triangular factor, R_A say, is updated with new data vectors. The second one corresponds to the Cholesky factor of the error covariance matrix $\Delta = R_\Delta^T R_\Delta$, which is assumed fixed. Although the QSVD updating itself is straightforward, generating e.g. GTLS solution vectors is somewhat more involved. We consider

only the simpler case where $R_B = R_A$ is non-singular. An approximate GTLS solution is then obtained as the column(s) of $X = QR_\Delta^{-1}$ corresponding to the smallest quotient singular value(s) α_i/β_i. It can be extracted from the array as follows. Suppose we could generate a vector t on the main diagonal with all its components equal to zero, except for one component equal to '1' at the position of the smallest quotient singular value. The ratios of diagonal elements in R_A and R_Δ are estimates for the quotient singular values, so that we can make use of a threshold function:

$$t_i = 1 \quad \text{iff} \quad r_{ii} < \gamma,$$

$$t_i = 0 \quad \text{iff} \quad r_{ii} > \gamma,$$

$$r_{ii} = r_{ii}^A / r_{ii}^\Delta,$$

where γ satisfies

$$\frac{\alpha_{n-1}}{\beta_{n-1}} > \gamma > \frac{\alpha_n}{\beta_n}$$

and is assumed to be known a priori.[4] The output vector then equals $x = QR_\Delta^{-1}t$, the computation of which thus requires a triangular backsolve, followed by a matrix–vector multiplication. The backsubstitution necessarily starts off at the bottom of the array, with intermediate results being propagated upwards, unlike the QR updates that run in precisely the opposite direciton. This particularly seems to rule out an elegant implementation.

The crucial observation here is that, as for the R_Δ matrix – which is only affected by the QSVD steps – , the QSVD transformations can be viewed as pipelined sequences of $n-1$ rotations, starting at the lower right corner and propagated upwards along the diagonal. In other words, the original sequence of transformations acting upon R_A, viz.

for $k = 1, 2, \ldots,$ ν
  for $i = 1, 2, \ldots, n-1$
    $R_A \Leftarrow \Theta_{A[k,i]} R_A Q_{[k-1}$
  end
end

can implicitly be rearranged into

for $k = 1, 2, \ldots,$ ν
  for $i = n-1, n-2, \ldots, 1$
  $R_A \Leftarrow \Theta_{A[k-i,i]} R_A Q_{[k-i,i]}$
  end
end

which is an equivalent ordering [7], except for a different start-up phase. Backsubstitutions are then readily inserted, as detailed next. Note that with a sequential algorithm, one would obtain a column of $Q_{[k]} R_{\Delta[k]}^{-1}$ as a solution at time k. For the parallel implementation the obtained solution corresponds to an intermediate version $Q_{[k]}^{\circ} R_{\Delta[k]}^{\circ -1}$. This latter equals $Q_{[k]} R_{\Delta[k]}^{-1}$ up to a limited number of (column) transformations. As we consider slowly time-varying systems, the error is assumed reasonably small. This has also been verified by software simulation.

Figure 3 shows how successive output vectors $x = QR_\Delta^{-1}t$ are generated. The backsubstitution is performed in the triangular part, the multiplication with Q is performed in the square part. On the main diagonal, components of $t^* = R_\Delta^{-1}t$ are generated – see below – which are propagated upwards. Off-diagonal processors in both the triangular part and the square part receive such components from their lower neighbours, and intermediate results from their right neighbours. The products $t_j^* q_{ij}$ (square part) or $t_j^* r_{ij}^\Delta$ (triangular part) are added to the intermediate results. The adjusted intermediate results are then passed on to the left neighbour, while the t^*-components are passed on upwards.

[4] Alternatively, one can at the same time output *all* columns of X that have a corresponding r_{ii} smaller (or greater) than a predefined threshold γ. A t-vector with more than one non-zero component is then looked upon as a concatenation of different vectors $t_1, t_2, \ldots$, and the matrix–vector products $Xt_1, Xt_2, \ldots$ are computed accordingly.

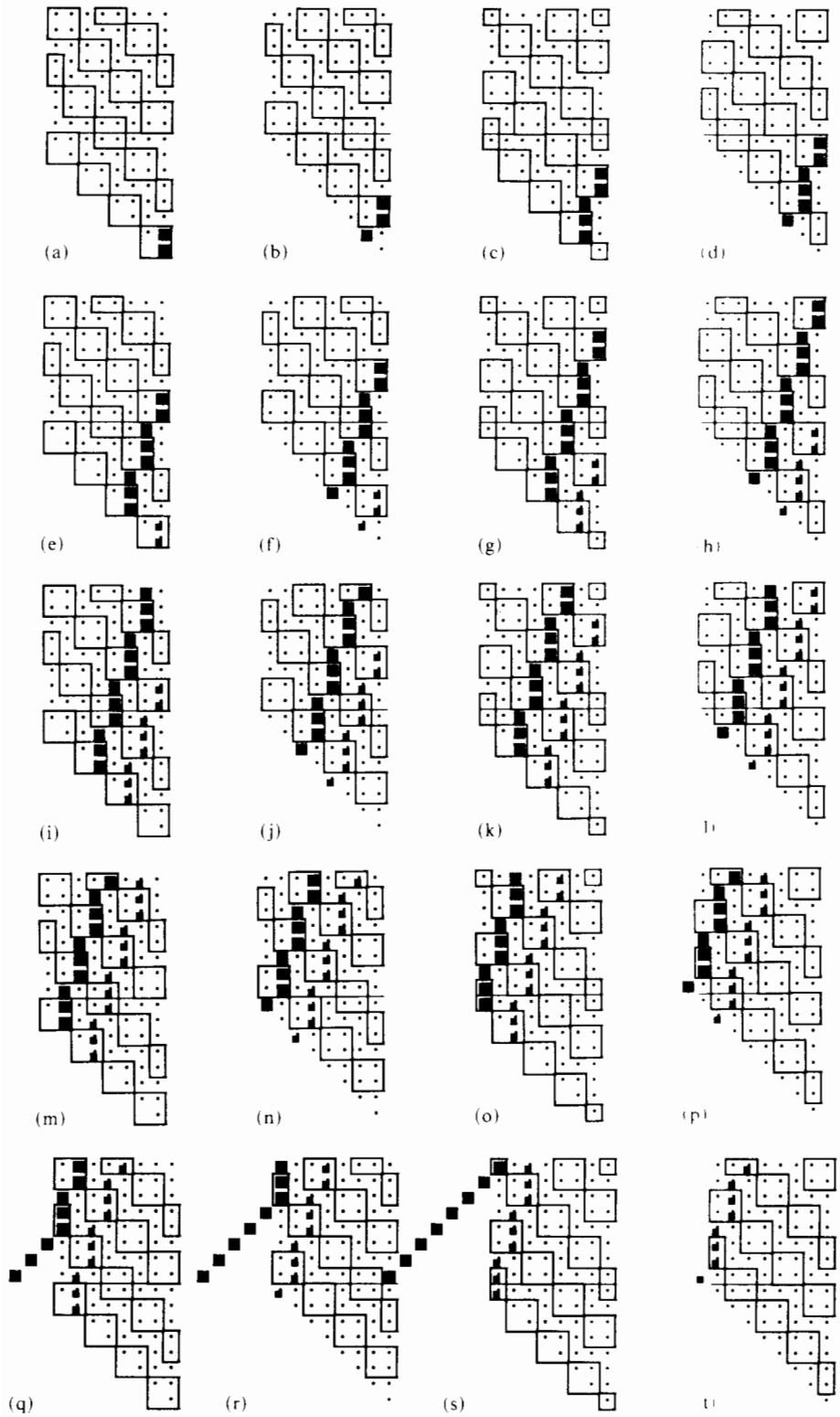

Fig. 3. Output vector generation (20 snapshots).

In this way, the ith diagonal processor receives the product

$$\sum_{j=i+1}^{n} t_j^* r_{ij}^{\Delta}$$

that is used to compute the next t^*-component

$$t_i^* = \left\{ t_i - \sum_{j=i+1}^{n} t_j^* r_{ij}^{\Delta} \right\} \{ r_{ii}^{\Delta} \}^{-1},$$

where t_i is generated by applying the aforementioned threshold function. Notice that row transformations applied to R_Δ in the QSVD steps (moving from the left to the right), should be applied to intermediate results as well (moving from the right to the left). Again this is a computational overhead, introduced whenever different computational fronts cross.

The output vectors run out at the left-hand side of the array in a skewed fashion as indicated in Fig. 3. If necessary, they can be bounced back into the array, and propagated to the right in order to make them available at the right-hand side – as was indicated in Fig. 1. Again, it should be stressed that all operations – viz. the QSVD updating and the output vector generation including the triangular backsolve – are carried out continuously and simultaneously.

4. Square root free implementation

A major computational bottle-neck for the above QSVD updating array appears to be the computation of the rotation parameters in the boundary processors, with explicit computation of a number of square roots. A square root free implementation can be obtained by combining modified Givens rotations with approximate SVD schemes. This is an extension of the well known square root free QR updating [2]. In [11], it has been shown how a generalized Gentleman procedure with a two-sided factorization of the triangular factor can be combined with approximate SVD schemes in order to obtain a square root free SVD updating algorithm, with an additional saving in the number of multiplications. The derivation of a square root free QSVD updating algorithm proceeds along the same lines.

At a certain time step, the triangular factors are reduced to R_A and R_B, with almost parallel rows, and both are stored in factorized form

$$D_A = D_{\text{row}_A}^{1/2} \bar{R}_A D_{\text{col}}^{1/2},$$

$$R_B = D_{\text{row}_B}^{1/2} \bar{R}_B D_{\text{col}}^{1/2}$$

(only $\bar{R}_A$, $\bar{R}_B$ and D_{row_A}, D_{row_B}, D_{col} are stored). Note that the column scaling is the same for R_A and R_B. In a systolic array, the diagonal matrices D_{row_A}, D_{row_B} and D_{col} are obviously stored in the processor elements on the main diagonal. The square part of the array contains a scaled version $\bar{Q}$ of the orthogonal matrix Q

$$Q = \bar{Q} D_{\text{col}}^{1/2}.$$

The only real difference compared to the SVD case of course concerns the 2×2 QSVDs on the main diagonal. In a first step an SVD of $\{R_A\}_{i,i+1} \text{adj}\{R_B\}_{i,i+1}$ is computed, while the second step consists in applying the same Givens rotation (column transformation) to both matrices [13]. As for the first SVD one immediately sees that

$$\begin{aligned}
&\{R_A\}_{i,i+1} \text{adj}\{R_B\}_{i,i+1} \\
&= \{D_{\text{row}_A}^{1/2}\}_{i,i+1} \{\bar{R}_A\}_{i,i+1} \overbrace{\{D_{\text{col}}^{1/2}\}_{i,i+1} \text{adj}\{D_{\text{col}}^{1/2}\}_{i,i+1}}^{\det\{D_{\text{col}}^{1/2}\}_{i,i+1} I_{2\times 2}} \\
&\quad \times \text{adj}\{\bar{R}_B\}_{i,i+1} \text{adj}\{D_{\text{row}_B}^{1/2}\}_{i,i+1} \\
&= \{D_{\text{row}_A}^{1/2}\}_{i,i+1} \{\bar{R}_A\}_{i,i+1} \text{adj}\{\bar{R}_B\}_{i,i+1} \\
&\quad \times \text{adj}\{D_{\text{row}_B}^{1/2}\}_{i,i+1} \det\{D_{\text{col}}^{1/2}\}_{i,i+1}
\end{aligned}$$

boils down to an SVD with a 2-sided factorization, which can be computed approximately without square roots, as was detailed in [11].

In an unfactored form, this approximate SVD reads as follows:

$$\begin{bmatrix} s_{i,i} & s_{i,i+1} \\ 0 & s_{i+1,i+1} \end{bmatrix}$$

$$\Leftarrow \underbrace{\begin{bmatrix} \cos\theta_A & -\sin\theta_A \\ \sin\theta_A & \cos\theta_A \end{bmatrix}}_{\{\Theta_{A[k]}\}_{i,i+1}} \underbrace{\begin{bmatrix} r^A_{i,i} & r^A_{i,i+1} \\ 0 & r^A_{i+1,i+1} \end{bmatrix}}_{\{R_{A[k-1]}\}_{i,i+1}}$$

$$\times \operatorname{adj}\Bigg\{ \underbrace{\begin{bmatrix} r^B_{i,i} & r^B_{i,i+1} \\ 0 & r^B_{i+1,i+1} \end{bmatrix}}_{\{R_{B[k-1]}\}_{i,i+1}} \Bigg\} \underbrace{\begin{bmatrix} \cos\theta_B & \sin\theta_B \\ -\sin\theta_B & \cos\theta_B \end{bmatrix}}_{\{\Theta^{\mathrm{T}}_{B[k]}\}_{i,i+1}},$$

from which it then follows that the second rows in $\{\Theta^{\mathrm{T}}_{A[k]} R_{A[k-1]}\}_{i,i+1}$ and $\{\Theta^{\mathrm{T}}_{B[k]} R_{B[k-1]}\}_{i,i+1}$ are parallel, so that again one column rotation $Q_{[k]}$ can upper-triangularize both matrices at the same time. For this Givens transformation, it suffices to simply apply Gentleman's square root free procedure. For details we refer to [11]. Again, it follows that on a systolic array, a square root free QSVD updating algorithm imposes hardly any changes compared to the original procedure.

5. Conclusions

It has been shown how a previously developed SVD updating technique can be extended to a generalized SVD updating problem, viz. QSVD updating where new rows are appended to either one or both of the matrices involved. A systolic implementation has been devised as well, with an additional possibility of continuously generating output vectors such as generalized total least squares solutions. Finally, it has briefly been shown how to obtain a square root free algorithm, by making use of approximate SVD schemes and a generalized Gentleman-procedure.

Acknowledgment

This work was sponsored in part by the BRA 3280 project of the European Commission. Marc Moonen is a senior research assistant with the N.F.W.O. (Belgian National Fund for Scientific Research).

References

[1] B. De Moor and G.H. Golub, Generalized singular value decompositions: A proposal for a standardized nomenclature, Internal Report, Dept. Comp. Sci., Stanford University, 1989.

[2] W.M. Gentleman, "Least squares computations by Givens transformations without square roots", *J. Inst. Math. Appl.*, Vol. 12, 1973, pp. 329–336.

[3] P.E. Gill, G.H. Golub, W. Murray and M.A. Saunders, "Methods for modifying matrix factorizations", *Math. Comp.*, Vol. 28, No. 126, 1974, pp. 505–535.

[4] M.T. Heath, A.J. Laub, C.C. Paige and R.C. Ward, "Computing the singular value decomposition of a product of two matrices", *SIAM J. Sci. Statist. Comput.*, Vol. 7, No. 4, 1986, pp. 1147–1159.

[5] F.T. Luk, "A parallel method for computing the GSVD", *Internat. J. Parallel Distr. Comp.*, Vol. 2, 1985, pp. 250–260.

[6] F.T. Luk, "A triangular processor array for computing singular values", *Linear Algebra Appl.*, Vol. 77, 1986, pp. 259–273.

[7] F.T. Luk and H. Park, "On parallel Jacobi orderings", *SIAM J. Sci. Statist. Comput.*, Vol. 10, No. 1, 1989, pp. 18–26.

[8] M. Moonen, Jacobi-type updating algorithms for signal processing, systems identification and control, *PhD Thesis*, Kath. Univ. Leuven, Dept. El. Eng., 1990.

[9] M. Moonen and J. Vandewalle, "A QSVD approach to on- and off-line state space identification", *Internat. J. control*, Vol. 51, No. 5, 1990, pp. 1133–1146.

[10] M. Moonen, P. Van Dooren and J. Vandewalle, "An SVD updating algorithm for subspace tracking", *SIAM J. Matrix Anal. Appl.*, to appear (also ESAT-SISTA report 89-13a).

[11] M. Moonen, P. Van Dooren and J. Vandewalle, A systolic array for SVD updating, ESAT-SISTA report 89-13b, K.U. Leuven, E.E. Dept. (submitted).

[12] C.C. Paige, "The general linear model and the generalized singular value decomposition", *Linear Algebra Appl.*, Vol. 70, 1985, pp. 269–284.

[13] C.C. Paige, "Computing the generalized singular value decomposition", *SIAM J. Sci. Statist. Comput.*, Vol. 7, 1986, pp. 1126–1146.

[14] C.C. Paige and M. Saunders, "Towards a generalized singular value decomposition", *SIAM J. Numer. Anal.*, Vol. 18, No. 3, 1981, pp. 398–405.

[15] S. Van Huffel, "Analysis and properties of the generalized total least squares problem $AX \approx B$ when some or all columns in A are subject to error", *SIAM J. Matrix Anal. Appl.*, Vol. 10, No. 3, 1989, pp. 294–315.