Parallel SVD–Updating Using Approximate Rotations

Jürgen Götze
Dept. of Electrical and Computer Engineering
Rice University, Houston, Texas 77251-1892, U.S.A
email: jugo@ece.rice.edu

Peter Rieder and Josef A. Nossek
Institute of Network Theory and Circuit Design
Technical University of Munich, Germany
email: peri@nws.e-technik.tu-muenchen.de

Abstract

In this paper a parallel implementation of the SVD–updating algorithm using approximate rotations is presented. In its original form the SVD–updating algorithm had numerical problems if no reorthogonalization steps were applied. Representing the orthogonal matrix V (right singular vectors) using its parameterization in terms of the rotation angles of $n(n-1)/2$ plane rotations these reorthogonalization steps can be avoided during the SVD-updating algorithm [18]. This results in a SVD–updating algorithm where all computations (matrix vector multiplication, QRD–updating, Kogbetliantz's algorithm) are entirely based on the evaluation and application of orthogonal plane rotations. Therefore, in this form the SVD–updating algorithm is amenable to an implementation using CORDIC–based approximate rotations.

Using CORDIC–based approximate rotations the $n(n-1)/2$ rotations representing V (as well as all other rotations) are only computed to a certain approximation accuracy (in the basis $\arctan 2^i$). All necessary computations required during the SVD–updating algorithm (exclusively rotations) are executed with the same accuracy, i.e., only $r \ll w$ (w: wordlength) elementary orthonormal μ–rotations are used per plane rotation. Simulations show the efficiency of the implementation using CORDIC–based approximate rotations.

1 Introduction

By computing the singular value decomposition (SVD) of an $m \times n$ data matrix it is possible to extract the signal and noise subspaces of the data. The knowledge of these subspaces is essential in many application fields, e.g. DOA–estimation [16], state–space system identification [13].

In practice, where the problems are usually time varying, it is important to be able to track these subspaces. Therefore, in recent years various subspace tracking algorithms have been proposed. These algorithms are based on rank revealing decompositions [3, 17], the Lanczos algorithm [4] or the SVD–updating algorithm [14].

The SVD–updating algorithm incorporates a new data vector by a matrix vector multiplication, a QRD–updating step and a fraction of a sweep of Kogbetliantz's SVD algorithm [14, 6]. These types of computations are well suited for parallel implementations and it has been shown in [15], that all the computations can nicely be combined to a systolic algorithm and architecture for SVD–updating.

However, this original version of the SVD–updating algorithm [14] exhibits numerical problems since due to the round–off error accumulation the orthogonality of the singular vectors is lost. Reorthogonalization steps can avoid this problem but are not appropriate for a systolic implementation. In [18] this problem was solved by using the parameterization of the matrix of right singular vectors by $n(n-1)/2$ plane rotations and updating the respective rotation angles. In this form the SVD–updating algorithm is entirely based on the evaluation and application of plane rotations.

These plane rotations can be implemented in the standard way using square roots and divisions or transzendental functions. It has been shown in [9] that the use of approximate rotations is worthwile in order to avoid square root computations or square root and division computations without degrading the performance of the SVD–updating algorithm. With respect to an efficient VLSI implementation another widely used method for the implementation of the rotations is the CORDIC [19, 20]. Approximate rotations based on the idea of CORDIC, i.e., representing the rotation angle Φ in the basis $\arctan 2^{-i}$ with digits $f_i \in \{+1, -1\}$ ($\Phi = \sum_i f_i \arctan 2^{-i}$), are advantageous for Jacobi–type methods [12] and QRD-updating [8]. In [10] a method was derived that allows the evaluation of the optimal μ–rotation angle (i.e. $\Phi_i = \arctan 2^{-i}$) using μ–rotations as well. An elementary architecture for evaluationg and applying the μ–rotations was presented in [11].

In this paper we demonstrate the efficiency of approximate rotations based on simple CORDIC–type μ–rotations for the SVD–updating algorithm. This requires the extension of the ideas presented in [12] to the SVD, i.e., the use of approximate CORDIC–type μ–rotation is discussed for Kogbetliantz's SVD algorithm and applied to the SVD–updating algorithm as given in [18]. Note, that only this numerically stable version of the SVD–updating algorithm is appropriate with respect to a formulation of the entire algorithm based on CORDIC–type μ–rotations, since only this version is entirely based on plane rotations. Simulations show that very coarse approximations, i.e. using r μ–rotations ($r \ll w$, w being the wordlength) per plane rotation works as well as using exact rotations (i.e. $r = w$ for the exact CORDIC).

In section 2 we present some preliminaries. First of all the definition of orthogonal plane rotations and their implementation using CORDIC is given. Then, the linear algebra algorithms composing the SVD–updating algorithm are reviewed, i.e., the QRD–updating and the SVD using Kogbetliantz's algorithm. Section 3 presents the SVD–updating algorithm as given in [18], requiring only applications and evaluations of plane rotations throughout the algorithm. Section 4 reviews the idea of using approximate rotations. Approximate rotations based on CORDIC–type μ–rotations are used for the SVD–updating algorithm in section 5. In section 6 simulations show the efficiency of the presented algorithm and section 7 concludes the paper.

2 Preliminaries

In this section the matrix decompositions (QRD,SVD) as required for the SVD–updating algorithm are defined and their computation as appropriate for the SVD–updating algorithm (QRD-updating and SVD computations using Kogbetliantz's algorithm) are reviewed. These algorithms are based on the evaluation and application of plane rotations. The implementation of orthonormal plane rotations using CORDIC is also discussed.

2.1 Orthogonal Rotations

Definition 1 (Orthonormal Plane Rotation) *A plane rotation $\boldsymbol{G}_{pq}(\Phi) \in R^{n\times n}$ is defined by the rotation angle Φ and the (p,q)-plane in which the rotation takes place. $\boldsymbol{G}_{pq}(\Phi)$ is given by the embedding of $\cos(\Phi)$ and sin(Φ) in the (pp,pq,qp,qq) positions of an $n \times n$ identity matrix as follows:*

$$\boldsymbol{G}_{pq}(\Phi) = \begin{array}{c} \\ \\ \\ \\ \\ \\ \\ \\ \end{array} \begin{array}{cc} \begin{array}{ccccccc} & & p & & q & & \end{array} & \\ \left[\begin{array}{ccccccc} 1 & & \vdots & & \vdots & & 0 \\ & \ddots & \vdots & & \vdots & & \\ \cdots & \cdots & \cos(\Phi) & \cdots & \sin(\Phi) & \cdots & \cdots \\ & & \vdots & \ddots & \vdots & & \\ \cdots & \cdots & -\sin(\Phi) & \cdots & \cos(\Phi) & \cdots & \cdots \\ & & \vdots & & \vdots & \ddots & \\ 0 & & \vdots & & \vdots & & 1 \end{array}\right] & \begin{array}{c} \\ \\ p \\ \\ q \\ \\ \\ \end{array} \end{array} . \quad (1)$$

$\boldsymbol{G}_{pq}(\Phi)$ is an orthogonal rotation, since $\boldsymbol{G}_{pq}^T(\Phi)\boldsymbol{G}_{pq}(\Phi) = \boldsymbol{I}$.

Without loss of generality we will only consider in detail the evaluation and application of 2×2 orthogonal rotations.

$$\boldsymbol{G}(\Phi) = \begin{bmatrix} \cos\Phi & \sin\Phi \\ -\sin\Phi & \cos\Phi \end{bmatrix} = \frac{1}{\sqrt{1+\tan^2\Phi}} \begin{bmatrix} 1 & \tan\Phi \\ -\tan\Phi & 1 \end{bmatrix} \quad (2)$$

in the following. A vector $[x,y]^T$ is rotated by Φ by

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \boldsymbol{G}(\Phi) \begin{bmatrix} x \\ y \end{bmatrix} . \quad (3)$$

Frequently, Φ is computed such that $y' = 0$, i.e.,

$$\Phi = \arctan\frac{y}{x}. \quad (4)$$

Various possibilities for the implementation of orthogonal plane rotations have been presented, e.g. square root free or square and division free rotations. With respect to an ASIC implementation CORDIC is another widely used method for rotation computations.

2.2 CORDIC

The CORDIC procedure [19, 20] can be interpreted as a representation of the desired rotation angle Φ in an "arctan 2^{-k}" number system with digits $f_k \in \{-1, 1\}$. Hence, after $w+1$ iteration steps the input vector $[x,y]^T$ is rotated by Φ with a precision of w bits. In matrix notation the CORDIC algorithm is governed by

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \frac{1}{K_w}\prod_{k=0}^{w} \begin{bmatrix} 1 & \text{sign}(\tau_k)2^{-k} \\ -\text{sign}(\tau_k)2^{-k} & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}, \quad (5)$$

where the scaling factor $\frac{1}{K_w}$ is independent of the rotation angle.

$$\frac{1}{K_w} = \prod_{k=0}^{w} \frac{1}{\sqrt{1+2^{-2k}}} \quad (6)$$

There have been efforts to eliminate the scaling factor or at least to bring it into a simple binary representation. Delosme [5] proposed a method for computing a variable scaling factor on–line. This case arises for variable iteration bounds in (5). Let an elementary rotation by angle Φ_k be composed of twice executing a rotation by $\Phi_k/2$. A double rotation is given by

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \frac{1}{K_w^2} \prod_{k=0}^{w} \begin{bmatrix} 1-2^{-2k} & \mathrm{sign}(\tau_k)2^{-k+1} \\ -\mathrm{sign}(\tau_k)2^{-k+1} & 1-2^{-2k} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}. \tag{7}$$

Now, four shift and four add operations are required per iteration step but the scaling factor is square root free. To avoid the division, too, the following simple identity can be used:

$$\frac{1}{1+2^{-2k}} = (1-2^{-2k})(1+2^{-4k})(1+2^{-8k})\cdots \tag{8}$$

We will elaborate this further when approximate rotations are discussed.

2.3 QR–Decomposition

Definition 2 (QR–decomposition) *The QR-decomposition of a matrix $\boldsymbol{X} \in R^{m\times n} (m \geq n)$ is defined by*

$$\boldsymbol{X} = \boldsymbol{Q} \begin{bmatrix} \boldsymbol{R} \\ \boldsymbol{O} \end{bmatrix}, \tag{9}$$

where $\boldsymbol{Q} \in R^{m\times m}$ is orthogonal ($\boldsymbol{Q}^T\boldsymbol{Q} = \boldsymbol{I}$) and $\boldsymbol{R} \in R^{n\times n}$ is upper triangular.

The triangular matrix R is obtained by applying a sequence of rotations $\boldsymbol{G}_{pq}(\Phi)$ to the matrix $\boldsymbol{X}$, where $\boldsymbol{G}_{pq}(\Phi)$ annihilates the x_{pq} element for $1 < q \leq n$ and $q+1 < p \leq m$, i.e.,

$$\boldsymbol{Q} = \prod_{p,q} \boldsymbol{G}_{pq}(\Phi) \tag{10}$$

An alternative for triangularizing $\boldsymbol{X}$ columnwise is to perform the triangularization row by row. This yields the recursive QRD–updating. Let $\boldsymbol{X}_{[k-1]}$ be the $k-1 \times n$ data matrix available at time step $k-1$ and $\boldsymbol{x}_{[k]}^T$ be the new data vector measured at time step k, one obtains

$$\boldsymbol{X}_{[k]} = \begin{bmatrix} \lambda \boldsymbol{X}_{[k-1]} \\ \boldsymbol{x}_{[k]}^T \end{bmatrix}, \tag{11}$$

where λ is the forgetting factor.

Given the QRD of $\boldsymbol{X}_{[k-1]}$

$$\boldsymbol{X}_{[k-1]} = \boldsymbol{Q}_{[k-1]} \begin{bmatrix} \boldsymbol{R}_{[k-1]} \\ \boldsymbol{O} \end{bmatrix}, \tag{12}$$

the upper triangular factor $\boldsymbol{R}_{[k]}$ is obtained by appending the new data vector $\boldsymbol{x}_{[k]}^T$ to the weighted matrix $\lambda\boldsymbol{R}_{[k-1]}$ and using a sequence of Givens rotations $\boldsymbol{G}_{pq}(\Phi)$ ($p = k; 1 \leq q \leq n$) to annihilate the appended row, i.e.,

$$\begin{bmatrix} \boldsymbol{R}_{[k]} \\ \mathbf{o}^T \end{bmatrix} \leftarrow \prod_{i=1}^{n} \boldsymbol{G}_{ki}(\Phi) \begin{bmatrix} \lambda\boldsymbol{R}_{[k-1]} \\ \boldsymbol{x}_{[k]}^T \end{bmatrix}. \tag{13}$$

2.4 Singular–Value–Decomposition

Definition 3 (SVD) *The SVD of a matrix $\boldsymbol{X} \in R^{m\times n}$ is defined by*

$$\boldsymbol{X} = \boldsymbol{U\Sigma V}^T \tag{14}$$

where $\boldsymbol{U} \in R^{m\times m}$ and $\boldsymbol{V} \in R^{n\times n}$ are orthogonal matrices ($\boldsymbol{U}^T\boldsymbol{U} = \boldsymbol{I}$, $\boldsymbol{V}^T\boldsymbol{V} = \boldsymbol{I}$) and $\boldsymbol{\Sigma} = diag(\sigma_1, \ldots, \sigma_n)$ is an $m \times n$ diagonal matrix containing the singular values σ_i.

With respect to a parallel implementation Kogbetliantz's algorithm is the method of choice for computing the SVD. It is well known that it is advantageous to apply Kogbetliantz's SVD algorithm to the upper triangular matrix $\boldsymbol{R}$ obtained by a preparatory QRD $\boldsymbol{X} = \boldsymbol{Q}\begin{bmatrix}\boldsymbol{R}\\ \boldsymbol{O}\end{bmatrix}$ (instead of applying it to $\boldsymbol{X}$ itself). The triangular Kogbetliantz SVD algorithm is given as follows:

```
for $l$=0,1,2,...
   for all index pairs $(p,q)$
```

$$\begin{array}{lcl} \boldsymbol{R} & \leftarrow & \boldsymbol{U}^T_{p,q,l}(\Phi_U)\cdot\boldsymbol{R}\cdot\boldsymbol{V}^T_{p,q,l}(\Phi_V) \\ \boldsymbol{V} & \leftarrow & \boldsymbol{V}\cdot\boldsymbol{V}_{p,q,l}(\Phi_V) \\ \boldsymbol{U} & \leftarrow & \boldsymbol{U}\cdot\boldsymbol{U}_{p,q,l}(\Phi_U) \end{array} \tag{15}$$

where $\boldsymbol{U}_{p,q,l}(\Phi_U)$ and $\boldsymbol{V}_{p,q,l}(\Phi_V)$ are the plane rotations in the (p,q)–plane during the l–th iteration. For the index pairs (p,q) a cyclic–by–row ordering scheme is used, i.e.,

$$(p,q) = (1,2),(1,3),\ldots,(1,n)(2,3),\ldots,(2,n),\ldots,(n-1,n). \tag{16}$$

The cyclic ordering scheme preserves the triangular structure of the initial matrix $\boldsymbol{R}$ such that Kogbetliantz's algorithm must only work with triangular matrices throughout the whole algorithm. The execution of all $n(n-1)/2$ pairs of (16) is called a sweep (l–th sweep).

The plane rotations $\boldsymbol{U}_{p,q,l}(\Phi_U)$ and $\boldsymbol{V}_{p,q,l}(\Phi_V)$ have to be determined such that the matrix $\boldsymbol{R}$ converges to a diagonal matrix (i.e. $\boldsymbol{\Sigma}$). Therefore, the off–diagonal quantity

$$S = \sqrt{||\boldsymbol{R}||_F^2 - \sum_{i=1}^{n} r_{ii}^2} \tag{17}$$

must be reduced by each transformation (15). This is usually obtained by annihilating the matrix entry r_{pq}. Again without restiction of generality we consider the 2×2 subproblem

$$\begin{bmatrix} r'_{pp} & 0 \\ 0 & r'_{qq}\end{bmatrix} = \begin{bmatrix} \cos\Phi_U & \sin\Phi_U \\ -\sin\Phi_U & \cos\Phi_U\end{bmatrix}\begin{bmatrix} r_{pp} & r_{pq} \\ 0 & r_{qq}\end{bmatrix}\begin{bmatrix} \cos\Phi_V & \sin\Phi_V \\ -\sin\Phi_V & \cos\Phi_V\end{bmatrix}. \tag{18}$$

It has been shown in [21, 2], that the angles Φ_U and Φ_V can be determined using two angles Φ_1 and Φ_2 that can be computed independent of each other:

$$\Phi_1 = \arctan\frac{-r_{12}}{r_{11}+r_{22}} \qquad \Phi_2 = \arctan\frac{r_{12}}{r_{22}-r_{11}} \tag{19}$$

$$\Phi_U = \frac{1}{2}(\Phi_2 - \Phi_1) \qquad \Phi_V = \frac{1}{2}(\Phi_2 + \Phi_1). \tag{20}$$

3 SVD–Updating Algorithm

The SVD–updating algorithm is based on a matrix vector multiplication, a QRD-updating step, and the computation of the SVD by the triangular Kogbetliantz algorithm. Let $\boldsymbol{X}_{[k-1]} = \boldsymbol{U}_{[k-1]}\boldsymbol{\Sigma}_{[k-1]}\boldsymbol{V}_{[k-1]}$ be the factorization of $\boldsymbol{X}_{[k-1]}$ at time step $k-1$ and $\boldsymbol{x}_{[k]}^T$ be the new data vector. In order to put the QRD–updating and the SVD–computation together it is necessary to multiply the new data vector $\boldsymbol{x}_{[k]}^T$ by the already computed matrix of right singular vectors $\boldsymbol{V}_{[k-1]}$

$$\tilde{\boldsymbol{x}}_{[k]}^T \leftarrow \boldsymbol{x}_{[k]}^T \boldsymbol{V}_{[k-1]}\,. \tag{21}$$

Then, the QRD–updating is executed using $\tilde{\boldsymbol{x}}_{[k]}^T$ as the appended vector:

$$\begin{bmatrix} \tilde{\boldsymbol{R}}_{[k]} \\ \mathbf{o}^T \end{bmatrix} \leftarrow \prod_{i=1}^{n} \boldsymbol{G}_{ki}(\Phi) \begin{bmatrix} \lambda \boldsymbol{R}_{(k-1)} \\ \tilde{\boldsymbol{x}}_{[k]}^T \end{bmatrix} \tag{22}$$

Now, the SVD of $\tilde{\boldsymbol{R}}_{[k]}$ is computed using Kogbetliantz's algorithm. In order to reduce the complexity of the Kogbetliantz's algorithm it was shown in [6, 14] that one sweep or even a fraction of a sweep of Kogbetliantz's SVD algorithm is sufficient to track the subspace. Annihilating only the matrix elements $\tilde{r}_{i,i+1}(i = 1, \ldots, n)$ after each update, i.e.

$$\boldsymbol{R}_{[k]} \quad \leftarrow \quad \prod_{i=1}^{n-1} \boldsymbol{U}_{i,i+1,[k]}^T(\Phi_U) \cdot \tilde{\boldsymbol{R}}_{[k]} \cdot \prod_{i=1}^{n-1} \boldsymbol{V}_{i,i+1,[k]}(\Phi_V) \tag{23}$$

$$\boldsymbol{V}_{[k]} \quad \leftarrow \quad \boldsymbol{V}_{[k-1]} \cdot \prod_{i=1}^{n-1} \boldsymbol{V}_{i,i+1,[k]}(\Phi_V) \tag{24}$$

also enables a regular implementation of the SVD–updating on a systolic array [15]. In this form ((21)(22)(23)) the SVD–updating algorithm requires a reorthogonalization of the matrices $\boldsymbol{V}_{[k]}$. This reorthogonalization can be avoided by parameterizing the $\boldsymbol{V}_{[k]}$ in terms of $n(n-1)/2$ orthogonal rotations and updating the respective rotation angles [18]. Now, the matrix vector multiplication (21) can also be refered to the application of rotations. Therefore, the SVD–updating algorithm as presented in [18], is completely based on the evaluation and application of orthogonal rotations. This enables the application of approximate rotations throughout the whole algorithm.

4 Approximate Rotations

While the execution of an exact rotation as described in (3), (4) guaranees $y' = 0$, an approximate rotation $\boldsymbol{G}(\tilde{\Phi})$, defined by an approximate rotation angle $\tilde{\Phi} \approx \Phi$ only ensures

$$\mid y' \mid = \mid d \mid \cdot \mid y \mid \qquad \text{with} \quad 0 \leq \mid d \mid < 1. \tag{25}$$

The reduction factor d depends on $\tau = x/y$ and $\tilde{t} = \tan\tilde{\Phi}$ as follows:

$$d(\tau, t) = \frac{1 - \tau \cdot \tilde{t}}{\sqrt{1 + \tilde{t}^2}}. \tag{26}$$

Obviously, for the exact rotation where $\tilde{t} = \tan y/x$ one obtains $d = 0$.

At this point having defined an approximate rotation we make use of the idea of CORDIC [19], i.e. with respect to a simple implementation of the rotation we restrict ourselves to the set of approximate angles

$$\tilde{\Phi} = \Phi_i = \arctan 2^{-i}, \tag{27}$$

where $i \in \mathcal{I} = \{0, 1, 2, \ldots, b\}$($b$ = wordlength). Therefore, we only allow rotations of the form

$$\boldsymbol{G}(\Phi_i) = \frac{1}{K_i}\boldsymbol{G}_u(i) = \frac{1}{\sqrt{1+2^{-2i}}}\begin{bmatrix} 1 & \sigma 2^{-i} \\ -\sigma 2^{-i} & 1 \end{bmatrix}., \tag{28}$$

where K_i is the scaling factor and $\boldsymbol{G}_u(i)$ is an (unscaled) μ–rotation.

The optimal CORDIC–based approximate rotation given by the optimal angle Φ_l (optimal angle index l) is defined by the CORDIC angle $\Phi_i (i \in \mathcal{I})$ which is closest to the exact rotation angle Φ, i.e.,

$$\mid \Phi_l - \Phi \mid = \min_{i \in \mathcal{I}} \mid \Phi_i - \Phi \mid . \tag{29}$$

This optimal Φ_l can be determined following the ideas presented in [10, 12].

Let $\mathrm{man}(a)$ and $\exp(a)$, repectively, denote the mantissa and the exponent of a binary floating point number a.

- According to $\tan \Phi = y/x \approx 2^{-l}$ an estimate for l can be obtained as

$$l_e = \exp(y) - \exp(x). \tag{30}$$

- Since $\mathrm{man}(y)/\mathrm{man}(x) \in [0.25, 1[$ one obtains

$$l \in \mathcal{I} = \{l_e, l_e + 1, l_e + 2\} \tag{31}$$

- By computing $[\hat{x}, \hat{y}]^T = \boldsymbol{G}_u(i) \cdot [x, y]^T$ with $i \in \mathcal{I}$ and checking the sign of $\hat{y}$ the optimal value for l can be identified [10]. Note, that here only unscaled rotations are necessary.

This procedure guarantees $\mid d(\tau, l) \mid \leq 1/3$. In virtue of an easy scaling factor compensation the approximate rotation is slightly changed by using a double rotation with $l = l + 1$ in order to avoid the square root in the scaling factor [12]. The unscaled approximate double rotation $\boldsymbol{G}_{du}(l)$ is

$$\boldsymbol{G}_{du}(l) = \begin{bmatrix} 1 - 2^{-2l} & \sigma 2^{-l+1} \\ -\sigma 2^{-l+1} & 1 - 2^{-2l} \end{bmatrix}. \tag{32}$$

The scaled double rotation is $\boldsymbol{G}_{ds}(l) = \frac{1}{K_l^2}\boldsymbol{G}_{du}(l)$, where $1/K_l^2 = 1/(1 + 2^{-2l})$ can recursively be computed by shift–and–add operations as follows [12] (see 8):

$$K_l^2 = (1 - 2^{-2l}) \prod_{i=1}^{b} (1 + 2^{-2^{i+1}l}) \quad \text{with} \quad b = log_2[\frac{w}{2l}]. \tag{33}$$

The number of shift and add operations for scaling decreases for increasing values of l (small angles), e.g. for $l > b/2$ no scaling is required at all. It is even possible to further reduce the effort for scaling by using different kinds of orthonormal μ–rotations [10]. This double rotation method slightly changes the original approximation and yields $\mid d(\mid \tau \mid, l) \mid < 0.51$ [8].

The CORDIC–based approximate rotation can also be used to actually generate $y' = 0$ by applying the described procedure iteratively. Thereby, only the CORDIC angles, which are really necessary to generate $y' = 0$, are executed while the original CORDIC uses the complete sequence of w CORDIC angles (suppose a small angle Φ, then CORDIC also executes the large angles although they do not contribute to the reduction of y).

5 SVD–Updating Using Approximate Rotations

Now, we examine the SVD–updating algorithm for its performance using approximate rotations.

QRD–updating. Using approximate rotations, results in an iterative version of the exact QR–algorithm [8]. Several sweeps are necessary to achieve a sufficient approximation of the exact QR–decomposition. Another possibility is to improve the accuracy of the orthogonal plane rotation by increasing the number of orthogonal μ–rotations per plane rotation. This version is applied to the QRD–updating.

Kogbetliantz's SVD algorithm. Approximations of the Kogbetliantz's SVD algorithm have already been examined. In [14, 15, 18] only a fraction of one sweep is executed per QRD–update without remarkable deterioration of the results. Using approximate rotations, i.e., annihilating the off–diagonal elements not completely by setting them to zero but only reducing them, is a further approximation of the Kogbetliantz algorithm.

Important in this context is, that the off–diagonal–quantity S of (17) must be reduced with each 2x2–rotation. Using the exact angles Φ_U and Φ_V (18), the matrix $\boldsymbol{R}$ is diagonalized. Using a few (say $r \ll w$) orthonormal μ–rotations representing $\tilde{\Phi}_U$ and $\tilde{\Phi}_V$, $\boldsymbol{R}$ is rotated to $\boldsymbol{R}'$. Since using approximate rotations based on CORDIC–type μ–rotations does not preserve the upper triangular structure we consider a 2×2 subproblem where $r_{21} \neq 0$ (usually small).

$$\begin{bmatrix} r_{11}' & r_{12}' \\ r_{21}' & r_{22}' \end{bmatrix} = \begin{bmatrix} \cos\tilde{\Phi}_U & \sin\tilde{\Phi}_U \\ -\sin\tilde{\Phi}_U & \cos\tilde{\Phi}_U \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix} \begin{bmatrix} \cos\tilde{\Phi}_V & \sin\tilde{\Phi}_V \\ -\sin\tilde{\Phi}_V & \cos\tilde{\Phi}_V \end{bmatrix} \tag{34}$$

In order to find the approximate rotations, we refer to (19), (20). The advantage of this method is, that Φ_1 and Φ_2 can be computed separately of each other. Therefore, it is possible to find the nearest approximate rotation based on $\tilde{\Phi}_1$ and $\tilde{\Phi}_2$. With

$$\begin{aligned} x_1 &= (r_{22} + r_{11})/2; \quad x_2 = (r_{22} - r_{11})/2; \\ y_1 &= (r_{21} - r_{12})/2; \quad y_2 = (r_{21} + r_{12})/2; \end{aligned} \tag{35}$$

we determine $\tilde{\Phi}_1$ and $\tilde{\Phi}_2$ using the method outlined in section 4. Therefore, we obtain

$$\begin{bmatrix} x_1' \\ y_1' \end{bmatrix} = \boldsymbol{G}(\tilde{\Phi}_1) \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}; \qquad \begin{bmatrix} x_2' \\ y_2' \end{bmatrix} = \boldsymbol{G}(\tilde{\Phi}_2) \begin{bmatrix} x_2 \\ y_2 \end{bmatrix}; \tag{36}$$

Note, that using Φ_1 and Φ_2 as obtained from (19), (20) would yield $y_1' = y_2' = 0$. However, by using approximate rotations, y_1' and y_2' are not completely set to zero but one obtains

$$y_1' = d_1 y_1; \qquad y_2' = d_2 y_2. \tag{37}$$

where $0 <| d_1 |, | d_2 |< d_{max} = 0.51$ (see section 4). Since

$$r_{12}' = -y_1' + y_2' = -d_1 y_1 + d_2 y_2 = -d_1(r_{21} - r_{12})/2 + d_2(r_{21} + r_{12})/2 \tag{38}$$

$$r_{21}' = y_1' + y_2' = d_1 y_1 + d_2 y_2 = d_1(r_{21} - r_{12})/2 + d_2(r_{21} + r_{12})/2 \tag{39}$$

it is easy to show that

$$r_{12}'^2 + r_{21}'^2 \leq d_{max}^2 (r_{12}^2 + r_{21}^2) \leq r_{12}^2 + r_{21}^2 \, . \tag{40}$$

This condition is sufficient for the convergence of Kogbetliantz's SVD algorithm [7, 9].

Representation and updating of singular vectors. At time step $k-1$ $\boldsymbol{V}(k-1)$ is described by the $n(n-1)/2$ angles of the systolic processor array by Bojanczyk et al. [1]. In order to get $\boldsymbol{V}(k)$, $\boldsymbol{V}(k-1)$ is updated with the $n-1$ rotations $\boldsymbol{V}_{i,i+1,[k]}(\Phi_V)$, i.e. the angles have to be updated. In [18] a method for executing this updating is proposed. It also refers the matrix vector multiplication to applications of rotations. Storing each angle in the basis $\arctan 2^{-i}$ with $f_i = \{0, 1\}$ this method can easily be extended to the use of approximate rotations.

6 Simulations

The performance of the algorithms is analysed for two different kinds of signals:

- $s_1(t)$ is a signal where the frequency jumps suddenly
- $s_2(t)$ is a frequency modulate signal

In both cases SNR= 20dB holds. The frequencies are estimated in each time step using the ESPRIT algorithm based on the significant subspaces as obtained from the singular vectors of $\boldsymbol{V}_{[k]}$ belonging to the dominant singular values. Figure 1 (jump of frequency) and Figure 2 (modulation of frequency) compare the performance of the SVD–updating algorithm using approximate rotations to the SVD–updating algorithm using exact rotations. Both figures are organized as follows. The dotted line in each subplot shows the estimated frequency if a complete SVD is executed at each time step (i.e. MATLAB: call svd($\boldsymbol{X}_{[k]}$) for each k). The solid lines show the estimated frequencies using the SVD–updating algorithm with exact rotations (upper plot) and the SVD–updating algorithm using approximate rotations composed of 2 μ–rotations (middle plot) resp. 3 μ–rotations (lower plot).

7 Conclusions

In this paper it was shown that the use of approximate rotations based on CORDIC–type μ–rotations is particularly well suited for the SVD–updating algorithm. For example, slowly time varying processes correspond to small rotation angles which is especially advantageous for approximate rotations. Compared to the original CORDIC (exact rotation) only a fraction of the shift–and–add operations (fraction of the number of μ–rotations) is required to obtain essentially the same performance. It is even possible to reduce the computational effort further by adaptively changing the number of orthogonal μ–rotations executed per rotation depending on the time variance of the analysed signal. For example, during the time where the respective frequency of $s_1(t)$ is constant approximating the rotations with one μ–rotation would be sufficient. Only when the frequency jumps 2 resp. 3 μ–rotations are required for approximating the exact rotations to a sufficient accuracy.

References

[1] A. Bojanczyk, R.P. Brent, and H.T. Kung. Numerically Stable Solution of Dense Systems of Linear Equations Using Mesh–Connected Processors. *SIAM J. Sci & Stat. Comput*, 1:95–104, 1984.

[2] J.R. Cavallaro and F.T. Luk. CORDIC Arithmetic for an SVD Processor. *J. Parallel & Distributed Computing*, 5:271–290, 1988.

[3] T.F. Chan. Rank Revealing QR Factorization. *Linear Algebra and Its Applications*, 89:67–82, 1987.

[4] P. Comon and G.H. Golub. Tracking a Few Extreme Singular Values and Vectors in Signal Processing. *Proceedings of the IEEE*, 78:1327–1343, 1990.

[5] J.-M. Delosme. CORDIC Algorithms: Theory and Extensions. In *Proc. SPIE Advanced Alg. and Arch. for Signal Processing IV*, volume 1152, pages 131–145, 1989.

[6] W. Ferzali and J.G. Proakis. Adaptive SVD Algorithm for Covariance Matrix Eigenstructure Computation. In *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, pages 2615–2618, Toronto (Canada), 1990.

[7] G. Forsythe and P. Henrici. The Cyclic Jacobi Method for Computing the Pricipal Values of a Complex Matrix. *Trans. Amer. Math. Soc.*, 94:1–23, 1960.

[8] J. Götze. An Iterative Version of the QRD for Adaptive RLS Filtering. In *Proc. SPIE's Int. Conf. on Advanced Signal Processing: Algorithms, Architectures and Implementations*, San Diego (USA), 1994.

[9] J. Götze. On the Parallel Implementation of Jacobi and Kogbetliantz Algorithms. *SIAM J. on Sci. Comput.*, 15:1331–1348, 1994.

[10] J. Götze and G.J. Hekstra. Adaptive Approximate Rotations for EVD. In M. Moonen and F. Catthoor, editors, *In Algorithms and Parallel VLSI Architectures*, Leuven, 1994.

[11] J. Götze and G.J. Hekstra. An Algorithm and Architecture based on Orthonormal μ–Rotations for Computing the Symmetric EVD. *VLSI–The Integration (submitted for publication)*, 1995.

[12] J. Götze, S. Paul, and M. Sauer. An Efficient Jacobi–Like Algorithm for Parallel Eigenvalue Computation. *IEEE Trans. on Computers*, 42:1058–1065, 1993.

[13] M. Moonen, B. De Moor, L. Vandenberghe, and J. Vandewalle. On– and Off–Line Identification of Linear State–Space Models. *Int. J. Control*, 49:219–232, 1989.

[14] M. Moonen, P. van Dooren, and J. Vandewalle. A Singular Value Decomposition Updating Algorithm for Subspace Tracking. *SIAM J. Matrix Anal. Appl.*, 13:1015–1038, 1992.

[15] M. Moonen, P. van Dooren, and J. Vandewalle. A Systolic Array for SVD Updating. *SIAM J. Matrix Anal. Appl.*, 14:353–371, 1993.

[16] A. Paulraj, R. Roy, and T. Kailath. A Subspace Approach to Signal Parameter Estimation. *Proceedings of the IEEE*, 74:1044–1045, 1986.

[17] G.W. Stewart. An Updating Algorithm for Subspace Tracking. *IEEE Trans. on Signal Processing*, 40:1535–1541, 1992.

[18] F. Vanpoucke, M. Moonen, and E. Deprettere. A Numerically Stable Jacobi Array for Parallel SVD Updating. In *SPIE Advanced Signal Processing: Algorithms, Architectures and Implementations V*, volume 2296, pages 403–412, San Diego (USA), 1994.

[19] J.E. Volder. The CORDIC Trigonometric Computing Technique. *IRE Trans. Electronic Computers*, EC-8:330–334, 1959.

[20] J.S. Walter. An Unified Algorithm for Elementary Functions. In *Proc. Spring Joint Computer Conference*, volume 38, page 397. AFIPS press, 1971.

[21] B. Yang and J.F. Böhme. Reducing the Computations of the Singular Value Decomposition Array Given by Brent and Luk. *SIAM J. Matrix Anal. Appl.*, 12:713–725, 1991.

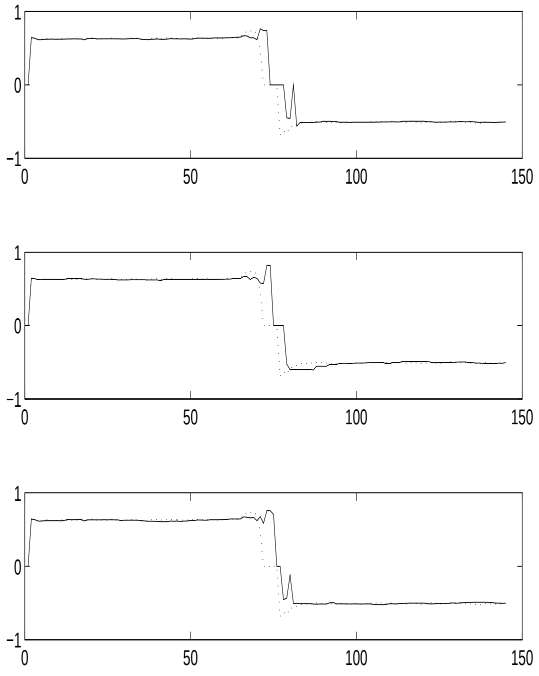

Figure 1: Frequency estimation of $s_1(t)$ using the exact SVD (dotted line), using the SVD–updating algorithm with exact rotations (upper plot, solid line), with 2 μ–rotations (middle plot, solid line), and with 3 μ–rotations per rotation (lower plot, solid line)

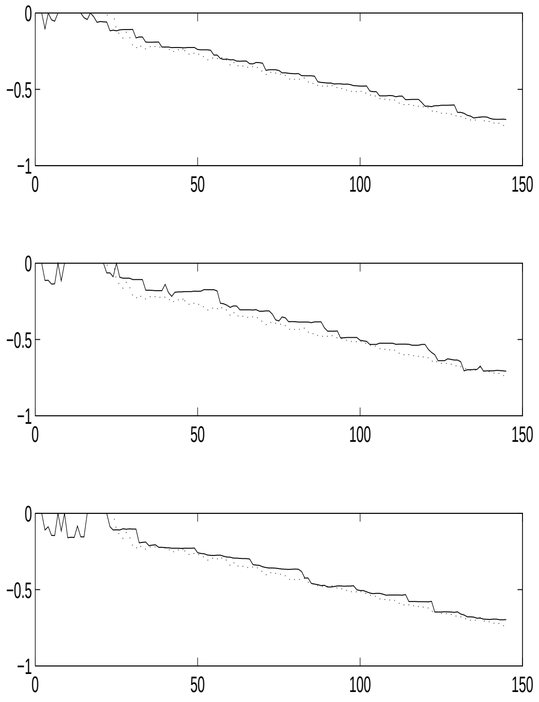

Figure 2: Frequency estimation of $s_2(t)$ using the exact SVD (dotted line), using SVD–updating algorithm with exact rotations (upper plot, solid line), with 2 μ–rotations (middle plot, solid line), and with 3 μ–rotations per rotation (lower plot, solid line)