ON THE PARALLEL IMPLEMENTATION OF JACOBI AND KOGBETLIANTZ ALGORITHMS *

JÜRGEN GÖTZE †

Abstract. *Modified Jacobi and Kogbetliantz algorithms are derived by combining methods for modifying the orthogonal rotations. These methods are characterized by the use of approximate orthogonal rotations and the factorization of these rotations. The presented new approximations exhibit better properties and require less computational cost than known approximations. Suitable approximations are applied together with factorized rotation schemes in order to gain square root free or square root and division free algorithms. The resulting approximate and factorized rotation schemes are highly suited for parallel implementations. The convergence of the algorithms is analyzed and an application in signal processing is discussed.*

Key words. Jacobi's/Kogbetliantz's algorithm, parallel algorithms, approximate and factorized rotations, convergence

AMS subject classifications. 65F15

1. Introduction.

For a real $m \times n$ $(m \geq n)$ matrix $\mathbf{A}$ the decomposition

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \qquad (\mathbf{U}^T\mathbf{U} = \mathbf{I},\ \mathbf{V}^T\mathbf{V} = \mathbf{I},\ \mathbf{\Sigma} \text{ diagonal})$$

is called the singular value decomposition (SVD) of $\mathbf{A}$. For a symmetric $\mathbf{A}$ $(\mathbf{A}^T = \mathbf{A})$ the corresponding decomposition

$$\mathbf{A} = \mathbf{W}\mathbf{\Lambda}\mathbf{W}^T \qquad (\mathbf{W}^T\mathbf{W} = \mathbf{I},\ \mathbf{\Lambda} \text{ diagonal})$$

is the eigenvalue decomposition (EVD) of $\mathbf{A}$. The methods of choice for the fast parallel computation of these decompositions are Kogbetliantz's (SVD) and Jacobi's (EVD) algorithm [3, 23], since they exhibit a significantly higher degree of parallelism than the $\mathbf{QR}$–algorithm.

It is well known [6, 20, 8, 5, 21], that it is advantageous to apply the Kogbetliantz algorithm (KA) to the triangular matrix $\mathbf{R}$ obtained from $\mathbf{A}$ by a preparatory QR-decomposition $\mathbf{A} = \mathbf{QR}$. This triangular Kogbetliantz algorithm (TKA) as well as the Jacobi algorithm (JA) only have to work with triangular matrices, since the upper triangular structure of the initial matrix (upper triangular part $(\mathbf{D} + \mathbf{T})$ of $\mathbf{A} = \mathbf{T}^T + \mathbf{D} + \mathbf{T}$ for the JA; upper triangular matrix $\mathbf{R}$ of the QR-decomposition $\mathbf{A} = \mathbf{QR}$ for the TKA) can be preserved during the algorithms.

According to this upper triangular structure one sweep of the JA and the TKA can be implemented on an upper triangular array of processors with nearest neighbor interconnections [24, 6]. The different parallel implementations of the TKA (JA) are distinguished by an algorithm for evaluating the rotations and by an ordering scheme for the rotations. Here, we are mainly interested in deriving efficient algorithms for evaluating the rotations. For ordering schemes, which enable an efficient parallel implementation on multiprocessor arrays with nearest neighbor interconnections, we refer to [23, 30].

In this paper new JAs and TKAs are derived by combining methods for modifying the evaluation of the orthogonal rotations. These methods are characterized by the

* This work was supported by the German National Science Foundation

† Institute of Network Theory and Circuit Design, Technical University of Munich, Arcisstr.21, 80333 Munich, Germany (jugo@nws.e-technik.tu-muenchen.de).

use of approximate (but still orthogonal) rotation schemes and the use of factorized rotation schemes for gaining square root free [11, 12] or square root and division free [14] rotations.

The possibility of using approximate rotation schemes has already been investigated for the sequential case in [29, 10, 33]. Since in a parallel environment the arithmetic is much more costly than other components (e.g. storage access), Modi and Pryce [25] and Charlier et al. [6] have shown, that the use of approximate rotations gives worthwile speedups on parallel computers.

The second method for modifying rotations is the use of factorized rotations [11, 12, 14]. By applying these factorized rotation schemes together with suitable approximations a further speedup can be achieved and the hardware requirements of the processor arrays can significantly be decreased. For example, it is possible to obtain a square root and division free TKA/JA (this is not possible if the factorized rotation scheme is applied to the exact rotations [27]), which is essential for regular processor arrays (e.g. application specific integrated circuit (ASIC)– based processor arrays).

In section 2 we only consider the JA, since the results for the JA can easily be extended to the TKA. A brief review of the JA is given and the possibility of using approximate rotation schemes is described. New approximations are presented, which exhibit better properties and require less computational cost than the known approximations in [33, 25, 6]. The factorized rotation schemes for gaining square root free or square root and division free rotations are also reviewed. Then, the approximate rotation scheme and the factorized rotation scheme are combined for obtaining a procedure for the design of square root free or square root and division free algorithms. This section ends with the comparison of the different Jacobi rotations concerning their required operations and their suitability for a parallel implementation. In section 3 it is shown that all the results for the JA can be extended to the TKA. The same approximations can be used and the factorized rotations can be derived in the same way. It is even possible to show that using the same approximation for the TKA as for the JA results in a better overall performance of the TKA compared to the JA. In section 4 the global and the ultimate quadratic convergence of the TKA and the JA with exact/approximate rotations is analyzed. In section 5 it is shown that the approximate and factorized scheme is particularly advantageous for SVD–based subspace tracking algorithms [9, 26]. Section 6 gives some concluding remarks.

2. Jacobi's Algorithm.

2.1. Basic Algorithm. The Jacobi algorithm works by applying a sequence of orthogonal similarity transformations to the symmetric matrix $\mathbf{A}$:

$\mathbf{A}^{(0)} := \mathbf{A}$
For $k = 0, 1, 2, \ldots$

$$\mathbf{A}^{(k+1)} = \mathbf{J}_{pq}\mathbf{A}^{(k)}\mathbf{J}_{pq} \tag{2.1}$$

We assume throughout this paper, that in virtue of a parallel implementation the index pairs (p, q) are choosen in an ordering scheme equivalent to the cyclic-by-row scheme and the rotation $\mathbf{J}_{pq}$ includes the required row (left-sided rotation) and column (right-sided rotation) exchanges (see section 3 of [6] for details), i.e.

$$\mathbf{J}_{pq} = \mathbf{J}_{pq}^T = \begin{bmatrix} 1 & & & & & & 0 \\ & \ddots & & & & & \\ & & \Leftrightarrow s & & c & & \\ & & & \ddots & & & \\ & & c & & s & & \\ & & & & & \ddots & \\ 0 & & & & & & 1 \end{bmatrix} \qquad \begin{aligned} c &= \cos\phi_k \\ s &= \sin\phi_k \end{aligned} \tag{2.2}$$

Since the $\mathbf{A}^{(k)}$ are symmetric, they are completely determined by their upper triangular part, such that only one triangle of $\mathbf{A}^{(k)}$ must be processed and the off-diagonal quantity of $\mathbf{A}^{(k)}$ can be measured by

$$S^{(k)} = \sqrt{\frac{1}{2}\left[\left\|\mathbf{A}^{(k)}\right\|_F^2 \Leftrightarrow \sum_{i=1}^{n}\left(a_{ii}^{(k)}\right)^2\right]}. \tag{2.3}$$

Since $\mathbf{J}_{pq}$ is an orthogonal transformation, i.e. $\left\|\mathbf{A}^{(k+1)}\right\|_F = \left\|\mathbf{A}^{(k)}\right\|_F$, and one similarity transformation (2.1) affects only rows and columns p and q of $\mathbf{A}^{(k)}$, it is easy to verify [12] that

$$\left[S^{(k+1)}\right]^2 = \left[S^{(k)}\right]^2 \Leftrightarrow \left[\left(a_{pq}^{(k)}\right)^2 \Leftrightarrow \left(a_{pq}^{(k+1)}\right)^2\right]. \tag{2.4}$$

Obviously, the maximal reduction of $S^{(k)}$ is obtained if $a_{pq}^{(k+1)} = 0$ after the similarity transformation (2.1). This can be achieved by the following cosine-sine pair (c, s), which defines the rotation (2.2):

$$t_{ex} = \tan\phi_k = \frac{\operatorname{sign}(\tau_J)}{|\tau_J| + \sqrt{1+\tau_J^2}} \text{ with } \tau_J = \frac{a_{pp}^{(k)} \Leftrightarrow a_{qq}^{(k)}}{2a_{pq}^{(k)}} \tag{2.5}$$

$$c = \frac{1}{\sqrt{1+t_{ex}^2}} \qquad s = t_{ex}c \tag{2.6}$$

In the subsequent sections a rotation $\mathbf{J}_{pq}$ defined by (c, s) of (2.6) is called an exact Jacobi rotation.

2.2. Approximate Rotation Schemes. For the reduction of $S^{(k)}$ it is not necessary to compute the exact Jacobi rotation, but it is sufficient to compute (c, s), such that the orthogonality is preserved and

$$|a_{pq}^{(k+1)}| = |d_J||a_{pq}^{(k)}| \text{ with } 0 \le |d_J| < 1 \tag{2.7}$$

holds. Therefore, the reduction of $\left[S^{(k)}\right]^2$ is a factor $(1 \Leftrightarrow d_J^2)$ less than the reduction by the exact rotation. A Jacobi rotation with an approximate computation of (c, s) is called an approximate Jacobi rotation and is described by $\tilde{\mathbf{J}}_{pq}$.

TABLE 2.1
Formulae for the tangent of the rotation angle and accuracy of the corresponding approximations

Approximation	Accuracy
$t_{KA1} = \frac{\sigma_J}{1+\lvert\sigma_J\rvert}$	$\lvert d_J\rvert \leq 0.21$
$t_{KA2} = \sigma_J$	$\lvert d_J\rvert \leq 1$
$t_{KA3} = \frac{\sigma_J}{1+\sigma_J^2}$	$\lvert d_J\rvert \leq 1$
$t_{KA4} = \frac{\sigma_J(1+\alpha\lvert\sigma_J\rvert)}{1+\beta\lvert\sigma_J\rvert+\alpha\sigma_J^2}$ $(\beta = 2\alpha = \sqrt{2}+1)$	$\lvert d_J\rvert < 0.25$
$t_{KA5} = \begin{cases} \operatorname{sign}(\sigma_J) & \text{if } \lvert\sigma_J\rvert \geq \frac{2}{1+\sqrt{2}} \\ \frac{4\sigma_J}{4-\sigma_J^2} & \text{if } \lvert\sigma_J\rvert < \frac{2}{1+\sqrt{2}} \end{cases}$	$\lvert d_J\rvert \leq 0.6036$
$t_{NA1} = \begin{cases} \frac{\operatorname{sign}(\tau_J)}{1+\lvert\tau_J\rvert+0.5\tau_J^2} & \text{if } \lvert\tau_J\rvert \leq 1 \\ \frac{\sigma_J}{1+\sigma_J^2} & \text{if } \lvert\tau_J\rvert > 1 \ (\lvert 2\sigma_J\rvert < 1) \end{cases}$	$\lvert d_J\rvert \leq 0.035$
$t_{NA2} = \begin{cases} \operatorname{sign}(\sigma_J) & \text{if } \lvert\sigma_J\rvert \geq 1 \\ \sigma_J & \text{if } \lvert\sigma_J\rvert < 1 \end{cases}$	$\lvert d_J\rvert \leq 0.5$
$t_{NA3} = \begin{cases} \operatorname{sign}(\sigma_J) & \text{if } \lvert\sigma_J\rvert \geq 1.3982 \\ \frac{\sigma_J}{1+\sigma_J^2} & \text{if } \lvert\sigma_J\rvert < 1.3982 \end{cases}$	$\lvert d_J\rvert \leq 0.3576$
$t_{NA4} = \begin{cases} \operatorname{sign}(\sigma_J) & \text{if } \lvert\sigma_J\rvert \geq 2 \\ r\cdot\sigma_J \text{ with } r = 1/2 & \text{if } \lvert\sigma_J\rvert \geq 1 \\ r = 2/3 & \text{if } \lvert\sigma_J\rvert \geq 0.5 \\ r = 1 & \text{if } \lvert\sigma_J\rvert < 0.5 \end{cases}$	$\lvert d_J\rvert \leq 0.25$
$t_{NA5} = \begin{cases} \operatorname{sign}(\sigma_J) & \text{if } \lvert\sigma_J\rvert \geq 2 \\ \frac{1}{2}\sigma_J & \text{if } \lvert\sigma_J\rvert \geq 1 \\ \frac{\sigma_J}{1+\sigma_J^2} & \text{if } \lvert\sigma_J\rvert < 1 \end{cases}$	$\lvert d_J\rvert \leq 0.25$

Since the orthogonality has to be preserved for the approximate rotation it is convenient to establish the approximations for $t = \tan\phi = s/c$ instead of (c, s). Since (2.1) yields

$$a_{pq}^{(k+1)} = d_J a_{pq}^{(k)} \text{ with } d_J = c^2 \Leftrightarrow s^2 \Leftrightarrow 2\tau_J cs$$

we obtain

$$|d_J(t,\tau_J)| = \left|\frac{1 \Leftrightarrow 2\tau_J t \Leftrightarrow t^2}{1+t^2}\right|. \tag{2.8}$$

The maximal value of $|d_J|$ is a measure for the badness of the approximation. It is easy to verify that the exact Jacobi rotation yields $d_J(t_{ex}, \tau_J) = 0$.

In the following the known approximations (KA) of [33, 25, 6] are discussed and some new approximations (NA) are derived. The formulae for the tangent of the rotation angle of the approximations are summarized in Table 2.1.

With $\sigma_J = \frac{1}{2\tau_J}$ the KAs of [6, 25] are given by t_{KA1} (formula 1 of [25], approximation 2 of [6]), t_{KA2} (formula 3 of [25], approximation 1 of [6]) and t_{KA3} (approximation 3 of [6]). KA1 yields $|d_J| < 0.21$. However, t_{KA1} does not converge to t_{ex} for $|\tau_J| \to \infty$, although $d_J(|\tau_J| \to \infty) \to 0$ holds (i.e ultimate quadratic convergence). t_{KA2} and t_{KA3} converge to t_{ex} for $|\tau_J| \to \infty$ and therefore they yield a faster convergence than t_{KA1} as $|\tau_J|$ increases during the algorithm. However, only $|d_J| \leq 1$ can be achieved, because of the bad approximation for $|\tau_J| \to 0$ ($t_{KA2} \to \infty$, $t_{KA3} \to 0$ while $t_{ex} \to 1$).

A way to circumvent these problems is to distinguish between small and large values of $|\tau_J|$. Approximation 3 of [6] can be obtained by using the approximation $\sqrt{1+x^2} \approx 1 + \frac{1}{2}x^2$ according to the Taylor series $\sqrt{1+x^2} = 1 + \frac{1}{2}x^2 \Leftrightarrow \frac{1}{6}x^4 + \cdots$.

TABLE 2.2
Required average number of sweeps for 10 *random matrices per n*

n	ex.	KA1	KA2	KA3	KA4	KA5	NA1	NA2	NA3	NA4	NA5	√&÷ free NA4	√&÷ free NA5
10	5.9	6.8	7.1	6.5	7.0	7.0	5.9	6.3	6.0	5.9	5.9	6.0	6.1
20	6.4	7.6	9.4	7.4	7.7	8.6	6.4	7.0	6.8	6.8	6.8	6.9	6.9
30	7.0	8.0	9.9	7.7	8.2	8.8	7.0	7.5	7.0	7.2	7.0	7.1	7.0
40	7.2	8.3	9.7	8.5	8.3	9.5	7.1	8.0	7.3	7.5	7.4	7.4	7.3

However, this approximation is only accurate if $|x| < 1$. Therefore, the approximation is bad for $|2\sigma_J| > 1$. But, if we apply this approximation to

$$t_{ex} = \frac{\operatorname{sign}(\tau_J)}{|\tau_J| + \sqrt{1+\tau_J^2}} = \frac{2\sigma_J}{1+\sqrt{1+4\sigma_J^2}} \tag{2.9}$$

for $|\tau_J| < 1$ ($\sqrt{1+\tau_J^2} \approx 1 + \frac{1}{2}\tau_J^2$) and $|2\sigma_J| < 1$ ($\sqrt{1+4\sigma_J^2} \approx 1 + 2\sigma_J^2$), respectively, we obtain t_{NA1} for which (2.8) yields $|d_J(t_{NA1}, \tau_J)| < 0.035$. This NA1 requires one square root less than the exact formula (as well as KA3) and requires a comparison (KA3 requires no comparison but has a much worse $|d_J|_{max}$).

Further NAs can be derived in a similiar way. Reducing the computational cost means an increase of the maximal value of $|d_J|$ but the following NAs still provide the same advantages as NA1 in comparison to the KAs, i.e. $|d_J|_{max} \ll 1$ and $t_{NA} \to t_{ex}$ for $|\tau_J| \to 0$ *and* $|\tau_J| \to \infty$. The easiest way to fulfill these conditions are the approximations NA2 and NA3. NA3 is slightly better than NA2, since for small $|\sigma_J|$ KA3 is a better approximation than KA2. Therefore, it can be used earlier as $|\sigma_J|$ decreases during the algorithm. Since $|d_J(\operatorname{sign}(\sigma_J), \sigma_J)| = 1/2|\sigma_J| = |\tau_J|$ holds, this is also the reason for $|d_J(t_{NA3}, \sigma_J)| < |d_J(t_{NA2}, \sigma_J)|$. Better approximations (smaller $|d_J|_{max}$) can easily be obtained by inserting more cases in NA2 and NA3. This results in NA4 and NA5, respectively.

Until now we have omitted KA4 (t_{KA4} is formula 2 of [25]) and KA5 (t_{KA5} is given in [33],p.276) for the following reasons. KA4 requires greater cost and has a greater $|d_J|_{max}$ than KA1. Therefore, it is not discussed further in [25] and it has already been omitted in [6]. Wilkinson [33] has already used different cases for his approximation KA5 (as well as all NAs). However, it is easy to see that KA5 is a worse approximation and requires greater computational cost than NA2.

In Table 2.2 the average number of sweeps required by 10 random matrices is shown for all the above mentioned approximations for $n = 10, 20, 30, 40$. For all our numerical examples the algorithms are terminated, if $S^{(k)} < 10^{-12} S^{(0)}$. The better performance of the NAs in comparison to the KAs is shown in Fig. 2.1 (Fig. 2.1 a: NA2 and NA4 in comparison to KA2 and the exact rotation; Fig. 2.1 b: NA3 and NA5 in comparison to KA3 and the exact rotation). The NAs require less sweeps and less computational cost than the corresponding KAs.

Furthermore, in the case of clusters of eigenvalues, for which $|\tau_J|$ remains small during the algorithm, it is essential that not only $|d_J|$ becomes small as $|\tau_J|$ increases during the algorithm but that the approximation is also good for small values of $|\tau_J|$. These requirements are fulfilled by the NAs. Therefore, for clusters of eigenvalues the NAs yield much better results than the KAs. As an example in Table 2.3 the required number of sweeps is shown for the Hilbert matrices of dimension $n = 10, 20, 30, 40$.

2.3. Factorized Rotation Schemes. Factorizations of Givens rotations [11] have been used for the parallel implementation of the QR-decomposition by several

TABLE 2.3
Required number of sweeps for Hilbert matrix of dimension n

n	ex.	KA1	KA2	KA3	KA4	KA5	NA1	NA2	NA3	NA4	NA5	√&÷ free NA4	√&÷ free NA5
10	5	8	8	9	8	8	5	6	7	9	7	7	6
20	5	8	7	10	9	8	6	6	7	7	8	8	6
30	5	9	10	13	8	10	6	7	7	9	6	8	7
40	6	8	8	10	10	12	6	7	7	7	7	8	7

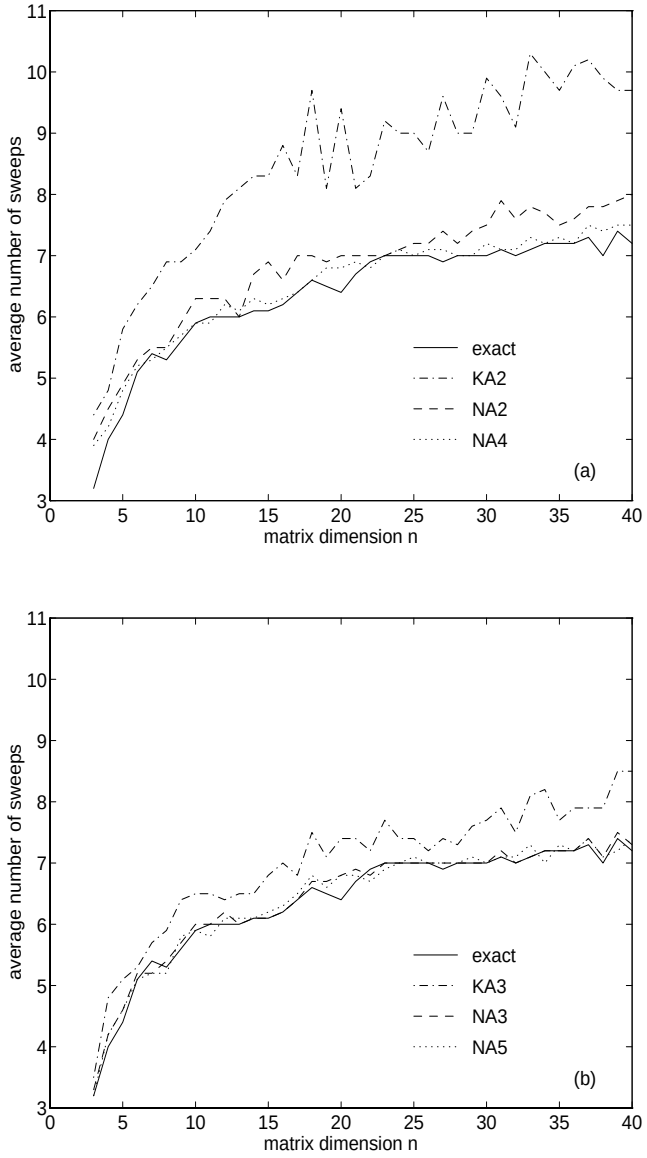

FIG. 2.1. *Average number of sweeps for* 10 *random matrices per n.*

authors [1], since the implementation of the square roots can be avoided. Recently, a square root and a division free Givens rotation has been presented [14], such that a processor array for the QR-decomposition only requires additions and multiplications and a distributed computation of the rotation factors is possible [15]. These factorized rotation schemes can also be applied to orthogonal similarity transformations [27, 16]. For that purpose the matrices $\mathbf{A}^{(k)}$ are factorized as follows:

$$\mathbf{A}^{(k)} = \left[\mathbf{Z}^{(k)}\right]^{-1/2} \mathbf{Y}^{(k)} \left[\mathbf{Z}^{(k)}\right]^{-1/2} \tag{2.10}$$

with $\mathbf{Z}^{(k)} = \mathrm{diag}\left(z_i^{(k)}\right)$. With a further diagonal matrix $\mathbf{Z}^{(k+1)} = \mathrm{diag}\left(z_i^{(k+1)}\right)$ a rotation $\mathbf{K}_{pq}$ (no matter whether it is a Givens or a Jacobi rotation) can also be described in factorized form:

$$\mathbf{K}_{pq} = \left[\mathbf{Z}^{(k)}\right]^{1/2} \mathbf{K}'_{pq} \left[\mathbf{Z}^{(k+1)}\right]^{-1/2} \tag{2.11}$$

The application of the factorized rotation scheme to the similarity transformation $\mathbf{A}^{(k+1)} = \mathbf{K}_{pq}^T \cdot \mathbf{A}^{(k)} \cdot \mathbf{K}_{pq}$ leads to the reduction of the square roots by one half, if $\mathbf{K}_{pq}$ is an exact Jacobi rotation [27] and to the complete avoidance of the square roots [27] or of the square roots and the divisions [16], if $\mathbf{K}_{pq}$ is a Givens rotation. A Givens rotation is distinguished from a Jacobi rotation, since different formulae are used to compute their rotation parameters. In contrary to Givens rotations the exact Jacobi rotations cannot be factorized, such that square roots or square roots and divisions can completely be avoided. However, if the factorized rotation schemes are applied to suitable approximations of the exact Jacobi rotation, it is possible to gain square root free or square root and division free Jacobi rotations.

The factorization of $\mathbf{K}_{pq} = \begin{bmatrix} \Leftrightarrow s & c \\ c & s \end{bmatrix}$ according to (2.11) yields:

$$\mathbf{K}_{pq} =$$

$$= \begin{bmatrix} z_q^{(k)}\left(1+t^2\right) & \\ & z_p^{(k)}\left(1+t^2\right) \end{bmatrix}^{-1/2} \begin{bmatrix} \Leftrightarrow t\sqrt{\frac{z_q^{(k)}}{z_p^{(k)}}} & 1 \\ 1 & t\sqrt{\frac{z_p^{(k)}}{z_q^{(k)}}} \end{bmatrix} \begin{bmatrix} z_p^{(k)} & \\ & z_q^{(k)} \end{bmatrix}^{1/2} \tag{2.12}$$

or

$$\mathbf{K}_{pq} =$$

$$= \begin{bmatrix} z_q^{(k)}\left(c_t^2+s_t^2\right) & \\ & z_p^{(k)}\left(c_t^2+s_t^2\right) \end{bmatrix}^{-1/2} \begin{bmatrix} \Leftrightarrow s_t\sqrt{\frac{z_q^{(k)}}{z_p^{(k)}}} & c_t \\ c_t & s_t\sqrt{\frac{z_p^{(k)}}{z_q^{(k)}}} \end{bmatrix} \begin{bmatrix} z_p^{(k)} & \\ & z_q^{(k)} \end{bmatrix}^{1/2} \tag{2.13}$$

depending on $t = s_t/c_t$ and (c_t, s_t), respectively (note, that s_t and c_t are the nominator and the denominator of the formula for the tangent t, i.e. they are different from c and s of the rotation). These factorized rotations result in a square root free or a square root and division free factorized rotation, if the matrix elements of the matrices $\mathbf{K}'_{pq}$

and $\mathbf{Z}^{(k+1)}$ can be computed without square roots in (2.12) and without square roots and divisions in (2.13).

Now, the formula for the tangent of the rotation angle $t = f(a_{ij}^{(k)}) = \frac{s_t(a_{ij}^{(k)})}{c_t(a_{ij}^{(k)})}$ must be adapted to the factorized representation by using $a_{ij}^{(k)} = \frac{y_{ij}^{(k)}}{\sqrt{z_i^{(k)} z_j^{(k)}}}$, which is the number description of the matrix elements $a_{ij}^{(k)}$ according to (2.10). This yields

$$t = f(y_{ij}^{(k)}, z_i^{(k)}, z_j^{(k)}) = \frac{s_t(y_{ij}^{(k)}, z_i^{(k)}, z_j^{(k)})}{c_t(y_{ij}^{(k)}, z_i^{(k)}, z_j^{(k)})}$$

If $s_t(y_{ij}^{(k)}, z_i^{(k)}, z_j^{(k)}) = s_t'(y_{ij}^{(k)}, z_i^{(k)}, z_j^{(k)}) \cdot \sqrt{z_i^{(k)} z_j^{(k)}}$ and the computation of $s_t'(y_{ij}^{(k)}, z_i^{(k)}, z_j^{(k)})$ and $c_t(y_{ij}^{(k)}, z_i^{(k)}, z_j^{(k)})$ only requires additions and multiplications the square root free rotation is defined by

$$(2.14)\ \mathbf{K}_{pq}' = \begin{bmatrix} \Leftrightarrow\frac{s_t' \cdot z_q^{(k)}}{c_t} & 1 \\ 1 & \frac{s_t' \cdot z_p^{(k)}}{c_t} \end{bmatrix}, \qquad \begin{aligned} z_p^{(k+1)} &= z_q^{(k)} \cdot \Delta \\ z_q^{(k+1)} &= z_p^{(k)} \cdot \Delta \end{aligned} \quad \text{with } \Delta = \Leftrightarrow\det\mathbf{K}_{pq}'$$

and the square root and division free rotation by

$$(2.15)\mathbf{K}_{pq}' = \begin{bmatrix} \Leftrightarrow s_t' \cdot z_q^{(k)} & c_t \\ c_t & s_t' \cdot z_p^{(k)} \end{bmatrix}, \qquad \begin{aligned} z_p^{(k+1)} &= z_q^{(k)} \cdot \Delta \\ z_q^{(k+1)} &= z_p^{(k)} \cdot \Delta \end{aligned} \quad \text{with } \Delta = \Leftrightarrow\det\mathbf{K}_{pq}'$$

This provides an easy scheme for deriving square root free or square root and division free rotations from the formula for the tangent of the rotation angle (e.g. for the Givens rotation, where $t = a_{pq}^{(k)}/a_{pp}^{(k)}$ one obtains with $a_{ij}^{(k)} = \frac{y_{ij}^{(k)}}{\sqrt{z_i^{(k)} z_j^{(k)}}}$ that $t = \frac{y_{pq}^{(k)} \sqrt{z_p^{(k)} z_q^{(k)}}}{y_{pp}^{(k)} z_q^{(k)}}$; i.e. $s_t' = y_{pq}^{(k)}$, $c_t = y_{pp}^{(k)} z_q^{(k)}$ for (2.14) and (2.15), respectively).

For all factorized schemes there is a problem with the growth of the matrix elements. To overcome this problem the scaling procedure of [14] can be applied to (2.14) and (2.15), respectively. This scaling procedure bounds the growth of the elements of the diagonal matrices $\mathbf{Z}^{(k)}$, such that $z_i^{(k)} \in [0.5, 2]$ for all k. It only requires two single shifts and four additions of exponents for scaling $\mathbf{K}_{pq}'$.

2.4. Factorized Approximate Rotations. Square root free or square root and division free JAs can be derived by combining the approximation and the factorization of the rotations.

For the KAs the formula for the tangents of the rotation angles can be described in an appropriate form only for KA2 and KA3. For KA1 a factorization without square roots is not possible, since

$$\begin{aligned} t_{KA1} &= \frac{s_{t_{KA1}}}{c_{t_{KA1}}} = \frac{\text{sign}\left(a_{pp}^{(k)} \Leftrightarrow a_{qq}^{(k)}\right) \cdot a_{pq}^{(k)}}{\left|a_{pq}^{(k)}\right| + \left|a_{pp}^{(k)} \Leftrightarrow a_{qq}^{(k)}\right|} = \\ &= \frac{\text{sign}\left(y_{pp}^{(k)} z_q^{(k)} \Leftrightarrow y_{qq}^{(k)} z_q^{(k)}\right) \cdot y_{pq}^{(k)} \sqrt{z_p^{(k)} z_q^{(k)}}}{\left|y_{pq}^{(k)}\right| \sqrt{z_p^{(k)} z_q^{(k)}} + \left|y_{pp}^{(k)} z_q^{(k)} \Leftrightarrow y_{qq}^{(k)} z_q^{(k)}\right|} = \frac{s_{t_{KA1}}' \sqrt{z_p^{(k)} z_q^{(k)}}}{c_{t_{KA1}}} \end{aligned}$$

Although $s_{t_{KA1}} = s'_{t_{KA1}}\sqrt{z_p^{(k)} z_q^{(k)}}$ holds, the computation of $c_{t_{KA1}}$ requires a square root operation. For KA2 and KA3 one obtains

$$\begin{aligned} t_{KA2} &= \frac{s_{t_{KA2}}}{c_{t_{KA2}}} = \frac{a_{pq}^{(k)}}{a_{pp}^{(k)} \Leftrightarrow a_{qq}^{(k)}} = \\ &= \frac{y_{pq}^{(k)}\sqrt{z_p^{(k)} z_q^{(k)}}}{y_{pp}^{(k)} z_q^{(k)} \Leftrightarrow y_{qq}^{(k)} z_q^{(k)}} = \frac{s'_{t_{KA2}}\sqrt{z_p^{(k)} z_q^{(k)}}}{c_{t_{KA2}}} \end{aligned}$$

$$\begin{aligned} t_{KA3} &= \frac{s_{t_{KA3}}}{c_{t_{KA3}}} = \frac{\left(a_{pp}^{(k)} \Leftrightarrow a_{qq}^{(k)}\right) \cdot a_{pq}^{(k)}}{\left(a_{pq}^{(k)}\right)^2 + \left(a_{pp}^{(k)} \Leftrightarrow a_{qq}^{(k)}\right)^2} = \\ &= \frac{\left(y_{pp}^{(k)} z_q^{(k)} \Leftrightarrow y_{qq}^{(k)} z_q^{(k)}\right) \cdot y_{pq}^{(k)}\sqrt{z_p^{(k)} z_q^{(k)}}}{\left(y_{pq}^{(k)}\right)^2 z_p^{(k)} z_q^{(k)} + \left(y_{pp}^{(k)} z_q^{(k)} \Leftrightarrow y_{qq}^{(k)} z_q^{(k)}\right)^2} = \frac{s'_{t_{KA3}}\sqrt{z_p^{(k)} z_q^{(k)}}}{c_{t_{KA3}}} \end{aligned}$$

such that square root free Jacobi rotations are obtained by (2.14) and square root and division free Jacobi rotations by (2.15).

For the NAs different formulae for the tangent are required in dependence of the value of $|\tau_J|$, whereby it must be possible to describe each formula in an appropriate form. The formula for $|\tau_J| < 1$ of NA1 enables no square root free factorization and therefore NA1 likewise not. All other NAs (NA2–5) are composed of KA2, KA3 and $t_1 = \text{sign}(\sigma_J)$. Therefore, the remaining problem, for deriving square root free or square root and division free Jacobi rotations for the NA2–5 is that $t_1 = \text{sign}(\sigma_J)$ cannot be factorized such that square roots are avoided.

In order to overcome this problem the approximation for the first cases of NA2–5, i.e. $t_1 = \text{sign}(\sigma_J)$, is replaced by the following approximation:

$$t_{f1} = \text{sign}(\sigma_J) \cdot \rho \cdot \sqrt{z_p^{(k)} z_q^{(k)}} \tag{2.16}$$

With $s'_{t_{f1}} = \rho \cdot \text{sign}(\sigma_J)$ and $c_{t_{f1}} = 1$ (2.14) and (2.15) yield the corresponding factorized rotations. Since the scaling procedure guarantees $z_i^{(k)} \in [0.5, 2]$ for all k, $|t_{f1}| \in [0.5, 2]$ holds. Since, this approximation t_{f1} is only used for $|\tau_J| < b$ (e.g. $b = 0.5$ for NA2), the following bounds for the maximal value of $|d_J|$ can be obtained:

$$|d_J\,(t_{f1}, \tau_J)| \leq 0.6 \qquad (\rho = 1) \tag{2.17}$$

$$|d_J\,(t_{f1}, \tau_J)| \leq 1/3 \text{ with } \begin{cases} \rho = 0.5 & \text{if } z_p^{(k)} z_q^{(k)} > 2 \\ \rho = \sqrt{2} & \text{if } z_p^{(k)} z_q^{(k)} < 0.5 \\ \rho = 1 & \text{otherwise} \end{cases} \tag{2.18}$$

(2.18) requires additional comparisons for determing ρ. However, these comparisons can be included, if they do not delay the data flow of the processor array (i.e. if the other formulae of the approximation are more complex, e.g. KA3). Furthermore, for NA4 and NA5 the difference between $|d_J|_{max} = 0.25$ for $|\tau_J| > b$ and $|d_J|_{max}$ for $|\tau_J| < b$ is reduced (1/3 instead of 0.6).

Although $|t_{NAi}| \rightarrow 1$ for $|\tau_J| \rightarrow 0$ no longer holds, the NA2–5 with t_{f1} instead of t_1 exhibit the same favourable properties as the NA2–5 of section 2.2, since $|d_J|_{max} \ll 1$ still holds and the approximation is still better than the KAs. The last two columns of Table 2.2 and Table 2.3 show the required number of sweeps of the square root and division free versions of NA4 and NA5. The required number of sweeps is about the same as for the non–factorized forms of NA4 and NA5.

Finally the comparison for determing the different cases, i.e. $1/2|\tau_J| = |\sigma_J| \geq b$, must also be referred to the factorized scheme:

$$|\sigma_J| \geq b \Leftrightarrow \left(y_{pq}^{(k)}\right)^2 z_p^{(k)} z_q^{(k)} \geq b^2 \left(y_{pp}^{(k)} z_q^{(k)} \Leftrightarrow y_{qq}^{(k)} z_p^{(k)}\right)^2 \tag{2.19}$$

Thus, the JAs based on the approximate rotation schemes, which are suitable for the factorized rotation schemes (KA2,KA3,NA2–5 with (2.17) or (2.18)), can be implemented requiring only $\{+, *, \div\}$ and $\{+, *\}$, respectively. At the end of the JA ($k = k_{end}$) the results must be refactorized. The eigenvalues are obtained by $\lambda_i = y_{ii}^{(k_{end})} / z_i^{(k_{end})}$, which requires n divisions. The eigenvectors require no refactorization. They are merely not normalized but the square of their lenght is contained in $\mathbf{Z}^{(k_{end})}$. Since the refactorization is only required at the end of the JA, the corresponding operations can be transferred to the host computer and must not be implemented on the multiprocessor array.

2.5. Comparison. Since the evaluation of the rotations is a more complex task than the pre- and postmultiplication of the rotations, which requires $(8\pm, 16*)$ for the two 2×2 matrix multiplications, the hardware requirements and the data pulse frequency of a regular (e.g.systolic) processor array are determined by the computational cost for the evaluation of the rotation.

In Table 2.4 the required operations for the evaluation of the different Jacobi rotations are shown. For the approximate rotations, which require different cases, the case with the greatest computational cost is indicated and the required operations for this case are specified. Since in general the operations $\{\div, \surd\}$ are (much) more expensive than the operations $\{\pm, *\}$, the number of the former operations is minimized for all rotations. Therefore, instead of computing $t = s_t/c_t$ and then (c, s) with t according to (2.6), the parameters (c, s) are computed with c_t and s_t as follows:

$$c = \frac{1}{\sqrt{c_t^2 + s_t^2}}, \qquad s = s_t \cdot c \tag{2.20}$$

(e.g. computing the exact rotation according to (2.5),(2.6) requires $(4\pm, 4*, 3\div, 2\surd)$ [25] while using (2.20) only requires $(4\pm, 5*, 2\div, 2\surd)$). The exponent operations for the scaling of the factorized schemes are not specified. We assume, that the processors contain a simple unit for the exponent manipulations (single shifts and additions), which can be executed in parallel to the other operations.

In a parallel implementation the number of sweeps N is predetermined [6, 3, 4]. If the evaluation of an approximate rotation requires t_a time, while the exact evaluation requires t_e time with $t_a = h \cdot t_e$ $(h < 1)$ and if only a few more sweeps are required for the approximate scheme, i.e. $N + N_1$ sweeps instead of N sweeps, then the time consumption of one sweep is reduced from $T_e = (n \Leftrightarrow 1)t_e$ to $T_a = (n \Leftrightarrow 1)t_a$ and the overall time consumption is $T_{AA} = (N + N_1)nt_a = (N + N_1)nht_e$ for the algorithm with approximate rotations instead of $T_{EA} = Nt_e$ for the algorithm with exact rotations. Obviously, $T_{AA} < T_{EA}$ holds, if $h(N + N_1) < N$ holds. For random matrices we can set $N_1 = 0$ for NA1, NA4 and NA5, such that $T_{AA} = hT_{EA}$ holds.

TABLE 2.4
Required operations of the different Jacobi rotations

rotation	required operations
exact	$4\pm$ $4*$ $2\div$ $2\surd$
KA1	$3\pm$ $3*$ $2\div$ $1\surd$
KA2	$2\pm$ $3*$ $2\div$ $1\surd$
KA3	$3\pm$ $4*$ $2\div$ $1\surd$
KA4	$5\pm$ $7*$ $2\div$ $1\surd$
KA5	$3\pm$ $3*$ $2\div$
NA1 (case 1)	$4\pm$ $6*$ $2\div$ $1\surd$
NA2 (case 2)	$2\pm$ $3*$ $2\div$ $1\surd$
NA3 (case 2)	$3\pm$ $4*$ $2\div$ $1\surd$
NA4 (case 2)	$2\pm$ $4*$ $2\div$ $1\surd$
NA5 (case 3)	$3\pm$ $4*$ $2\div$ $1\surd$
• $\surd$ free	
KA2,NA2	$2\pm$ $7*$ $1\div$
NA4	$2\pm$ $8*$ $1\div$
KA3,NA3,NA5	$3\pm$ $12*$ $1\div$
• $\surd$ and $\div$ free	
KA2,NA2	$2\pm$ $8*$
NA4	$2\pm$ $9*$
KA3,NA3,NA5	$3\pm$ $12*$

Even, if a few more sweeps are needed with the approximate scheme — e.g. in the case of clusters of eigenvalues ($N_1 = 4$ is the worst case in Table 2.2) or in the case of simpler approximations ($N_1 \leq 2$ for NA2) — we have $T_{AA} < T_{EA}$, if $h(N + N_1) < N$. For example, if we set $N = 10$ [3] and $N_1 = 2$ ($N_1 = 4$) $T_{AA} < T_{EA}$ holds if $h < 0.83$ ($h < 0.71$). Therefore, the choice of the computation scheme for the evaluation of the rotation strongly depends on the value of h for the particular computer and the particular implementation (see e.g. [25]; $h = 0.566$ for KA2 (NA2) on the DAP).

The square root and division free schemes only require $\{\pm, *\}$ to be implemented in all processor cells. They enable a very regular (all processors need the same hardware) parallel implementation. Therefore, these algorithms are particularly suited for ASIC–based processor arrays. Furthermore , the operations $\{\pm, *\}$ can be implemented with a time complexity of $O(\log_2 w)$ [31], while the operations $\{\div, \surd\}$ can only be implemented with a time complexity of $O(w)$, where w is the wordlenth of the data. Clearly, NA4 and NA5 are used for a square root and division free implementation, since they exhibit the highest accuracy of the approximation requiring the same amount of $\{\pm, *\}$ as the other worse approximations.

3. Triangular Kogbetliantz Algorithm.

3.1. Basic Algorithm. The TKA works by applying a sequence of two-sided orthogonal transformations to the upper triangular matrix $\mathbf{R}$ obtained by a preparatory QR-decomposition $\mathbf{A} = \mathbf{QR}$ of the arbitrary $m \times n$ matrix $\mathbf{A}$:

$\mathbf{A}^{(0)} := \mathbf{R}$

For $k = 0, 1, 2, \ldots$

$$\mathbf{A}^{(k+1)} = \mathbf{Q}_{pq}\mathbf{A}^{(k)}\mathbf{V}_{pq} \tag{3.1}$$

Except for the preparatory QR-decomposition, the only difference between the TKA and the JA is, that the orthogonal $n \times n$ rotation matrices multiplied to the left and right of $\mathbf{A}^{(k)}$ are defined by different rotation parameters. $\mathbf{Q}_{pq}$ is defined by $c_\Phi = \cos\Phi_k$, $s_\Phi = \sin\Phi_k$ and $\mathbf{V}_{pq}$ by $c_\Psi = \cos\Psi_k$, $s_\Psi = \sin\Psi_k$.

In order to preserve the upper triangular structure of $\mathbf{A}^{(k)}$ for all k

$$a_{qp}^{(k+1)} = \Leftrightarrow c_\Phi s_\Psi a_{pp}^{(k)} + c_\Phi c_\Psi a_{pq}^{(k)} + s_\Phi c_\Psi a_{qq}^{(k)} := 0 \tag{3.2}$$

must be met. For a maximal reduction of the off–diagonal quantity

$$a_{pq}^{(k+1)} = \Leftrightarrow s_\Phi c_\Psi a_{pp}^{(k)} \Leftrightarrow s_\Phi s_\Psi a_{pq}^{(k)} + c_\Phi s_\Psi a_{qq}^{(k)} = 0 \tag{3.3}$$

must be fullfilled. From (3.2) and (3.3) the following exact formulae for computing $\mathbf{Q}_{pq}$, $\mathbf{V}_{pq}$ are obtained:
Case 1 ($t_\Phi = \tan\Phi_k$ is computed first; $t_\Psi = \tan\Psi_k = f(t_\Phi)$)

$$\tau_{K1} = \frac{\left(a_{pp}^{(k)}\right)^2 \Leftrightarrow \left(a_{qq}^{(k)}\right)^2 + \left(a_{pq}^{(k)}\right)^2}{2a_{qq}^{(k)} a_{pq}^{(k)}} \tag{3.4}$$

$$t_\Phi = \frac{\operatorname{sign}(\tau_{K1})}{|\tau_{K1}| + \sqrt{1+\tau_{K1}^2}} \tag{3.5}$$

$$t_\Psi = \frac{a_{qq}^{(k)} t_\Phi + a_{pq}^{(k)}}{a_{pp}^{(k)}} \tag{3.6}$$

Case 2 (t_Ψ is computed first; $t_\Phi = f(t_\Psi)$)

$$\tau_{K2} = \frac{\left(a_{qq}^{(k)}\right)^2 \Leftrightarrow \left(a_{pp}^{(k)}\right)^2 + \left(a_{pq}^{(k)}\right)^2}{2a_{pp}^{(k)} a_{pq}^{(k)}} \tag{3.7}$$

$$t_\Psi = \frac{\Leftrightarrow\operatorname{sign}(\tau_{K2})}{|\tau_{K2}| + \sqrt{1+\tau_{K2}^2}} \tag{3.8}$$

$$t_\Phi = \frac{a_{pp}^{(k)} t_\Psi \Leftrightarrow a_{pq}^{(k)}}{a_{qq}^{(k)}} \tag{3.9}$$

In both cases (c_Φ, s_Φ) and (c_Ψ, s_Ψ) are obtained from t_Φ and t_Ψ according to (2.20). With these formulae $a_{ii}^{(k)} > 0$ $(i = 1, \ldots, n)$ holds for all k, if $a_{ii}^{(0)} > 0$ $(i = 1, \ldots, n)$ holds, which can always be obtained by the initial QR-decomposition [6].

3.2. Approximate Rotations. For a reduction of the off–diagonal quantity it is not necessary to satisfy $a_{pq}^{(k+1)} := 0$, but it is sufficient that

$$|a_{pq}^{(k+1)}| = |d_K||a_{pq}^{(k)}| \text{ with } 0 \le |d_K| < 1$$

holds. If a corresponding approximate formula is used for t_Φ (3.5) and t_Ψ (3.8), one has to distinguish between using case 1 if $a_{qq}^{(k)} \le a_{pp}^{(k)}$ and using case 2 if $a_{pp}^{(k)} < a_{qq}^{(k)}$ in order to guarantee $|d_K| < 1$ (for the exact scheme using one case is sufficient, but the test $a_{qq}^{(k)} \le a_{pp}^{(k)}$ is also necessary for a stable computation [20, 6]). Obviously, (3.5) is the same formula for computing t_Φ with τ_{K1} as (2.5) for computing t_{ex} with τ_J (the same holds for (3.8) and τ_{K2}). Therefore, all the approximations of Table 2.1 can also be applied to the TKA (i.e. to (3.5) and (3.8)). For (3.6) and (3.9) no further approximations are possible, since these formulae are obtained from (3.2), which must be met exactly in order to preserve the triangular structure.

We will not repeat the numerical examples of the JA for the TKA, since the following theorem shows: If the same approximation is used for the TKA and the JA,

the TKA yields a better approximation than the JA (compare Fig. 6.1 and Fig. 6.2 in [6]). In [6] the result of the following theorem is only shown for the special case $|\sigma| = 1/2|\tau| \to \infty$ in order to show that $|d_K(|\tau| \to \infty)| < 1$ is guaranteed for the TKA, while it is not for the JA (note, that the NAs guarantee $|d| \ll 1$ anyway).

THEOREM 3.1. *If the same approximation is used for the TKA (for t_Φ and $\Leftrightarrow t_\Psi$, respectively) and the JA (for t_{ex}), then for $|\tau_{Ki}| = |\tau_J|$ $(i = 1, 2)$*

$$|d_K(\tau_{Ki})| = |r| \cdot |d_J(\tau_J)|$$

holds with $|r| < 1$ and therefore

$$|d_K|_{max} < |d_J|_{max}$$

Proof. We assume that $|a_{pq}^{(k)}| > 0$, since otherwise no transformation (3.1) is executed.
Case 1 ($a_{qq}^{(k)} \leq a_{pp}^{(k)}$):
With $\text{sign}(\tau_{K1}) = \text{sign}(t_\Phi)$ and (3.6) one obtains from (3.3):

$$|d_K(\tau_{K1})| = \left| \frac{1 \Leftrightarrow 2|\tau_{K1}||t_\Phi| \Leftrightarrow t_\Phi^2}{1 + t_\Phi^2} \right| \cdot \left| \frac{a_{qq}^{(k)} \cdot c_\Psi}{a_{pp}^{(k)} \cdot c_\Phi} \right|$$

Since the same approximation is used for the TKA and the JA we have for each $|\tau_{K1}| = |\tau_J|$:

$$|d_K(\tau_{K1})| = |r| \cdot |d_J(\tau_J)|$$

with $|r| = \left| \frac{a_{qq}^{(k)} \cdot c_\Psi}{a_{pp}^{(k)} \cdot c_\Phi} \right|$. It remains to show that $|r| < 1$ holds, i.e. since all variables of $|r|$ are positive

$$\left(a_{qq}^{(k)}\right)^2 c_\Psi^2 < \left(a_{pp}^{(k)}\right)^2 c_\Phi^2$$

With $\cos^2\varphi = (1 + \tan^2\varphi)^{-1}$ we have

$$\left(a_{qq}^{(k)}\right)^2 (1 + t_\Phi^2) < \left(a_{pp}^{(k)}\right)^2 (1 + t_\Psi^2)$$

and with (3.6)

$$\left(a_{pp}^{(k)}\right)^2 \Leftrightarrow \left(a_{qq}^{(k)}\right)^2 + \left(a_{pq}^{(k)}\right)^2 + 2a_{qq}^{(k)} a_{pq}^{(k)} t_\Phi > 0$$

Since $a_{qq}^{(k)} > 0$ holds the multiplication of this inequality with $1/2a_{qq}^{(k)} a_{pq}^{(k)}$ yields

$$\begin{aligned} \tau_{K1} + t_\Phi > 0 \quad & \text{if } a_{pq}^{(k)} > 0 \\ \tau_{K1} + t_\Phi < 0 \quad & \text{if } a_{pq}^{(k)} < 0 \end{aligned}$$

Since $\text{sign}(\tau_{K1}) = \text{sign}(t_\Phi)$ these inequalities are equivalent to

$$\begin{aligned} \tau_{K1} > 0 \quad & \text{if } a_{pq}^{(k)} > 0 \\ \tau_{K1} < 0 \quad & \text{if } a_{pq}^{(k)} < 0 \end{aligned}$$

With (3.4) and $a_{qq}^{(k)} > 0$ this is again equivalent to

$$\left(a_{pp}^{(k)}\right)^2 \Leftrightarrow \left(a_{qq}^{(k)}\right)^2 + \left(a_{pq}^{(k)}\right)^2 > 0$$

which is guaranteed by $a_{pp}^{(k)} \geq a_{qq}^{(k)}$. This completes the proof for case 1.
Case 2 ($a_{pp}^{(k)} < a_{qq}^{(k)}$):

With (3.9) and $\text{sign}(\tau_{K2}) = \Leftrightarrow\text{sign}(t_\Psi)$ one obtains from (3.3):

$$|d_K(\tau_{K2})| = \left|\frac{1 \Leftrightarrow 2|\tau_{K2}||t_\Psi| \Leftrightarrow t_\Psi^2}{1+t_\Psi^2}\right| \cdot \left|\frac{a_{pp}^{(k)} \cdot c_\Phi}{a_{qq}^{(k)} \cdot c_\Psi}\right|$$

Therefore, $|d_K(\tau_{K2})| = |r| \cdot |d_J(\tau_J)|$ holds for each $|\tau_{K2}| = |\tau_J|$ with $|r| < 1$, if $\left(a_{pp}^{(k)}\right)^2 c_\Phi^2 < \left(a_{qq}^{(k)}\right)^2 c_\Psi^2$. Then, with (3.9), $a_{pp}^{(k)} > 0$, $\text{sign}(\tau_{K2}) = \Leftrightarrow\text{sign}(t_\Psi)$ and (3.7) the rest of the proof is identical to case 1. □

3.3. Factorized Approximate Rotations. For the TKA it is also possible to derive square root free and square root and division free rotations. The factorization

$$\mathbf{A}^{(k)} = \left[\mathbf{Z}^{(k)}\right]^{-1/2} \mathbf{Y}^{(k)} \left[\mathbf{X}^{(k)}\right]^{-1/2} \qquad \left(a_{ij}^{(k)} = \frac{y_{ij}^{(k)}}{\sqrt{z_i^{(k)} x_j^{(k)}}}\right) \tag{3.10}$$

must be used and it must be distinguished between approximating t_Φ (case 1) or t_Ψ (case 2). As for the exact formula the corresponding approximate formulae are only distinguished by the sign except for $t_{\Phi_1} = \text{sign}(\tau_{K1})$ and $t_{\Psi_1} = \Leftrightarrow\text{sign}(\tau_{K2})$ for which $t_{\Phi_{f1}} = \text{sign}(\tau_{K1})\rho\sqrt{z_p^{(k)} z_q^{(k)}}$ and $t_{\Psi_{f1}} = \text{sign}(\tau_{K2})\rho\sqrt{x_p^{(k)} x_q^{(k)}}$ must be used, respectively. The methods of section 2 can be applied to the TKA and the square root and division free rotations (square root free rotations in a similiar way by using (2.14) instead of (2.15) in the sequel) can be obtained as follows:
Case 1:
Using (3.10) and a suitable approximation t_{Φ_A} for the exact t_Φ (3.5), the approximate formula t_{Φ_A} can be written as:

$$t_{\Phi_A} = \frac{s_{\Phi_A}}{c_{\Phi_A}} = \frac{s'_{\Phi_A}\sqrt{z_p^{(k)} z_q^{(k)}}}{c_{\Phi_A}} \tag{3.11}$$

whereby the computation of $s'_{\Phi_A} = f(y_{ij}^{(k)}, z_i^{(k)}, x_j^{(k)})$ and $c_{\Phi_A} = f(y_{ij}^{(k)}, z_i^{(k)}, x_j^{(k)})$ $(i,j \in \{p,q\})$ requires only additions and multiplications. Then, (2.15) yields the rotation matrix

$$\tilde{\mathbf{Q}}'_{pq} = \begin{bmatrix} \Leftrightarrow s'_{\Phi_A} z_q^{(k)} & c_{\Phi_A} \\ c_{\Phi_A} & s'_{\Phi_A} z_p^{(k)} \end{bmatrix}, \qquad \begin{aligned} z_q^{(k+1)} &= z_p^{(k)} \cdot \Delta_\Phi \\ z_p^{(k+1)} &= z_q^{(k)} \cdot \Delta_\Phi \end{aligned} \quad \text{with } \Delta_\Phi = \Leftrightarrow\det\tilde{\mathbf{Q}}'_{pq}$$

With (3.10) and (3.11) one obtains for t_Ψ (3.6):

$$t_\Psi = \frac{s_\Psi}{c_\Psi} = \frac{a_{qq}^{(k)} t_{\Phi_A} + a_{pq}^{(k)}}{a_{pp}^{(k)}} = \frac{\left(y_{qq}^{(k)} z_p^{(k)} s'_{\Phi_A} + y_{pq}^{(k)} c_{\Phi_A}\right)\sqrt{x_p^{(k)} x_q^{(k)}}}{y_{pp}^{(k)} x_q^{(k)} c_{\Phi_A}} = \frac{s'_\Psi\sqrt{x_p^{(k)} x_q^{(k)}}}{c_\Psi}$$

and thereby

$$\tilde{\mathbf{V}}'_{pq} = \begin{bmatrix} \Leftrightarrow s'_\Psi x_q^{(k)} & c_\Psi \\ c_\Psi & s'_\Psi x_p^{(k)} \end{bmatrix}, \qquad \begin{aligned} x_q^{(k+1)} &= x_p^{(k)} \cdot \Delta_\Psi \\ x_p^{(k+1)} &= x_q^{(k)} \cdot \Delta_\Psi \end{aligned} \quad \text{with } \Delta_\Psi = \Leftrightarrow \det \tilde{\mathbf{V}}'_{pq}$$

Case 2:
The factorized rotations can be derived in a similiar way exchanging the role of t_Φ and t_Ψ.

3.4. Summary. In summary, it is clear why we started with the JA and extended the results to the TKA afterwards. All results for the JA (approximations, derivation of factorized rotations) can be extended to the TKA. Particularly, the measure for the accuracy of the approximations for the TKA, i.e. $|d_K|$, can be derived from the respective measure for the JA, i.e. $|d_J|$, where $|d_K| = |r| \cdot |d_J|$ holds with $|r| < 1$.

4. Convergence. The global convergence of the JA and the original KA has already been proved by Forsythe and Henrici [10] including approximate rotation schemes. The ultimate quadratic convergence of the JA has been proved by Wilkinson [32] and Paige and van Dooren [28] have extended this result to the original KA. However, for the original KA there are problems with the global and (in the case of clusters of eigenvalues) the ultimate quadratic convergence. These problems are completely removed by the TKA [20, 5]. Furthermore, the TKA is advantageous for a parallel (and a sequential) implementation. The global convergence of the TKA has been proved by Hari and Veselić [20] and by Fernando [8] and the ultimate quadratic convergence by Hari [21] and Charlier and van Dooren [5]. All these proofs of the TKA are established for the exact scheme. Although the convergence of the approximate schemes is used in [25, 6] (based on the results of [10]), no explicit proofs have been published yet (e.g. the proof of Fernando [8] cannot be extended to approximate schemes).

In [13] the global and the ultimate quadratic convergence of the TKA and the JA is proved for the approximate schemes. The known results for the exact schemes are obtained as special cases ($d = 0$). Furthermore, for the first time the presented proofs hold for the TKA as well as for the JA. This is achieved by assuming that during the TKA and the JA the matrices remain 'essential triangular' [21]. Although for the JA $\mathbf{A}^{(k)^T} = \mathbf{A}^{(k)}$ actually holds, we only use one essential triangular part, such that the JA also proceeds as shown in Fig. 2.1.1 of [20] for the TKA.

In the following the theorems concerning the global and the ultimate quadratic convergence are given and the methods used for the proofs of these theorems are briefly outlined. For the details of these proofs see [13].

THEOREM 4.1 (Global convergence). *If $0 \leq |d| < 1$ holds throughout the TKA and the JA, then the column (and row) cyclic schemes of these algorithms are always convergent.*

Proof. The proof of this theorem is modeled after the proof of the global convergence of the TKA with exact rotations [20]. We prove that $\left[S^{(k+M)}\right]^2 \leq c_n \left[S^{(k)}\right]^2$ ($M = n(n \Leftrightarrow 1)/2$) with $c_n < 1$ holds for all the different algorithms presented in this paper, i.e for the TKA/JA with exact/approximate rotations (see [13]). □

THEOREM 4.2 (Ultimate quadratic convergence). *Let*

$$|\sigma_i \Leftrightarrow \sigma_j| \geq 2\delta \qquad (i \neq j) \qquad \textit{(distinct singular-/eigenvalues)} \tag{4.1}$$

and suppose we have reached the stage r, where

$$S^{(r)} < \frac{\delta}{4} \tag{4.2}$$

then for some (k > r)

$$S^{(k+M)} \leq \frac{\gamma}{\sqrt{1 \Leftrightarrow d_{max}^2}} \cdot \frac{\left[S^{(k)}\right]^2}{\delta} + O\left(\left[S^{(k)}\right]^3\right) \tag{4.3}$$

holds where $\gamma = 7/6$ for the TKA and $\gamma = 1$ for the JA.

Proof. The proof of this theorem (see [13]) follows the proofs of Paige and van Dooren [28] (TKA) and Wilkinson [32] (JA) ($x_i := a_{pq}^{(i)}$) with $|x_{i+1}| = |d_i x_i|$ (instead of $x_{i+1} = 0$). □

In the case of clusters of singular values (eigenvalues) the proofs for the approximate schemes are completely identical to the proofs for the exact scheme of [22, 5, 21]. Only the bound (4.3) for the case of distinct singular/eigenvalues must be used instead of the bound for the exact scheme.

5. Application in Signal Processing. By computing the singular value decomposition (SVD) of a 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 (ESPRIT, MUSIC), state–space system identification. In practice, where the subspaces are usually time varying, it is even more important to be able to track these subspaces. Therefore, in recent years different subspace tracking algorithms have been proposed. These algorithms are based on the rank revealing **QR**–decomposition (QRD) [2], the Lanczos algorithm [7] or the SVD–updating algorithm [9, 26], where the SVD–updating algorithms are favourable in virtue of a parallel implementation. The SVD–updating algorithm works as follows: At time step k a new meassurement vector is incorporated in the upper triangular matrix $\mathbf{R}_{[k-1]}$ by a QRD–update resulting in $\mathbf{R}_{[k]}$. Then, the SVD of $\mathbf{R}_{[k]}$ is computed using the TKA.

The main result of [9, 26] is that it is usually sufficient to execute only one sweep [9], or even only a fraction of one sweep [26] (only the elements of the first subdiagonal of $\mathbf{R}_{[k]}$ are annihilated) of the TKA after each QRD-update. Since the approximation of the rotations is a marginal approximation compared to the approximation of the TKA (i.e. executing only one sweep or a fraction of one sweep), the use of approximate rotations yields essentially the same results with respect to tracking capability as using exact rotation.

To illustrate this the example of Ferzali and Proakis [9] is used, i.e. 400 sample points of a signal composed of 3 sinusoids:

$$s(k) = 2\cos(2\pi \cdot 0.15k) + 2\cos(2\pi \cdot 0.2k) + 2\cos(2\pi \cdot f \cdot k) + u(k)$$

whereby f jumps from 0.35 to 0.45 at $k = 200$ and $u(k)$ is Gaussian white noise (SNR = 20 dB). Data vectors with dimension $m = 7$ are formed from the samples. Fig. 5.1 shows $e(k) = \|\mathbf{sv}(k) \Leftrightarrow \mathbf{sve}(k)\|_2$ vs. k. $\mathbf{sve}(k)$ is a vector which contains the exact singular values of the $k \times m$ data matrix $\mathbf{X}(k)$ available at time k. $\mathbf{sv}(k)$ is the vector which contains the singular values as obtained by the SVD–updating algorithm. Here, only the first subdiagonal of $\mathbf{R}_{[k]}$ is annihilated by the TKA, i.e only a part of a complete sweep of the TKA is executed after each QRD–update. Obviously, tracking the singular values using the approximation KA2 (Fig. 5.1 b) works as well as using exact rotations (Fig. 5.1 a).

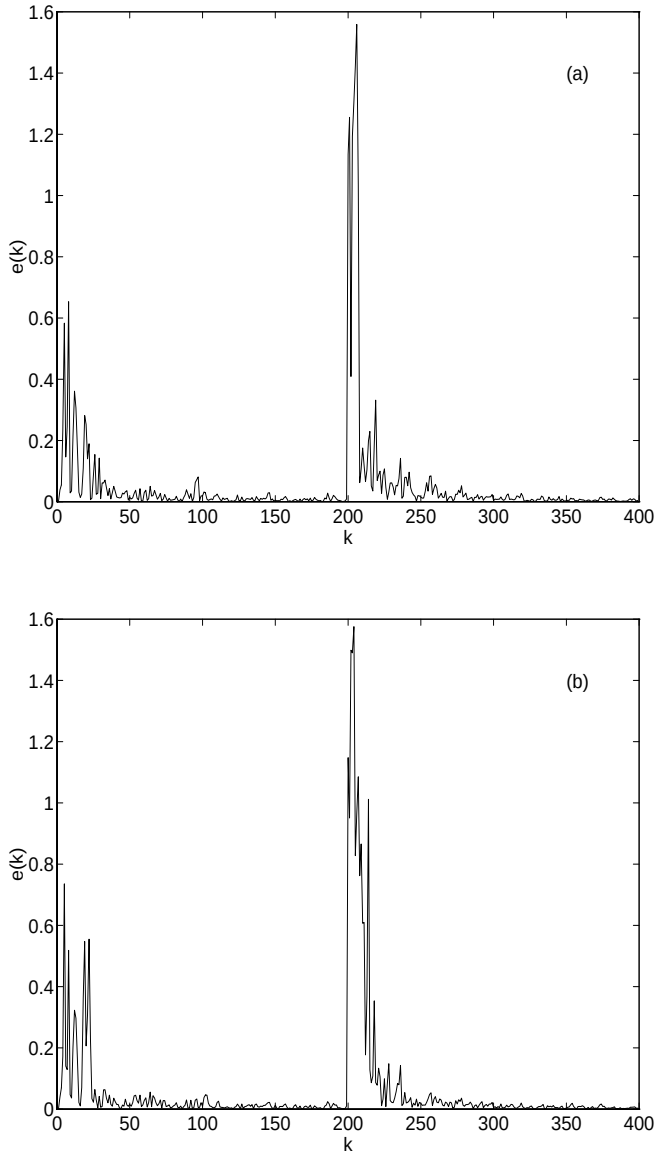

FIG. 5.1. *$e(k)$ vs k using (a) exact rotations and (b) the approximate rotation KA2.*

Therefore, an application specific processor array for subspace tracking can be derived, which exhibits all advantages of the approximate factorized schemes, while the slight increase concerning the required number of sweeps is marginal (in many practical applications even a fraction of a sweep is sufficient).

6. Concluding Remarks. In this paper new JAs and TKAs were presented by combining methods for modifying orthogonal rotations. Suitable approximate rotations were described in factorized form in order to gain square root free or square

root and division free rotations for the JA and the TKA.

In summary, we recommend the use of approximate schemes for the TKA as well as the JA, since

- the new approximations exhibit better properties and require less computational cost than the known approximations. Even for the JA the required number of sweeps of NA4 and NA5 is about the same as for the exact scheme. The TKA always performs better than the JA, since $|d_K|_{max} < |d_J|_{max}$ (Theorem 3.1)
- the convergence problems (for the JA) of KA2 and KA3 are removed by the new approximations ($|d| \ll 1$ is guaranteed).
- the use of approximate rotations enables the derivation of square root free or square root and division free factorized rotations, which is not possible for the exact scheme.
- for a parallel implementation the speedups gained by using approximate rotations and by using factorized rotations can be combined. Furthermore, the hardware requirements are significantly decreased (e.g. the square root and division free versions of the JA and the TKA can efficiently be implemented on an array of application specific processors, e.g. Transputers or DSPs, which contain only $\{\pm, *\}$ as fast hardware).
- for a VLSI-implementation of a systolic array executing the TKA and the JA only $\{\pm, *\}$ must be implemented in all processor cells (ASICs). In contrary to the operations $\{\div, \sqrt{}\}$, which require $O(w)$ time, the operations $\{\pm, *\}$ can be implemented needing $O(\log_2 w)$ time [31]. This enables a further speedup for the square root and division free algorithms. The approximate rotation scheme can also be combined efficiently with the CORDIC scheme in order to obtain an efficient VLSI-implementation based on the CORDIC algorithm [17, 18].
- the approximate factorized schemes are particularly useful in signal processing applications (e.g. subspace tracking), since the approximate rotations perform as well as exact rotations for many practical applications.

Acknowledgement. This work was performed while the author was with the Department of Computer Science, Yale University, New Haven funded by a grant of the German National Science Foundation. Thanks are to Prof. I. Ipsen for giving me the opportunity to spend a wonderful year at Yale. Thanks are also to Prof. A. Bojanczyk, who brought the use of approximate rotations [6, 25] to the attention of the author.

REFERENCES

[1] J. L. Barlow and I. C. F. Ipsen, *Scaled Givens rotations for the solution of linear least squares problems on systolic arrays*, SIAM J. Sci. Stat. Comput., 5 (1987), pp. 716–733.

[2] C. H. Bischof, *On updating signal subspaces*, IEEE Trans. on Signal Proc., 40 (1992), pp. 96–105.

[3] R. P. Brent and F. T. Luk, *The solution of singular value and symmetric eigenvalue problems on multiprocessor arrays*, SIAM J. Sci. Stat. Comput., 6 (1985), pp. 69–84.

[4] R. P. Brent, F. T. Luk and C. van Loan, *Computation of the singular value decomposition using mesh connected processors*, J. VLSI Computer Systems 3, (1985), pp. 242–270.

[5] J.-P. Charlier and P. van Dooren, *On Kogbetliantz's SVD algorithm in the presence of clusters*, Linear Algebra & Applic. 95, (1987), pp. 135–160.

[6] J.-P. Charlier, M. Vanbegin and P. van Dooren, *On efficient implementations of Kogbetliantz's algorithm for computing the singular value decomposition*, Numer. Math., 52

(1988), pp. 279–300.

[7] P. Comon and G. H. Golub, *Tracking a few singular values and vectors in signal processing*, Proc. IEEE, 78 (1990), pp. 1327–1343.

[8] K. V. Fernando, *Linear convergence of the row cyclic Jacobi and Kogbetliantz methods*, Numer. Math., 56 (1989), pp. 73–91.

[9] W. Ferzali and J. G. Proakis, *Adaptive SVD algorithm for covariance matrix eigenstructure computation*, IEEE Int. Conf. on Acoust., Speech & Signal Processing, Toronto 1990, pp. 2615–2618.

[10] G. E. Forsythe and P.Henrici, *The cyclic Jacobi method for computing the principal values of a complex matrix*, Trans. Amer. Math. Soc., 94 (1960), pp. 1–23.

[11] W.M. Gentleman, *Computation of Givens transformations without square roots*, J. Inst. Maths. Applies, 12 (1973), pp. 329–336.

[12] G. H. Golub and C. van Loan, *Matrix Computations*, Second ed., The John Hopkins University Press, Baltimore, MD, 1989.

[13] J. Götze, *On the parallel implementation of Jacobi's and Kogbetliantz's algorithm*, Research Report, Department of Computer Science, Yale University, New Haven, CT, 1991, (preprint).

[14] J. Götze and U. Schwiegelshohn, *A square root and division free Givens rotation for solving least squares problems on systolic arrays*, SIAM J. Sci. Stat. Comput., 4 (1991), pp. 800-807.

[15] ———, *VLSI-suited orthogonal solution of systems of linear equations*, J. Parallel & Distributed Comput., 11 (1991), pp. 276–283.

[16] J. Götze, M. Ali and U. Schwiegelshohn, *Efficient orthogonal matrix decompositions for digital signal processing*, Proc. URSI Int. Symp. on Signals, Systems and Electronics, Erlangen 1989, pp. 803–806.

[17] J. Götze, S. Paul, M. Sauer, *An efficient Jacobi–like algorithm for parallel eigenvalue computation*, IEEE Trans. on Computers, 42 (1993), pp. 1058–1065.

[18] ———, *A CORDIC–Based Jacobi–like Algorithm for Eigenvalue Computation*, Proc. IEEE Int. Conf. on Acoust., Speech, Signal Processing, Minneapolis 1993, Vol. 3, pp. 296–299.

[19] E.R. Hansen, *On cyclic Jacobi methods*, J. Soc. Indust. Appl. Math., 2 (1963), pp. 448–459.

[20] V. Hari and K. Veselić, *On Jacobi methods for singular value decomposition*, SIAM J. Sci. Stat. Comput., 5 (1987), pp. 741–754.

[21] V. Hari, *On the quadratic convergence of the serial singular value decomposition Jacobi methods for triangular matrices*, SIAM J. Sci. Stat. Comput., 6 (1989), pp. 1076–1096.

[22] H. P. M. van Kempen, *On the quadratic convergence of the special cyclic Jacobi method*, Numer. Math., 9 (1966), pp. 19–22.

[23] F. T. Luk and H. Park, *On parallel Jacobi orderings*, SIAM J. Sci. Stat. Comput., 1 (1989), pp. 18–26.

[24] F. T. Luk, *A triangular processor array for computing singular values*, Linear Algebra & Appl., 77 (1986), pp. 259–273.

[25] J. J. Modi and J. D. Pryce, *Efficient implementation of Jacobi's diagonalization method on the DAP*, Numer. Math., 46 (1985), pp. 443–454.

[26] M. Moonen, P. van Dooren and J. Vandewalle, *A singular value decomposition updating algorithm for subspace tracking*, SIAM J. Matrix Anal. Appl., 4 (1992), pp. 1015–1038.

[27] W. Rath, *Fast Givens rotations for orthogonal similarity transformations*, Numer. Math., 40 (1982), pp. 47–56.

[28] C. C. Paige and P. van Dooren, *On the quadratic convergence of Kogbetliantz's algorithm for computing the singular value decomposition*, Linear Algebra & Appl., 77(1986), pp. 301–313.

[29] D. A. Pope and C. Tompkins, *Maximizing functions of rotations – Experiments concerning the speed of diagonalization of symmetric matrices using Jacobi's method*, J. ACM, 4 (1957), pp. 459–466.

[30] U. Schwiegelshohn and L. Thiele, *A systolic array for cyclic–by–rows Jacobi algorithms*, J. Parallel Distrib. Comput., 4 (1987), pp. 334–340.

[31] J. Vuillemin, *A very fast multiplication algorithm for VLSI implementation*, Integration: the VLSI J., 1 (1983), pp. 39–52.

[32] J. H. Wilkinson, *Note on the quadratic convergence of the cyclic Jacobi process*, Numer. Math., 6 (1962), pp. 296–300.

[33] J. H. Wilkinson, *The Algebraic Eigenvalue Problem*, Oxford University Press, London, 1965.