Error Analysis of CORDIC–Based Jacobi Algorithms

*Steffen Paul, Jürgen Götze, and Matthias Sauer**

Abstract

The Jacobi algorithm for eigenvalue calculation of symmetric matrices can be performed with a CORDIC algorithm as its basic module. Recently, a simplified Jacobi algorithm, by employing approximate rotations based on CORDIC rotations, was proposed. It fully exploits the binary data structure and reduces the overall computational cost significantly.

In this paper an error analysis of the approximate CORDIC–based Jacobi algorithm and the conventional CORDIC–based Jacobi algorithm is performed. The new algorithm behaves numerically better than the conventional CORDIC–based Jacobi algorithm for fixed as well as floating point arithmetic.

Index Terms: Error analysis, eigenvalue calculation, Jacobi algorithms, CORDIC, approximate rotations.

1 Introduction

The Jacobi algorithm (JA) is well suited for eigenvalue calculation on multiprocessor arrays [4], [1]. Apart from the conventional way of calculating the rotation matrices by divisions, square roots and transcendental functions [14] the CORDIC–based Jacobi algorithm (CJA) calculates and performs the rotations by the CORDIC algorithm in order to simplify the VLSI realization [12]. The CORDIC algorithm composes a rotation by an angle Φ of a sequence of elementary rotations by $\arctan 2^{-i}$ and requires only shift and addition operations.

It was shown in [8] how the combination of approximate Jacobi rotations with the CORDIC algorithm provides a simple and very fast algorithm for the symmetric eigenvalue problem. The resulting approximate CJA (ACJA) reduces the overall computational cost significantly. As was already mentioned by Delosme [3], this reduction might be accompanied by smaller numerical errors due to rounding. Apart from the speed of this new algorithm it is also appropriate for floating point arithmetic. The floating point CORDIC architecture of [9] can efficiently be adjusted to the requirements of the ACJA [7].

*Institute for Network Theory and Circuit Design, Technical University Munich, Arcisstr. 21, D–80290 Munich, e-mail: stpa@nws.e-technik.tu-muenchen.de

In this paper, an error analysis is performed for the CJA and the ACJA for fixed and floating point arithmetic, respectively. In Sect. 2 the general structure of a CORDIC–based Jacobi algorithm is introduced briefly. Since we are mainly interested in the error analysis of our new algorithm [8], we choose a common CJA for comparison and do not cover all implementation details. It is easy, however, to execute the error analysis for other implementations (double rotation method, different scaling procedures, etc.) following the lines of the presented error analysis. The errors due to the different steps in the algorithms are discussed in Sect. 3. The error analysis is kept close to that of Wilkinson [14]. This is convenient since the CORDIC–based algorithms differ from the conventional Jacobi algorithm only in the error bounds for the rotation and the scaling operation. The global structure of the algorithm is not affected.

2 Jacobi Algorithm

Jacobi's algorithm computes the eigenvalue decomposition of a symmetric $n \times n$ matrix $\boldsymbol{A} = \boldsymbol{Q}^T \boldsymbol{\Lambda} \boldsymbol{Q}$ by applying a sequence of two–sided orthogonal rotations to the matrix $\boldsymbol{A}$:

Algorithm JA ($\boldsymbol{A}^{(0)} = \boldsymbol{A}$)

$$
\begin{aligned}
&\textit{while} \;\; S^{(k)} > \epsilon \\
&\qquad \textit{for} \;\; \text{all index pairs} \;\; \{p,q\} \\
&\qquad\qquad \boldsymbol{A}^{(k+1)} = \boldsymbol{J}_{pq}^{T(k)} \boldsymbol{A}^{(k)} \boldsymbol{J}_{pq}^{(k)} \,,
\end{aligned} \tag{1}
$$

where the off–diagonal norm $S^{(k)}$ of $\boldsymbol{A}^{(k)}$ is defined by

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

The multiplication by $\boldsymbol{J}_{pq}^{(k)}$ performs a 2×2 vector rotation in the (p,q) plane. We assume that the index pairs $\{p,q\}$ are choosen cyclic–by–row. The rotations $\boldsymbol{J}_{pq}^{(k)}$ must be computed such that the off diagonal norm $S^{(k)}$ is reduced by each similarity transformation (1). This implies that $\boldsymbol{A}^{(k)}$ converges to the diagonal matrix $\boldsymbol{\Lambda}$ containing the eigenvalues of $\boldsymbol{A}$. Since $\boldsymbol{J}_{pq}^{(k)}$ is an orthogonal transformation, i.e. $\left\|\boldsymbol{A}^{(k+1)}\right\|_F = \left\|\boldsymbol{A}^{(k)}\right\|_F$, and one similarity transformation (1) affects only rows and columns p and q of $\boldsymbol{A}^{(k)}$,

$$
\left[S^{(k+1)}\right]^2 = \left[S^{(k)}\right]^2 - \left[\left(a_{pq}^{(k)}\right)^2 - \left(a_{pq}^{(k+1)}\right)^2\right] \tag{3}
$$

holds [4]. Obviously, the maximal reduction of $S^{(k)}$ is obtained if $a_{pq}^{(k+1)} = 0$.

As shown in [8], the CORDIC algorithm can efficiently be combined with approximate rotations, i.e. only $|a_{pq}^{(k+1)}| < |a_{pq}^{(k)}|$ is met (instead of $a_{pq}^{(k+1)} = 0$) which still guarantees a reduction of the off–diagonal quantity. Approximate rotation schemes are mainly based on the fact that the annihilated off–diagonal entries ($a_{pq}^{(k+1)} = 0$) are destroyed in later steps of the Jacobi algorithm such that their generation is not

strictly justified [6]. Although the Jacobi algorithm based on this approximate CORDIC–like rotation scheme lacks the ultimate quadratic convergence of the Jacobi algorithm based on the conventional CORDIC, the number of required shift–and–add operations is significantly reduced [8]. The quadratic convergence is retained, however, by increasing the number of CORDIC angles per rotation in a specific plane [8]. The increase of the number of CORDIC angles (this number is denoted by r in [8]) can be controlled by monitoring the stage of diagonalization [5].

The coarsest approximation based on the CORDIC idea is applying only one CORDIC angle, i.e. a rotation by an angle $\Phi_\ell = \arctan 2^{-\ell}$. This CORDIC angle, i.e. the value of ℓ where $\ell \in \Im = \{0, 1, \ldots, b\}$ (b is the word length), is determined such that $||\Phi_\ell| - |\Phi|| = \min_{i \in \Im} ||\Phi_i| - |\Phi||$.

For the CORDIC–based rotations the Jacobi algorithm consists of three steps to be considered (i) compute rotation angle such that $|a_{pq}^{(k+1)}| < |a_{pq}^{(k)}|$, (ii) apply two–sided rotation, (iii) scaling. This overall structure holds for the exact CJA and the ACJA and is used as guideline for the following error analysis.

3 Error Analysis

The error analysis of the CORDIC–based Jacobi algorithms is performed for floating point and fixed point arithmetic separately. To take advantage of the results of Wilkinson we analyze the algorithm in a similar step by step manner. The steps of analysis are: *evaluate angle, apply angle, scaling, premultiplication= apply angle + scaling, overall error for one sweep and for the whole algorithm.* Our analysis is distinct from that of [14] in determination of the error bounds for premultiplication. For the bounds of the overall error we make use of Wilkinson's derivation and substitute only the bound for premultiplication.

For the conventional CORDIC algorithm an error analysis was performed in [13], [11] as well as [10]. The latter bounds are applied to the Jacobi algorithm.

Because we are mostly interested in the ACJA, some simplifying assumptions for the CJA are made. To be as general as possible, some implementation details are neglected. Their contribution to the overall error is assumed to be the least of all possible implementations of a specific operation.

3.1 Error Models

The standard rounding error models for fixed and floating point arithmetic are used in the sequel [14].

3.1.1 Floating Point Arithmetic

Each floating point number is represented by $x = a2^e$ with a range for the mantissa of $\frac{1}{2} \leq |a| < 1$. The mantissa a possesses t digits and the machine accuracy is given by $\varepsilon_{Fl} = 2^{-t-1}$. Then for the basic arithmetic operations holds $z = Fl\,(x \text{ op } y) = x \text{ op } y(1 + \varepsilon_1)$ with op $\in \{+, -, *, /\}$ and $|\varepsilon_1| \leq 2^{-t}$.

3.1.2 Fixed Point Arithmetic

For fixed point arithmetic all numbers have to be restricted to $-1 \leq x \leq 1$. Again, the number possesses t digits and the machine accuracy is given by $\varepsilon_{Fi} = 2^{-t-1}$. If the result of the computation lies within the range $z = Fi(x \pm y) = x \pm y$ holds. Multiplications and divisions are calculated with a rounding error according to $z = Fi(x \text{ op } y) = x \text{ op } y + \varepsilon_2$ with op $\in \{*, /\}$ and $|\varepsilon_2| \leq 2^{-t-1}$. The restrictions concerning the range of the numbers is a serious limitation for large matrices because they have to be scaled in the course of the algorithm.

3.2 CORDIC–based Algorithm

The vector rotation can be considered the basic operation in the CJA. It is worthwhile to investigate the quantization error of this rotation. It is governed by:

$$\begin{bmatrix} \hat{a}_{pl} \\ \hat{a}_{ql} \end{bmatrix} = \prod_{i=1}^{b-1} \begin{bmatrix} 1 & \sigma_i 2^{-i} \\ -\sigma_i 2^{-i} & 1 \end{bmatrix} \begin{bmatrix} a_{pl} \\ a_{ql} \end{bmatrix} + \sum_{i=0}^{b-1} \prod_{j=i+1}^{b-1} \begin{bmatrix} 1 & \sigma_j 2^{-j} \\ -\sigma_j 2^{-j} & 1 \end{bmatrix} \begin{bmatrix} e_1(i) \\ e_2(i) \end{bmatrix} + \begin{bmatrix} e_1(b) \\ e_2(b) \end{bmatrix} \quad (4)$$

as derived in [10], Lemma 3. $e_1(i), e_2(i)$ denote the absolute rounding errors in each step. We have assumed error free input data ($e_1(0) = 0, e_2(0) = 0$) so that the sum index starts at 1 as opposed to [10]. The sum of the error products is due to the propagation of errors in previous matrix–vector multiplications. Worst case bounds depend on the type of arithmetic and will be given below.

Furthermore, we make use of the estimate for

$$(1 - 2^{-t})^b \leq (1 + \varepsilon) \leq (1 + 2^{-t})^b \quad (5)$$

to be $|\varepsilon| \leq b2^{-t_1}$ with $t_1 = t - \log_2 1.06$ valid for $b2^{-t} < 0.1$ [14].

Evaluate Angle: The angle evaluation gives rise to a sequence of rotation directions $\sigma_i \in \{-1, +1\}$. We assume error free calculation of the rotation angles. The finite angle resolution (finite word length) prevents a correct annihilation of off–diagonal matrix entries $a_{pq}^{(k+1)}$. Strictly, every Jacobi algorithm performed with CORDIC can be considered an approximate Jacobi algorithm.

3.2.1 Floating Point Arithmetic

Apply Angle: The worst case bound of (4) is given by Theorem 4 of [10] as

$$\left\| \begin{bmatrix} \hat{a}_{pl} \\ \hat{a}_{ql} \end{bmatrix} - \boldsymbol{T} \begin{bmatrix} a_{pl} \\ a_{ql} \end{bmatrix} \right\|_2 \leq \varepsilon_3 \left\| \begin{bmatrix} a_{pl} \\ a_{ql} \end{bmatrix} \right\|_2, \quad \text{where } \varepsilon_3 = b\varepsilon_{Fl} \quad (6)$$

with rotation matrix

$$\boldsymbol{T} = \prod_{i=0}^{b-1} \begin{bmatrix} 1 & \sigma_i 2^{-i} \\ -\sigma_i 2^{-i} & 1 \end{bmatrix}. \quad (7)$$

Scaling: Numerous strategies for scaling were proposed to force K to a simple binary representation. We assume the rounding error of K to be $K(1+\varepsilon_{Fl})$ as for a multiplication in the worst case. Hence, scaling introduces the error:

$$\begin{bmatrix} \tilde{a}_{pl} \\ \tilde{a}_{ql} \end{bmatrix} = K \begin{bmatrix} \hat{a}_{pl} \\ \hat{a}_{ql} \end{bmatrix} (1+\varepsilon_1) \tag{8}$$

supposed the rotated vector $[\hat{a}_{pl}, \hat{a}_{ql}]^T$ would be correct ($\varepsilon_1 = 2\varepsilon_{Fl}$) [10]. The scaling factor is independent of the actual rotation angle and is constant for the whole algorithm.

Premultiplication: The overall error for rotation (6) and scaling (8) amounts to

$$\begin{bmatrix} \tilde{a}_{pl} \\ \tilde{a}_{ql} \end{bmatrix} = (1+\varepsilon_4) K \boldsymbol{T} \begin{bmatrix} a_{pl} \\ a_{ql} \end{bmatrix}. \tag{9}$$

For ε_4 holds under the condition $t \geq 4$

$$\varepsilon_4 \leq \varepsilon_3 + \varepsilon_{Fl} + \varepsilon_3\varepsilon_{Fl} = (1 + b(1+\varepsilon_{Fl}))\varepsilon_{Fl} \leq (1+1.06b)\,\varepsilon_{Fl}\,. \tag{10}$$

Clearly, the length of the rotation loop b has the main impact on the error. Since the rotation length b and word length t are related by $t = b + \alpha$, $\alpha \in \{0, 1 \ldots\}$ an optimal number of digits for a required accuracy can be determined. In Table 1 the internal word length t for a required number of correct digits t_r

$$(1+1.06b)\,2^{-t} \leq 2^{-t_r} \tag{11}$$

is listed ($t = b$). The numbers are in agreement with results published in [2]. It turns out that a variation of α does not influence t_r significantly.

Overall error: If each matrix entry a_{ij} $(i \neq j)$ is rotated towards zero once, then one sweep is completed ($\boldsymbol{A}_S$). Using the bound for ε_4 of (10) and formula (27.2), Chapter 3 of [14] one obtains the bound for the Frobenius norm after one sweep

$$\|\boldsymbol{A}_S - \boldsymbol{T}_S \boldsymbol{A} \boldsymbol{T}_S^T\|_E \leq 2\varepsilon_4 n^{\frac{3}{2}} (1+\varepsilon_4)^{(4n-7)} \|\boldsymbol{A}\|_E\,, \tag{12}$$

where $\boldsymbol{T}_S$ denotes the product of the correct transformations of one sweep. For p sweeps one obtains a bound of

$$\|\boldsymbol{\Lambda} - \boldsymbol{Q}\boldsymbol{A}\boldsymbol{Q}^T\|_E \leq p C_{CFl} \|\boldsymbol{A}\|_E\,, \tag{13}$$

where

$$C_{CFl} = 2\varepsilon_4 n^{\frac{3}{2}} (1+\varepsilon_4)^{p(4n-7)}\,. \tag{14}$$

The factor C_{CFl} is shown in Table 2 for some typical parameters. Of course, the values are of interest only for a word length $b > 8$.

3.2.2 Fixed Point Arithmetic

Apply angle: Since the rotation angle is applied recursively, an absolute error due to the shift is multiply added with subsequent transformations. As explained in [10], Theorem 5 we obtain a bound

$$\left\| \begin{bmatrix} \hat{a}_{pl} \\ \hat{a}_{ql} \end{bmatrix} - \boldsymbol{T} \begin{bmatrix} a_{pl} \\ a_{ql} \end{bmatrix} \right\|_2 \leq \varepsilon_5 = \sqrt{2}\varepsilon_{Fi} G(b)\,, \tag{15}$$

where

$$G(b) = \left(1 + \sum_{i=1}^{b-1} \prod_{j=i}^{b-1} \left\| \begin{bmatrix} 1 & \sigma_j 2^{-j} \\ -\sigma_j 2^{-j} & 1 \end{bmatrix} \right\|_2 \right) = 1 + \sum_{i=1}^{b-1} \prod_{j=i}^{b-1} \sqrt{1 + 2^{-2j}}\,. \tag{16}$$

The factor $\prod_{j=i}^{b-1} \sqrt{1+2^{-2j}}$ tends to unity for increasing j. Thus, $G(b-1)$ is proportional to b for large b, as was already mentioned in [10], Fig. 2.

Scaling: Scaling introduces an additive error

$$\begin{bmatrix} \tilde{a}_{pl} \\ \tilde{a}_{ql} \end{bmatrix} = K \begin{bmatrix} \hat{a}_{pl} \\ \hat{a}_{ql} \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} \varepsilon_{Fi}. \tag{17}$$

Premultiplication: Substituting the intermediate results yields:

$$\left\| \begin{bmatrix} \tilde{a}_{pl} \\ \tilde{a}_{ql} \end{bmatrix} - K\boldsymbol{T} \begin{bmatrix} a_{pl} \\ a_{ql} \end{bmatrix} \right\|_2 \leq C_{CFi}\varepsilon_{Fi}\,, \tag{18}$$

where

$$C_{CFi} = \sqrt{2}(KG(b) + 1)\,. \tag{19}$$

Overall error: The 2–norm of the absolute error of one sweep, similar to (35.18), Chapter 3 of [14], is given by

$$\|\boldsymbol{A}_S - \boldsymbol{T}_S \boldsymbol{A}_0 \boldsymbol{T}_S^T\|_2 \leq \frac{1}{\sqrt{2}} C_{CFi}\varepsilon_{Fi}(2n-4)^{\frac{1}{2}} \frac{n(n-1)}{2} \tag{20}$$

and for p sweeps holds (some values of C_{CFi} are shown in Table 3):

$$\|\boldsymbol{\Lambda} - \boldsymbol{Q}\boldsymbol{A}\boldsymbol{Q}^T\|_2 \leq pC_{CFi}\varepsilon_{Fi}(2n-4)^{\frac{1}{2}} \frac{n(n-1)}{2}\,. \tag{21}$$

3.3 Approximate CORDIC–based Jacobi Algorithm

The error analysis of the ACJA is similar to the CJA. The main differences are the simplification that just one rotation angle is applied twice so that (4) turns into

$$\begin{bmatrix} \hat{a}_{pl} \\ \hat{a}_{ql} \end{bmatrix} = \prod_{i=1}^{2} \begin{bmatrix} 1 & \sigma_\ell 2^{-\ell} \\ -\sigma_\ell 2^{-\ell} & 1 \end{bmatrix} \begin{bmatrix} a_{pl} \\ a_{ql} \end{bmatrix} + \begin{bmatrix} 1 & \sigma_\ell 2^{-\ell} \\ -\sigma_\ell 2^{-\ell} & 1 \end{bmatrix} \begin{bmatrix} e_1(1) \\ e_2(1) \end{bmatrix} + \begin{bmatrix} e_1(2) \\ e_2(2) \end{bmatrix}. \tag{22}$$

and that the scaling factor is no longer constant but depends on the actual rotation angle Φ_ℓ. Hence, it has to be calculated anew for each rotation.

3.3.1 Floating Point Arithmetic

Apply Angle: The angle application consists of two elementary rotations only (compare with [10], proof of Theorem 4)

$$\left\| \begin{bmatrix} \hat{a}_{pl} \\ \hat{a}_{ql} \end{bmatrix} - \boldsymbol{T} \begin{bmatrix} a_{pl} \\ a_{ql} \end{bmatrix} \right\|_2 \leq \left(1 + \sqrt{1 + 2^{-2\ell}}\right) \varepsilon_{Fl} \left\| \begin{bmatrix} a_{pl} \\ a_{ql} \end{bmatrix} \right\|_2 \tag{23}$$

with rotation matrix

$$\boldsymbol{T} = \prod_{i=1}^{2} \begin{bmatrix} 1 & \sigma_\ell 2^{-\ell} \\ -\sigma_\ell 2^{-\ell} & 1 \end{bmatrix}. \tag{24}$$

The error bound of the rotation in the ACJA does no longer depend on the length of the iteration loop. In the worst case $\ell = 1$ one obtains a factor of only $\left(1 + 2^{-2}\right) = 1.25$ instead of b.

Scaling: Scaling for the approximate rotations by K_m is done by a sequence of shifts and adds, since the scaling factor depends on the rotation angle and is not fixed as in the full CORDIC algorithm. The length of the scaling loop depends on the rotation angle, thus on ℓ. The maximum loop length m arises for $\ell = 1$

$$m = \lceil \log_2 (b/2) \rceil. \tag{25}$$

The entire scaling procedure yields:

$$\begin{bmatrix} \tilde{a}_{pl} \\ \tilde{a}_{ql} \end{bmatrix} = (1 + \varepsilon_5) K_m \begin{bmatrix} \hat{a}_{pl} \\ \hat{a}_{ql} \end{bmatrix} \tag{26}$$

with K_m from (24) of [8] and

$$\left(1 - 2^{-t}\right)^m \leq (1 + \varepsilon_5) \leq \left(1 + 2^{-t}\right)^m \tag{27}$$

which can be simplified to

$$\varepsilon_5 \leq 1.06\, m\, 2^{-t} = 1.06\, m\, 2\, \varepsilon_{Fl}. \tag{28}$$

Obviously, in the ACJA scaling introduces a larger error than in the exact CJA. During the course of the Jacobi algorithm as the matrix gets more diagonaldominant, i.e., the angles become smaller, the scaling error of the ACJA decreases since the shift value ℓ increases. Therefore, m is actually given by $m = \lceil \log_2 \ell/2 \rceil$ but we assume the worst case, i.e., m of (25) in the sequel.

Premultiplication: The error for rotation and scaling is given by

$$\left\| \begin{bmatrix} \tilde{a}_{pl} \\ \tilde{a}_{ql} \end{bmatrix} - K_m \boldsymbol{T} \begin{bmatrix} a_{pl} \\ a_{ql} \end{bmatrix} \right\|_2 \leq \varepsilon_6 \left\| \begin{bmatrix} a_{pl} \\ a_{ql} \end{bmatrix} \right\|_2, \tag{29}$$

where

$$\varepsilon_6 = \left(1 + \sqrt{1 + 2^{-2\ell}} + 2m1.06 \left(1 + \sqrt{1 + 2^{-2\ell}}\right)\right) \varepsilon_{Fl}\,. \tag{30}$$

The internal word length for a required accuracy is shown in Table 1 for some typical values. Note the reduced internal word length of the ACJA in comparison to the CJA.

Overall Error: The error bounds are given as in (12) and (13), (14) using ε_6 insted of ε_4, i.e., we must set C_{AFl} for C_{CFl} in (13), where

$$C_{AFl} = 2\varepsilon_6 n^{\frac{3}{2}} (1+\varepsilon_6)^{p(4n-7)} \ . \tag{31}$$

In table 2 factors C_{AFl} (ACJA) are shown for some typical parameters. Comparing these values to C_{CFl} (CJA) one observes an improvement of the error bounds for the ACJA especially for long word lengths.

3.3.2 Fixed Point Arithmetic

Apply Angle: The rotation by one CORDIC angle introduces an additive error

$$\left\| \begin{bmatrix} \hat{a}_{pl} \\ \hat{a}_{ql} \end{bmatrix} - \boldsymbol{T} \begin{bmatrix} a_{pl} \\ a_{ql} \end{bmatrix} \right\|_2 \leq \varepsilon_7 = \sqrt{2}\varepsilon_{Fi} G'(m) \, , \tag{32}$$

where (from (22)) with the same reasoning as in (16)

$$G'(m) = 1 + \sqrt{1+2^{-2\ell}}. \tag{33}$$

Scaling: The scaling is done in the same way as for floating point arithmetic, again with a loop length of $m = \lceil \log_2 (b/2) \rceil$ for the worst case $\ell = 1$. Therefore, the scaled vector yields ($s_j = -1$ for $j = 1$ and $s_j = 1$ otherwise)

$$\begin{bmatrix} \tilde{a}_{pl} \\ \tilde{a}_{ql} \end{bmatrix} = K_m \begin{bmatrix} \hat{a}_{pl} \\ \hat{a}_{ql} \end{bmatrix} + \sum_{i=1}^{m} \left(\prod_{j=i}^{m} \left(1 + s_j 2^{-2^j} \right) \right) \begin{bmatrix} 1 \\ 1 \end{bmatrix} \varepsilon_{Fi}. \tag{34}$$

Premultiplication: Summarizing the last two steps yields

$$\begin{aligned} \begin{bmatrix} \tilde{a}_{pl} \\ \tilde{a}_{ql} \end{bmatrix} &= K_m \boldsymbol{T} \begin{bmatrix} a_{pl} \\ a_{ql} \end{bmatrix} + \left(K_m \begin{bmatrix} 1 & -\sigma 2^{-\ell} \\ \sigma 2^{-\ell} & 1 \end{bmatrix} + K_m + \sum_{i=1}^{m} \left(\prod_{j=i}^{m} 1 + s_j 2^{-2^j} \right) \right) \begin{bmatrix} 1 \\ 1 \end{bmatrix} \varepsilon_{Fi} \\ &= K_m \boldsymbol{T} \begin{bmatrix} a_{pl} \\ a_{ql} \end{bmatrix} + \left(K_m \left(2 + 2^{-\ell} \right) + \sum_{i=1}^{m} \left(\prod_{j=i}^{m} 1 + s_j 2^{-2^j} \right) \right) \begin{bmatrix} 1 \\ 1 \end{bmatrix} \varepsilon_{Fi} \, . \end{aligned} \tag{35}$$

Thus one obtains

$$\left\| \begin{bmatrix} \tilde{a}_{pl} \\ \tilde{a}_{ql} \end{bmatrix} - K_m \boldsymbol{T} \begin{bmatrix} a_{pl} \\ a_{ql} \end{bmatrix} \right\|_2 \leq C_{AFi} \varepsilon_{Fi} \, , \tag{36}$$

where

$$C_{AFi} = \sqrt{2} \left[K_m \left(2 + 2^{-\ell} \right) + \sum_{i=1}^{m} \left(\prod_{j=i}^{m} 1 + s_j 2^{-2^j} \right) \right] \ . \tag{37}$$

The term $K_m \begin{bmatrix} 1 & -\sigma 2^{-\ell} \\ \sigma 2^{-\ell} & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \varepsilon_{Fi}$ in (35) arises from the application of the double angle. Its maximum norm appears for $\ell = 1$. This is assumed in the sequel.

Overall Error: The error bounds are given as in (20) and (21) using C_{AFi} (ACJA) instead of C_{CFi} (CJA). In table 3 factors C_{AFi} are shown for various word lengths. As for floating point arithmetic one observes an improvement of the error bounds for the ACJA (compared to the CJA).

3.4 Comparison

Now both algorithms can be compared for the two types of arithmetic. The ACJA requires more sweeps than the CJA. We introduce the sweep ratio of the CJA and the ACJA as

$$s = \frac{p_C}{p_A}\,, \tag{38}$$

where p_C and p_A are the number of sweeps required by the CJA and the ACJA, respectively. The gain in accuracy of the ACJA compared to the CJA is defined by

$$g_{acc} = s \cdot \frac{C_C}{C_A}\,, \tag{39}$$

where C_C and C_A denote the factors of the CJA and ACJA, respectively. The ACJA is more accurate than the CJA if $g_{acc} > 1$. The overall computational cost of the ACJA is significantly smaller than that for the CJA [8].

As shown in [8] using the most simple version of the ACJA, i.e. one CORDIC angle per rotation, the value of s depends on the word length and the dimension of the matrix (e.g. for a random matrix of dimension $n = 70$ one obtains $s = 6/11$ for $b = 16$ and $s = 8/20$ for $b = 48$, see Fig. 5 of [8]). It is very pessimistic, however, to assume the values of s as obtained by the most simple version since it is possible to regain the quadratic convergence of the Jacobi method by increasing the number of CORDIC angles per rotation during the course of the algorithm [8]. This adaptation of the number of CORDIC angles can be done by a very simple procedure for monitoring the stage of diagonalization during the course of the algorithm [5], such that s can be kept close to 1.

For our overall comparison of the CJA and the ACJA we assume the most pessimistic assumptions made above (e.g. s for one CORDIC angle per rotation, scaling error, ...).

3.4.1 Fixed Point Arithmetic

The new algorithm is superior in accuracy for a word length $b > 8$ (Table 4). The discontinuity in the gain vs. matrix dimensions results from the integer values of the sweep numbers. The ACJA is a significant improvement in the accuracy for large word lengths.

3.4.2 Floating Point Arithmetic

The accuracy gain in floating point arithmetic is not as convincing as for fixed point arithmetic (Table 5). Only for large word lengths the ACJA yields better upper bounds than the CJA. We note, however, that this is to a significant extend due to the pessimistic assumptions made above. The reduced number of shift and add operations required by the ACJA compared to the CJA gives rise to better numerical results also for shorter word lengths than indicated in Table 5. It is important to note that the ACJA is well suited for a floating point CORDIC implementation [7], [9].

4 Conclusion

In this paper we have performed an error analysis for the CJA and the ACJA. The error analysis was performed for floating point and fixed point arithmetic.

The significant reduction of the computational cost of the ACJA compared to CJA is accompanied by smaller numerical errors. This was already mentioned by Delosme in [3] and could be confirmed by the error bounds. The upper bounds for the ACJA using floating point arithmetic are only better than the upper bounds for the CJA if the word length is large. We note, however, that this is due to the pessimistic assumptions (worst cases for scaling and number of required sweeps). Therefore, better numerical examples can actually be expected for wordlength $b > 32$ (even smaller). Furthermore, the ACJA requires a reduced internal wordlength for obtaining the same accuracy in fixed point as well as floating point arithmetic and the ACJA is more amenable to floating point arithmetic than the CJA.

References

[1] 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(1):69–84, 1985.

[2] A. A. DeLange, A. van der Hoeven, E. Deprettere, and J. Bu. An optimal floating point pipeline CMOS CORDIC- processor. In *Proc. IEEE Int. Symp. on Circuit and Systems*, pages 2043–2047, 1988.

[3] J.-M. Delosme. Bit-level systolic algorithms for real symmetric and hermitian eigenvalue problems. *Journal of VLSI Signal Processing*, 4:69–88, 1992.

[4] G. Golub and C. F. van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, 2nd edition, 1989.

[5] J. Götze. Monitoring the stage of diagonalization in Jacobi–type methods. In *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, pages III:441–444, Adelaide (Australia), 1994.

[6] J. Götze. On the parallel implementation of Jacobi and Kogbetliantz algorithms. *SIAM J. Sci. Comput.*, 15:1331–1348, 1994.

[7] J. Götze and G.J. Heckstra. Adaptive approximate rotations for computing the EVD. In *Algorithms and Parallel VLSI Architectures*, M. Moonen and F. Catthor (eds.)., Elsevier. To be published.

[8] J. Götze, St. Paul, and M. Sauer. An efficient Jacobi–like algorithm for parallel eigenvalue computation. *IEEE Transactions on Computers*, 42(9):1058–1063, 1993.

[9] G. J. Heckstra and E. F. Deprettere. Floating point CORDIC. In *11th Symposium on Computer Arithmetic*, Windsor, Canada, 1993.

[10] Yu Hen Hu. The quantization effects of the CORDIC algorithm. *IEEE Transactions on Acoust., Speech, Signal Processing*, **SP-40**:834–844, April 1992.

[11] K. Kota and J. R. Cavallaro. Numerical accuracy and hardware tradeoffs for CORDIC arithmetic for special-purpose processors. *IEEE Transactions on Computers*, 42(7):769–779, 1993.

[12] J. E. Volder. The CORDIC trigonometric computing technique. *IRE Transactions on Electronic Computers*, **EC-8**:330–334, 1959.

[13] J. S. Walther. A univied algorithm for elementary functions. In *AFIPS Spring Joint Computer Conference*, pages 379–385, 1971.

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

Accuracy (t_r bit)	8	16	32	64
Internal CJA (t bit)	12	21	38	71
Internal ACJA (t bit)	11	20	36	68

Table 1: Internal word length of CJA and ACJA for required accuracy.

b	8	16	32	64
C_{CFl}, $n = 10$, $p = 5$	24.2	$9\ 10^{-3}$	$3\ 10^{-7}$	$1\ 10^{-16}$
C_{CFl}, $n = 200$, $p = 9$	– – –	2.1	$2\ 10^{-5}$	$1\ 10^{-14}$
C_{AFl}, $n = 10$, $p = 17$	– – –	$8\ 10^{-3}$	$1\ 10^{-7}$	$4\ 10^{-17}$
C_{AFl}, $n = 200$, $p = 19$	– – –	4.0	$1\ 10^{-5}$	$4\ 10^{-15}$

Table 2: Factor C_{CFl} (14) of CJA and factor C_{AFl} (31) of ACJA.

b	8	16	32	64
C_{CFi}	8.5	15.3	29.1	56.6
C_{AFi}	3.0	3.7	4.4	5.0

Table 3: Factor C_{CFi} (19) of CJA and factor C_{AFi} (37) of ACJA.

n	10	50	100	200
b s	3	2.5	2.25	2
8	1.0	1.1	1.3	1.4
16	1.4	1.7	1.8	2.1
32	2.2	2.6	2.9	3.3
64	3.7	4.4	4.9	5.5

Table 4: Gain in accuracy g_{acc} for fixed point arithmetic.

n	10	50	100	200
b s	3	2.5	2.25	2
8	– – –	– – –	– – –	– – –
16	0.5	0.5	0.5	0.3
32	0.6	0.7	0.8	0.9
64	0.9	1.1	1.2	1.4

Table 5: Gain in accuracy g_{acc} for floating point arithmetic.