AN EFFICIENT JACOBI–LIKE ALGORITHM FOR PARALLEL EIGENVALUE COMPUTATION

Jürgen Götze, Steffen Paul and Matthias Sauer
Institute of Network Theory and Circuit Design
Technical University of Munich
Arcisstr. 21, D-80333 Munich
Germany

Abstract

A very fast Jacobi-like algorithm for the parallel solution of symmetric eigenvalue problems is proposed. It becomes possible by not focusing on the realization of the Jacobi rotation with a CORDIC processor, but by applying approximate rotations and adjusting them to single steps of the CORDIC algorithm, i.e. only one angle of the CORDIC angle sequence defines the Jacobi rotation in each step. This angle can be determined by some shift, add and compare operations.

Although only linear convergence is obtained for the most simple version of the new algorithm, the overall operation count (shifts and adds) decreases dramatically. A slow increase of the number of involved CORDIC angles during the runtime retains quadratic convergence.

1 Introduction

The eigenvalues of a real symmetric $n \times n$ matrix $\mathbf{A}$ are calculated by an orthogonal similarity transform:

$$\mathbf{\Lambda} = \mathbf{Q}^T \mathbf{A} \mathbf{Q}, \qquad \mathbf{Q}^T \mathbf{Q} = \mathbf{I}, \quad \mathbf{\Lambda} \quad \text{diagonal.}$$

The fast computation of eigenvalue decompositions is of great interest in many applications, e.g. signal processing. This concerns efficient algorithms, on the one hand, and their simple implementation either on a multiprocessor system or as dedicated VLSI hardware, on the other hand. The Jacobi algorithm is the method of choice with respect to parallel implementation [1], [12], since it exhibits a significantly higher degree of parallelism than the QR–algorithm. The matrix $\mathbf{Q}$ is composed of simple 2×2 plane rotations calculated such that $\mathbf{A}$ is stepwise reduced to diagonal form by annihilating an off–diagonal element $a_{pq}^{(k)}$, i.e. $a_{pq}^{(k+1)} = 0$, where k denotes the iteration index. If each off–diagonal element is annihilated once, a so called sweep is completed. For cyclic Jacobi methods, the matrix elements a_{pq} are processed in a fixed order, e. g. cyclic ordering scheme [9]

One sweep of the cyclic Jacobi method can be implemented on an upper triangular array of processors with nearest neighbor interconnections [1], [2], [3] (Fig. 1). Each

processor contains a 2×2 block of the upper triangular part of $\mathbf{A}^{(k)}$. The $n/2$ diagonal processors simultaneously evaluate the $n/2$ left and right sided rotations for the 2×2 block they have in storage, such that one sweep consisting of $\frac{n}{2}(n \Leftrightarrow 1)$ two-sided rotations can be executed in $(n \Leftrightarrow 1)$ steps. The left-sided (right-sided) rotations are sent to the off-diagonal processors of the same row (same column), which pre- and postmultiply the rotations received from the left and from the bottom, respectively, to the 2×2 blocks they have in storage.

To simplify the rotation angle calculation two different strategies were followed. One strategy applies the CORDIC (**CO**ordinate **R**otation **DI**gital **C**omputer), well suited for binary data representation, to implement the annihilation of the off-diagonal entry $a_{pq}^{(k)}$ exactly, i.e. $a_{pq}^{(k+1)} = 0$ is demanded. But the exact annihilation of entries is not strictly justified, since in subsequent steps already generated zeros are destroyed. Another strategy does not keep to rigid rotation of $a_{pq}^{(k+1)}$ to zero but demands only a reduction $\left|a_{pq}^{(k+1)}\right| < \left|a_{pq}^{(k)}\right|$ from step to step [9], [22], [3], [10]. This modification is called approximate rotation scheme. Generally, for these schemes the number of sweeps increases but with a simple approximation the cost per sweep is smaller than for the exact algorithm.

In this paper, it is shown that the binary data representation in the CORDIC scheme can be combined with the approximate rotation scheme efficiently.

A CORDIC processor calculates the rotation angles by a fixed length sequence of shift and add operations. The length depends on the required precison and the chosen set of sequences [8], [17]. The modification in the sequences of Volders's original work [19] were introduced to avoid, or at least to simplify, the calculation of the scaling factor, e. g. Delosme proposed a half angle procedure to avoid square roots and divisions for scaling [5].

Approximate rotation schemes are especially appropriate for CORDIC processors. This was already noticed by Delsome [6]. For this purpose the processing of the complete sequence is given up and in every step only a few angles out of this sequence are applied. In [6] 3 or 4 subsequent angles were proposed, but no procedure on how to determine them was presented and it is not guaranteed that all the angles of this sequence are really necessary. It is shown in this paper that the application of just one angle is best for almost all matrices. With a simple preprocessing (shift, add) of the exponents and a few comparisons the angle from the set of CORDIC angles that reduces $\left|a_{pq}^{(k+1)}\right|$ most can be found. This determination requires only a few shift, add and comparison operations [10]. Though the quadratic convergence is lost and the number of sweeps increases, the entire operation count is reduced dramatically. Quadratic convergence is obtained again, if not just one optimal CORDIC angle is employed but a few more, i.e. a multiple application of the one angle procedure whereby, in contrary to [6], no unnecessary angles are applied. Scaling is done in the same way as by Delosme [5]. Since all unnecessary rotation angles are avoided, an improved accuracy is to be expected.

Section 2 reviews the Jacobi algorithm including approximate rotations. Necessary convergence proofs are also given. Implementation issues of the CORDIC algorithm are discussed in Section 3 including the modification to avoid square-roots and divisions in the scaling factor. The new algorithm is described in Section 4 and numerical examples demonstrate the efficiency. The complexities of the new algorithm as well as

other known methods for the symmetric eigenvelue problem are compared in Section 5.

2 Jacobi Methods

2.1 Exact Jacobi Rotations

Jacobi methods for the symmetric eigenvalue problem work by applying a sequence of orthogonal similarity transformations to the symmetric $n \times n$ matrix $\mathbf{A}$ [12]:

$$\begin{aligned} &\mathbf{A}^{(0)} := \mathbf{A} \\ &\text{For } k = 0, 1, 2, \ldots \\ &\qquad \mathbf{A}^{(k+1)} = \mathbf{J}_{pq}^T \mathbf{A}^{(k)} \mathbf{J}_{pq} \end{aligned} \tag{1}$$

where $\mathbf{J}_{pq}$ is a plane rotation in the (p, q) plane defined by the parameters $(c, s, \Leftrightarrow s, c)$ in the (pp, pq, qp, qq) entries of an $n \times n$ identity matrix. $c = \cos\Phi$ ($s = \sin\Phi$) is the cosine (sine) of the rotation angle Φ.

The Jacobi rotations $\mathbf{J}_{pq}$ have to be computed such that the off-diagonal quantity

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

is reduced by each similarity transformation (1). Thereby

$$\lim_{k \to \infty} S^{(k)} \to 0 \Leftrightarrow \lim_{k \to \infty} \mathbf{A}^{(k)} \to \mathrm{diag}(\lambda_i) \quad (\lambda_i := \text{eigenvalues of } \mathbf{A}).$$

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 update (1) affects only rows and columns p and q of $\mathbf{A}^{(k)}$, it is easy to verify that

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

Obviously, the maximal reduction of $S^{(k)}$ is gained if $a_{pq}^{(k+1)} := 0$ holds. This can be achieved by the following cosine-sine pair (c, s) [12]. With $\tau = \frac{a_{qq}^{(k)} - a_{pp}^{(k)}}{2a_{pq}^{(k)}}$ one obtains:

$$t = \tan\Phi = \frac{\mathrm{sign}(\tau)}{|\tau| + \sqrt{1 + \tau^2}} \tag{4}$$

and thus

$$c = \frac{1}{\sqrt{1 + t^2}}, \qquad s = t \cdot c. \tag{5}$$

In the subsequent Sections, a Jacobi rotation $\mathbf{J}_{pq}$ with (c, s) of (5) is called an exact Jacobi rotation due to the fact that $a_{pq}^{(k)}$ is exactly annihilated ($a_{pq}^{(k+1)} = 0$).

2.2 Approximate Jacobi Rotations

For the reduction of $S^{(k)}$, it is not necessary to compute the exact Jacobi rotation, i.e. (c, s) of (5), but it is sufficient to compute (c, s) approximately, such that the orthogonality is preserved and

$$\left|a_{pq}^{(k+1)}\right| \leq \left|da_{pq}^{(k)}\right| \text{ with } 0 \leq |d| < 1. \tag{6}$$

A Jacobi rotation with an approximate computation of (c, s) is called an approximate Jacobi rotation [3], [16] and is described by $\tilde{\mathbf{J}}_{pq}$.

Since the orthogonality has to be preserved for the approximate rotation it is convenient to establish the approximations for $t = \tan\Phi = s/c$ rather than for (c, s). Since (1) yields

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

we obtain

$$d(\tau, t) = \frac{1 \Leftrightarrow 2\tau t \Leftrightarrow t^2}{1 + t^2}. \tag{7}$$

The maximal value of $|d|$, i.e. $|d|_{max}$, is a meassure for the quality of the approximation. It is easy to verify that the exact Jacobi rotation, with t as in (4), i.e. $t = \frac{\text{sign}(\tau)}{|\tau|+\sqrt{1+\tau^2}}$, yields $d = 0$.

The matrices $\mathbf{A}^{(k)}$ converge to a diagonal matrix containing the eigenvalues of $\mathbf{A}$ if the used approximation guarantees $|d(\tau)| < 1$ [9]. Furthermore, if $|d(\tau)|$ decreases during the course of the algorithm, i.e. $|d(\tau)|_{\tau\to\infty} \to 0$, ultimate quadratic convergence is achieved [21], [11]. These statements together with formula (7) are the keys for the design of our new algorithm in Section 4.

There we will use $\tilde{t} = \text{sign}(\tau)\cdot 2^{-\ell}$ $(\ell = 0, 1, 2, \ldots)$ for approximating the tangent t of (4). This approximation corresponds to one iteration step of the CORDIC algorithm. This is the reason for reviewing some basic results of the CORDIC scheme in Section 3.

3 CORDIC Algorithm

There have been lots of publications on the use and implementation of CORDIC for eigenvalue and singular value computations [2],[7],[4],[8],[15],[17]. The CORDIC procedure [19], [20] uses the basic angles $\pm\arctan 2^{-k}$ to compose the desired rotation angle Φ. This can be interpreted as a representation of Φ in an "$\arctan 2^{-k}$" number system with digits $f_k \in \{\Leftrightarrow 1, 1\}$. Hence, after $b + 1$ iteration steps the input vector $(x, y)^T$ is rotated by Φ with a precision of b bits. The iteration equations for the CORDIC rotation procedure in circular coordinates are:

$$\begin{array}{lllr} \text{For} & k = & 0, 1, \ldots, b & \\ & x_{k+1} & = & x_k + \text{sign}(\tau_k) y_k 2^{-k} \\ & y_{k+1} & = & y_k \Leftrightarrow \text{sign}(\tau_k) x_k 2^{-k} \\ & \tau_{k+1} & = & \tau_k \Leftrightarrow \text{sign}(\tau_k) \arctan 2^{-k}. \end{array} \tag{8}$$

The resulting vector after $b + 1$ iterations has to be scaled by

$$\frac{1}{K_b} = \prod_{k=0}^{b} \frac{1}{\sqrt{1 + 2^{-2k}}}$$

independent of the rotation angle. In matrix notation the CORDIC algorithm is governed by

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \frac{1}{K_b} \prod_{k=0}^{b} \begin{pmatrix} 1 & \text{sign}(\tau_k)2^{-k} \\ \Leftrightarrow \text{sign}(\tau_k)2^{-k} & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}. \tag{9}$$

There have been efforts to eliminate the scaling factor or at least bring it into an easy realizable binary representation by introducing additional rotations or correction rotations [8], [2], [5], e.g., when redundant CORDIC methods are used [17], [15]. A modified and fast method using correction rotations which has been extended to angle calculations was presented in [13]. For the exact Jacobi algorithm, K_b is constant for a specific value of b if non-redundant CORDIC is used and thus is precomputable. Therefore, the scaling can be executed using only the necessary additions and shifts instead of a full multiplication, resulting in approximately $b/4$ additions for the scaling. The scaling factor for a two-sided rotation of a 2×2 matrix differs from the scaling factor of a vector rotation by the power of 2.

For methods where the scaling factor is no longer constant Delosme [5] proposed a method for computing a variable scaling factor on-line. This case arises for variable iteration bounds in (8). Let an elementary rotation by angle Φ_k be composed of twice executing a rotation by $\Phi_k/2$. Hence (9) writes:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \frac{1}{K_b^2} \prod_{k=0}^{b} \begin{pmatrix} 1 \Leftrightarrow 2^{-2k} & \text{sign}(\tau_k)2^{-k+1} \\ \Leftrightarrow \text{sign}(\tau_k)2^{-k+1} & 1 \Leftrightarrow 2^{-2k} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}. \tag{10}$$

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

$$(1 + 2^{-2k}) = (1 + 2^{-2k}) \frac{1 \Leftrightarrow 2^{-2k}}{1 \Leftrightarrow 2^{-2k}} = \frac{1 \Leftrightarrow 2^{-4k}}{1 \Leftrightarrow 2^{-2k}}$$

and hence the scaling factor K_b^2 is:

$$K_b^2 = \prod_{k=0}^{b} (1 + 2^{-2k}) = 2 \prod_{k=1}^{b} \frac{1 \Leftrightarrow 2^{-4k}}{1 \Leftrightarrow 2^{-2k}}. \tag{11}$$

As for a given precision b of the adders in a processor, all factors $(1 \Leftrightarrow 2^{-i})$ with $i > b$ do not contribute to K_b. So all terms in the numerator of (11) can be cancelled out and the scaling factor simplifies to:

$$\frac{1}{K_b^2} = 2 \prod_{k=1}^{\lceil b/4 \rceil} (1 \Leftrightarrow 2^{-(4k-2)}). \tag{12}$$

In case of taking only one specific iteration step ℓ of the product (10), i.e.

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \frac{1}{K_\ell^2} \begin{pmatrix} 1 \Leftrightarrow 2^{-2\ell} & \text{sign}(\tau_\ell)2^{-\ell+1} \\ \Leftrightarrow \text{sign}(\tau_\ell)2^{-\ell+1} & 1 \Leftrightarrow 2^{-2\ell} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}, \tag{13}$$

the scaling factor depends on ℓ and thus the scaling hardware must offer the flexibility to perform all possible multiplications. An expansion similar to that of (11) can be made such that the scaling in (13) can be performed recursively as follows:

$$K_{i+1}^2 = K_i^2 (1 + 2^{-2^{i+1}\ell}); \; K_1^2 = 1 \Leftrightarrow 2^{-\ell}. \tag{14}$$

The recursion terminates if $2^{i+1}\ell < b$, hence after $\log_2 \lceil \frac{b}{2\ell} \rceil$ steps.

4 New Algorithm

4.1 The Approximate Rotation

An approximation of (4) based on the idea of the CORDIC algorithm is

$$\tilde{t} = \mathrm{sign}(\tau) \cdot 2^{-\ell} \approx t \tag{15}$$

where $\ell \in \{0, 1, 2, \ldots, b\}$. Here ℓ is choosen such that $\tilde{\Phi} = \arctan 2^{-\ell}$ is the angle which is closest to the exact rotation angle Φ.

Describing $\mathbf{J}_{pq}$ in dependence of t and using this approximation yields

$$\tilde{\mathbf{J}}_{pq}(\ell) = \frac{1}{\sqrt{1+2^{-2\ell}}} \begin{bmatrix} 1 & \mathrm{sign}(\tau)2^{-\ell} \\ \Leftrightarrow \mathrm{sign}(\tau)2^{-\ell} & 1 \end{bmatrix} \tag{16}$$

While the CORDIC algorithm evaluates the exact rotation $\mathbf{J}_{pq}$ by executing iterations (i.e. (16) is executed for $\ell = 0, 1, 2, \ldots, b$) for determing t, here only one iteration (one definite ℓ) is executed for an approximate rotation $\tilde{\mathbf{J}}_{pq}(\ell)$.

4.2 Evaluation of $\tilde{\mathbf{J}}_{pq}(\ell)$

For the evaluation of the Jacobi rotation $\tilde{\mathbf{J}}_{pq}(\ell)$ it is sufficient to determine the shift-value ℓ. For this determination of ℓ and for a minimization of $|d|_{max}$ we consider $d(|\tau|, \ell)$ as obtained from (7) with $t := \mathrm{sign}(\tau)2^{-\ell}$:

$$d(|\tau|, \ell) = \frac{1 \Leftrightarrow 2|\tau|2^{-\ell} \Leftrightarrow 2^{-2\ell}}{1+2^{-2\ell}}. \tag{17}$$

Figure 2 shows $d(|\tau|, \ell)$ vs. τ for $\ell = 0, 1, \ldots$. Obviously, $|d(|\tau|, \ell)| \leq 1/3$ can be achieved if we set $\ell = i$ for $|\tau| \in [\tau_i, \tau_{i+1}[$. The bounds of these intervals (the τ_i) can be obtained from (17) with $|d(|\tau_i|, \ell)| = 1/3$ and $\ell = i$, which yields

$$|\tau_i| = \frac{1}{3}\left(2^i \Leftrightarrow 2^{-i+1}\right). \tag{18}$$

Using (18), the comparison $|\tau_i| \leq |\tau| < |\tau_{i+1}|$ for determing ℓ can also be referred to shift-and-add operations. With $|a_D| = |a_{qq} \Leftrightarrow a_{pp}|$ we obtain the following comparison for the determination of ℓ (i.e. $\tilde{\mathbf{J}}_{pq}$):

$$\begin{aligned} &\text{If} \quad (2^i \Leftrightarrow 2^{-i+1})\,|a_{pq}| \leq |a_D| + 2^{-1}|a_D| < (2^{i+1} \Leftrightarrow 2^{-i})\,|a_{pq}| \\ &\text{Then} \qquad\qquad \ell = i. \end{aligned} \tag{19}$$

In order to avoid a great number of comparisons (e.g. compare for $i = 0, 1, 2, \ldots$ until ℓ is determined) an estimate of the interval $[\tau_i, \tau_{i+1}[$, i.e. an estimate for ℓ, is required. Such an estimate can be obtained by computing an estimate for $|\tau|$. With $\exp(a)$ describing the exponent of the number a and with $|\tau| = |a_D|/2|a_{pq}|$ one obtains

$$|\tau| \in \left[2^{k-2}, 2^k\right[\text{ with } k = \exp(a_D) \Leftrightarrow \exp(a_{pq}). \tag{20}$$

Comparing these intervals $\left[2^{k-2}, 2^k\right[$ yielding an estimate for $|\tau|$ with the intervals $[|\tau_i|, |\tau_{i+1}|[$ ($|\tau_i|$ from (18)), which determine the values of ℓ ($\ell = i$), shows that maximal three values of ℓ are possible for a given k. This can be seen easily by considering the corresponding intervalls in Fig. 2.

$$\begin{array}{ll} k \leq \Leftrightarrow 2 & \ell = 0 \\ \Leftrightarrow 2 \leq k \leq 0 & \ell \in \{0, 1\} \\ k > 0 & \ell \in \{k \Leftrightarrow 1, k, k+1\}\,. \end{array} \tag{21}$$

A maximum of two comparisons (19) must be executed for determing ℓ if $k > 0$.

4.3 Scaling

The problem of scaling $\tilde{\mathbf{J}}_{pq}(\ell)$ has been adressed in Section 3. Since the scaling factor depends on ℓ we obtain a different scaling factor for every ℓ. According to eq. (12), scaling can be performed by a sequence of at most $\log_2 \lceil \frac{b}{2\ell} \rceil$ shift and add operations if we execute two rotations with the half rotation angle. Therefore, in virtue of an easy compensation of the scaling factor, we use a slightly different approximation than (15). The approximation corresponds to working with the next smaller angle, i.e. setting

$$\ell := \ell + 1 \tag{22}$$

and applying the rotation $\tilde{\mathbf{J}}_{pq}(\ell)$ two times which corresponds to applying

$$\bar{\mathbf{J}}_{pq}(\ell) = \begin{bmatrix} 1 \Leftrightarrow 2^{-2\ell} & \text{sign}(\tau) 2^{-\ell+1} \\ \Leftrightarrow \text{sign}(\tau) 2^{-\ell+1} & 1 \Leftrightarrow 2^{-2\ell} \end{bmatrix} \tag{23}$$

and executeing the scaling by using the expansion (14), e.g. for $b = 32$

$$\frac{1}{K_\ell^2} = \frac{1}{1 + 2^{-2\ell}} = (1 \Leftrightarrow 2^{-2\ell})(1 + 2^{-4\ell})(1 + 2^{-8\ell})(1 + 2^{-16\ell})(1 + 2^{-32\ell}) \tag{24}$$

for each rotation (23).

By setting $\ell := \ell + 1$ (22) and using a double rotation (23) the first approximation (15) has been changed. Figure 3 shows $d(|\tau|, \ell)$ resulting from these changes. The maximal reduction factor $|d|_{max}$ is bounded to 0.6. However, the comparison of Figures 2 and 3 show that this approximation converges very rapidly to our original approximation (15) (Fig. 4). Therefore, the convergence is only deteriorated for small $|\tau|$, i.e. in the very beginning of the algorithm.

4.4 Summary of the Algorithm

In this subsection we summarize the new algorithm for the diagonal and the off–diagonal processor cells. Throughout this description $\mathbf{A}$ denotes the 2×2 matrix contained in the corresponding processsor cell.

The algorithm for the diagonal processor cells (for later use $[\mathbf{A}, \ell] = vect(\mathbf{A})$ refers to a call of this algorithm) works as follows:

- determine the shift value ℓ such that $\tilde{\Phi} = \arctan(2^{-\ell})$, $\ell = 0, 1, 2, \ldots, b$ is the best approximation of the exact rotation angle Φ, i.e. execute the comparison (19) due to (21) with k of (20),
- in virtue of an easy scaling factor compensation set $\ell := \ell + 1$ (22) and use $\bar{\mathbf{J}}_{pq}(\ell)$ of (23) (instead of $\tilde{\mathbf{J}}_{pq}(\ell)$),
- compute $\mathbf{A} := \bar{\mathbf{J}}_{pq}^T(\ell) \cdot \mathbf{A} \cdot \bar{\mathbf{J}}_{pq}(\ell)$ with $\bar{\mathbf{J}}_{pq}(\ell)$ of (23),
- compensate the scaling factor $K_\ell^2 = 1 + 2^{-2\ell}$ by executing the shift–and–add steps required due to (24) two times (left and right sided rotation).

The value of ℓ is sent to the off–diagonal cells of the same row and the same column.

An off-diagonal cell which obtains ℓ_1 from the right and ℓ_2 from the bottom works as follows ($[\mathbf{A}, \ell_1, \ell_2] = rot(\mathbf{A}, \ell_1, \ell_2)$):

- compute $\mathbf{A} := \bar{\mathbf{J}}_{pq}^T(\ell_1) \cdot \mathbf{A} \cdot \bar{\mathbf{J}}_{pq}(\ell_2)$ with $\bar{\mathbf{J}}_{pq}(\ell)$ of (23).
- compensate the scaling factors K_{ℓ_1} and K_{ℓ_2} by executing the shift–and–add steps required due to (24) for $\ell = \ell_1$ and $\ell = \ell_2$.

4.5 Discussion and Numerical Examples

As can be seen from Figure 3, the new algorithm guarantees $|d|_{max} = 0.6 < 1$ and therefore linear convergence of the Jacobi method. However, the condition for quadratic convergence is not met since for $\tau \to \infty$ only $|d(\tau)| \leq 1/3$ holds but not $d(\tau) \to 0$. Although the ultimate quadratic convergence is lost and therefore the number of sweeps increases, the complexity is significantly reduced if the presented one angle rotation scheme is applied.

In Figure 5, the decrease of the off–diagonal norm S of a random matrix of dimension $n = 70$ is shown. The exact Jacobi method converges ultimately quadratic while the approximate CORDIC Jacobi method converges only linearly. However, since the convergence of the exact Jacobi method is also only linear in the first sweeps, the difference in the number of required sweeps only grows significantly if the tolerance for the off–diagonal norm, which stops the algorithm, is decreased. (Note that in a parallel environment the number of sweeps is predetermined). A decrease of the tolerance requires an increase of the wordlength b, e.g. for $S_{tol} = 10^{-14}$ to be a reasonable limit a wordlength of at least $b = 48$ is required and correspondingly $b = 32$ for $S_{tol} = 10^{-10}$ resp. $b = 16$ for $S_{tol} = 10^{-5}$. Thus for usual lengths of the datawords the difference in the number of required sweeps is minor in comparison to the reduced complexity of the new algorithm (e.g. in Figure 5 for $b = 32$ we require 18 instead of 7 sweeps but in each sweep only one instead of 32 CORDIC iterations is executed per rotation). This estimate is even very pessimistic since actually $\frac{n(n-1)}{2}$ matrix elements of length b each are accumulated for determing the off–diagonal norm S. Numerical simulations indicate an increase of the number of sweep by a factor $2 \Leftrightarrow 3$. Note that the higher performance of the new algorithm can be evaluated only if the number of sweeps and the cost per sweep are considered.

4.6 Retaining the Ultimate Quadratic Convergence

In the following we describe how to retain the ultimate quadratic convergence, although, as mentioned above, for usual lengths of the datawords using one angle rotations yields the best results. If the quadratic convergence shall be retained the algorithm of the diagonal cell can be iterated, i.e.

$$\text{For } m = 1, 2, \ldots, r$$
$$[\mathbf{A}, \ell] = \text{vect}(\mathbf{A}) \tag{25}$$

Thereby, by increasing r the accurracy of the rotation is improved and converges to the exact rotation which guarantees ultimate quadratic convergence. Figure 6 shows an example for the rotation of a 2×2 matrix $\mathbf{A}$ with $r = 5$. Note, that this example illustrates the main difference to the original CORDIC approach. In contrast to the original CORDIC scheme, only the required CORDIC angles are executed (unnecessary CORDIC angles are skipped). As can be seen in Figure 6, $r = 5$ is sufficient to achieve the same result as the original CORDIC algorithm for $b = 16$; i.e. 11 of the 16 CORDIC angles are not necessary. The fact that the first angle found for $m = 1$ yields the greatest reasonable angle of the CORDIC sequence and the fact that the produced zero elements are destroyed during the Jacobi method anyway (i.e. producing a zero is usually not justified) are the reasons for the improved performance of the new algorithm.

Thus, the ultimate quadratic convergence can be retained if the value of r is increased during the Jacobi method (e.g. using $r = 1$ in the first sweeps and $r = 4$ for the last couple of sweeps). For the processor array, the increase of r merely means that the dataflow of the matrix elements is delayed by r (in the processor cells the same 2×2 matrix is iterated for r steps). All the other computations (vectorization in the diagonal cells and rotation in the off–diagonal cells) run as for $r = 1$.

5 Complexity

The reduced overall operation count of the new algorithm can be evaluated by a complexity analysis. In order to obtain a realistic complexity measure we distinguish additions (i.e arithmetic operations) and shifts. We will evaluate the complexities for the execution of a rotation operation (C_{rot}), for the determination of the rotation angle (C_{angle}) and for the scaling (C_{scaling}). During one sweep of the Jacobi algorithm for a $n \times n$ matrix $n(n \Leftrightarrow 1)/2$ rotation and scaling operations and $n/2$ angle determination operations have to be executed. In the exact Jacobi algorithm "angle determination" means that the third equation for the CORDIC in (8) has to be computed in order to determine the correct sequence of positive and negative rotations. For the comparison we assume that the same adder implementation is used in the three algorithms and therefore denote the adder complexity by ADD. The use of redundant adders like, e.g. in [17] or [13] does not alter the principal results of the evaluation. However, redundant adders may significantly increase the achievable clock rates in a hardware implementation as the time for an addition is then independent from the operand precision.

The actual complexity of a shift by k bits depends on the way it is implemented. In order to keep the discussion independent form implementation aspects we use a common cost figure $SHIFT$ for the shift operation which is independent from the shift

distance. The actual complexities of the implemented shift operations have to be taken into consideration if the details of the implementation are known. The double rotation approach is included into the comparison, because it is used for the approximate rotations proposed by Delosme [6].

For the evaluation of the angle determination in (19) we assume that a comparison operation ($\leq$) is approximately as expensive as an addition. Furthermore we neglect the cost of sign conversion and thus evaluate addition and subtraction equally.

The conventional CORDIC requires two additions and two shifts per iteration for the rotation operation, hence

$$C_{\text{rot,cordic}} = 2bADD + 2bSHIFT. \tag{26}$$

The angle determination , i.e. the evaluation of the τ-iteration in (8) requires

$$C_{\text{angle,cordic}} = bADD \tag{27}$$

and the scaling is done with approximately $b/4$ additions, which yields

$$C_{\text{scaling,cordic}} = \frac{b}{2}ADD + \frac{b}{2}SHIFT. \tag{28}$$

For the double rotation approach we get

$$C_{\text{rot,dr}} = 4bADD + 4bSHIFT \tag{29}$$

for the rotation and

$$C_{\text{angle,dr}} = C_{\text{angle,cordic}} = bADD \tag{30}$$

for the angle determination. Scaling is done according to (12) which yields

$$C_{\text{scaling,dr}} = 2(\lceil\frac{b}{4}\rceil ADD + \lceil\frac{b}{4}\rceil SHIFT). \tag{31}$$

The complexity of the rotation operation in the new algorithm corresponds to one iteration step in the double rotation method, hence to (13) and is therefore

$$C_{\text{rot,ss}} = 4ADD + 4SHIFT. \tag{32}$$

Scaling is performed according to (14) which results in an operation count of:

$$C_{\text{scaling,ss}} = \log_2 \lceil\frac{b}{2\ell}\rceil (ADD + SHIFT) \leq \log_2 \lceil\frac{b}{2}\rceil (ADD + SHIFT). \tag{33}$$

In the new algorithm the "angle determination" corresponds to the selection of ℓ with (19) and (20). One addition is required for the determination of $|a_D|$ and one addition for the calculation of k in (20). One evaluation of (19) requires 5 additions and 4 shifts. As there are maximal two comparisons to be executed, this accumulates to

$$C_{\text{angle,ss}} = 12ADD + 8SHIFT. \tag{34}$$

Note that the complexities for the new algorithm are of $O(1)$ or $O(\log b)$ compared to $O(b)$ for the exact Jacobi rotations. The number of additions and shifts per sweep is obtained from:

$$C_{\text{sweep}} = \frac{n(n \Leftrightarrow 1)}{2}C_{\text{angle}} + \frac{n(n \Leftrightarrow 1)(n \Leftrightarrow 2)}{2}C_{\text{rot}} + \frac{n(n \Leftrightarrow 1)^2}{2}C_{\text{scaling}} \tag{35}$$

In Fig. 7, these complexities are plotted versus the precision b for a matrix dimension $n = 70$.

6 Conclusions

A highly efficient Jacobi-like algorithm for the parallel solution of symmetric eigenvalue problems was proposed. The algorithm combines implementation properties of CORDIC processors for vector rotation with approximate rotation schemes.

From the binary data representation, the best suited CORDIC rotation angle for a reduction of the off-diagonal norm can be obtained very easily and only this angle is applied. The square-roots and divisions for scaling are avoided by the subsequent application of two rotations of half angle and some algebraic manipulations.

Though the quadratic convergence is lost, the algorithm is much faster than other Jacobi-like algorithms with CORDIC processors. The higher performance of the new algorithm is due to the fact that the savings in the rotation computations (one angle) are significantly higher than the cost of the additional sweeps. Rounding error analysis has shown that the savings in computations are accompanied by significantly smaller error bounds. Details will be published in a forthcoming paper.

The algorithm can be modified in the sense that more than one rotation angle is performed, but always only the necessary angles. It is also possible to increase the number of rotation angles as the algorithm proceeds. By doing so, quadratic convergence can be retained.

The proposed algorithm is also well-suited for floating point arithmetic (in contrary to the conventional CORDIC), because only one floating point unit (i.e. adder plus shifter) is necessary for each processing cell.

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(1985), 69–84.

[2] J. R. Cavallaro and F. D. Luk. CORDIC arithmetic for an SVD processor. *Journal of Parallel and Distributed Computing*, 5:271–290, 1988.

[3] 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), 279-300.

[4] A. A. J. de Lange, A. J. van der Hoeven, E. F. Deprettere, and J. Bu. An optimal floating-point pipeline CMOS CORDIC processor, algorithm, automated design, layout and performance. In *Proc. IEEE Int. Symp. on CAS*, pages 2043–2047, 1988.

[5] J.-M. Delosme. CORDIC algorithms: theory and extensions. In *Proc. SPIE Advanced Algorithms and Architectures for Signal Processing* IV(1989), 131–145.

[6] J.-M. Delosme. Bit-level systolic algorithms for real symmetric and hermitian eigenvalue problems. *Journal of VLSI Signal Processing*, 4(1):69–88, 1992.

[7] E. F. Deprettere, A. A. J. de Lange, and P. Dewilde. The synthesis and implementation of signal processing applications specific VLSI CORDIC arrays. In *Proc. Int. Conf. Acoustics Speech and Signal Processing*, pages 974–977, 1990.

[8] J. Duprat and J.-M. Muller. The CORDIC algorithm: new results for fast VLSI implementation. Technical Report 90-04, ENS de Lyon, 1990.

[9] 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), 1–23.

[10] J. Götze. Parallel methods for iterative matrix decompositions. In *Proc. IEEE Int. Symp. on CAS*, pages 233–236, 1991.

[11] J. Götze. On the Parallel implementation of Jacobi's and Kogbetliantz's algorithm. Research Report RR–879, Department of Computer Science, Yale University, New Haven, November 1991. (to appear SIAM J. Sci. & Stat. Comput.)

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

[13] J.-A. Lee and T. Lang. Constant-factor redundant CORDIC for angle calculation and rotation. *IEEE Transactions on Computers*, C-41(8):1016–1025, August 1992.

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

[15] H. X. Lin and H. J. Sips. On-line CORDIC algorithms. *IEEE Transactions on Computers*, C-39:1038–1052, 1990.

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

[17] N. Takagi, T. Asada, and S. Yajima. Redundant CORDIC methods with a constant scale factor for sine and cosine computation. *IEEE Transactions on Computers*, 40(9):989–995, 1991.

[18] J. D. Ullman. *Computational Aspects of VLSI.* Computer Science Press, 1984.

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

[20] J. Walther. A unified algorithm for elementary functions. In *Joint Computer Conference Proceedings*, 1971.

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

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

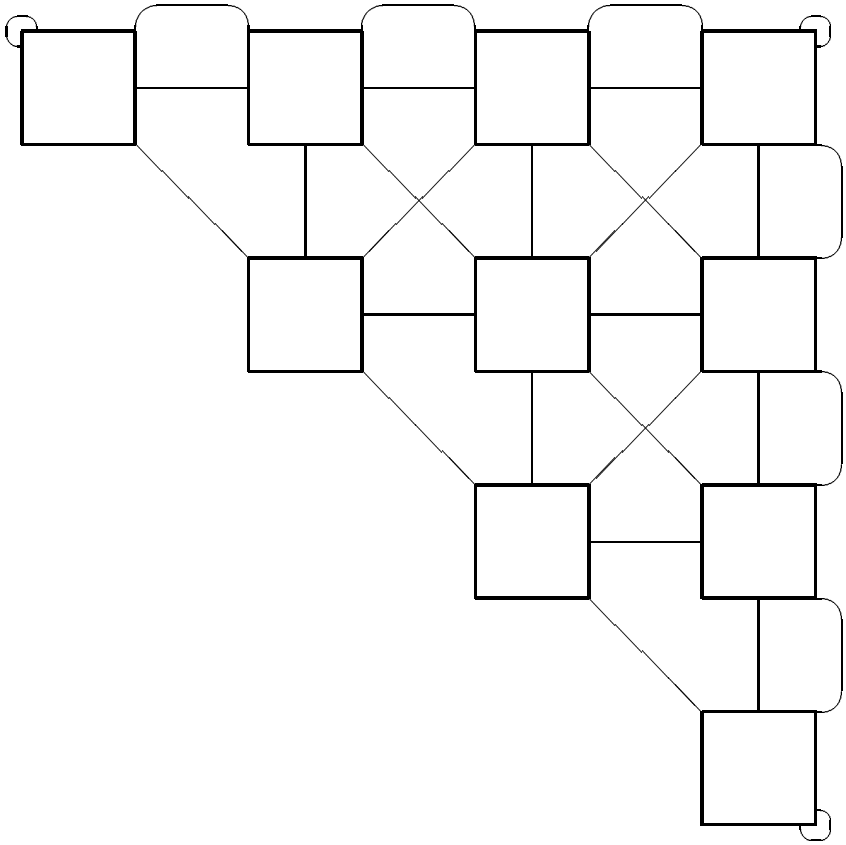

Figure 1: Processor array for the Jacobi algorithm ($n = 8$).

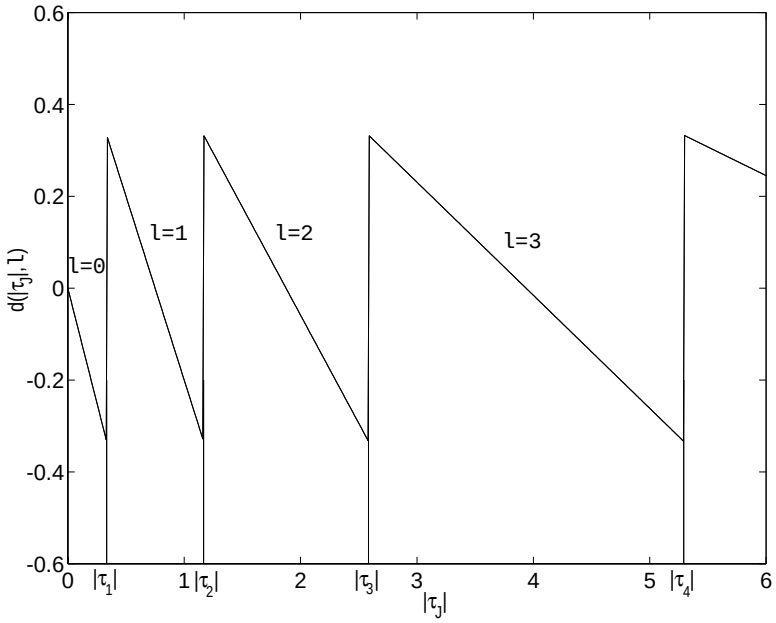

Figure 2: Reduction factor d vs. $|\tau|$, ℓ for approximate rotation $\tilde{\mathbf{J}}_{pq}$.

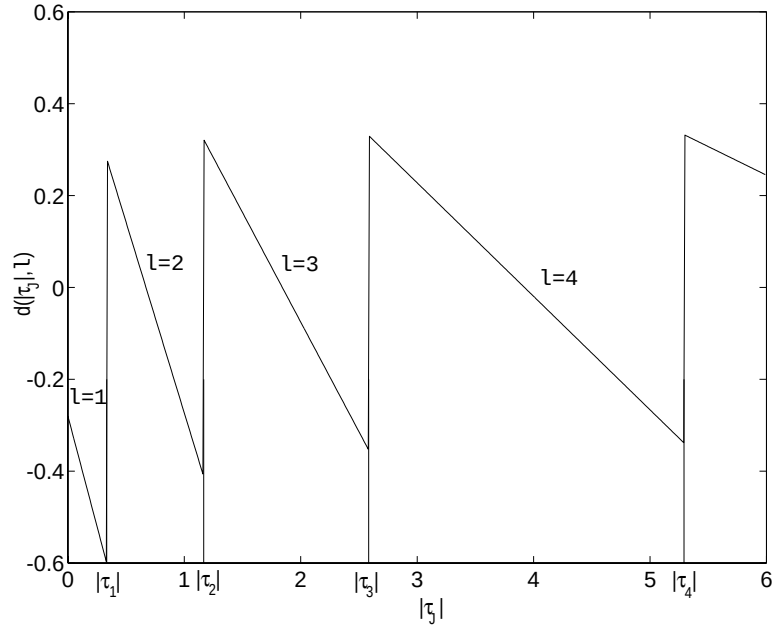

Figure 3: Reduction factor d vs. $|\tau|$, ℓ for approximate rotation $\bar{\mathbf{J}}_{pq}$.

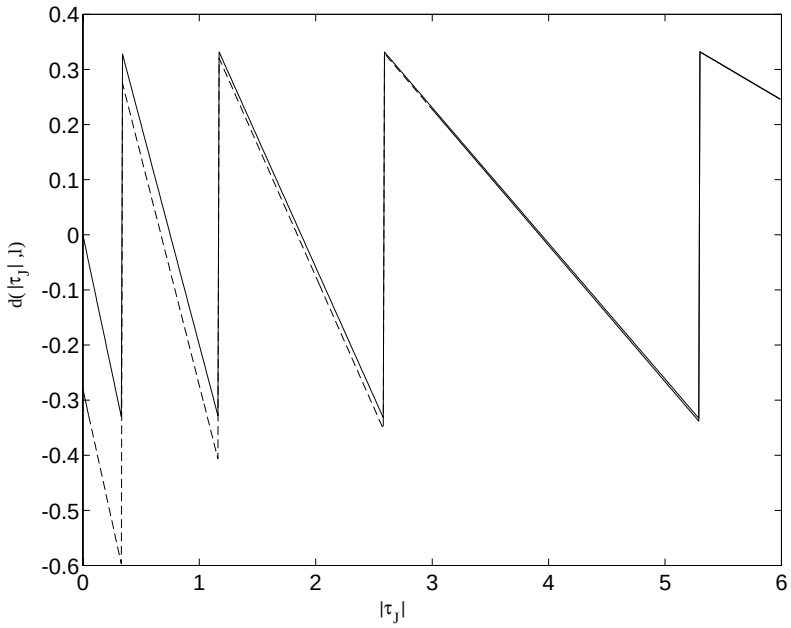

Figure 4: Comparison of Figs. 2 and 3.

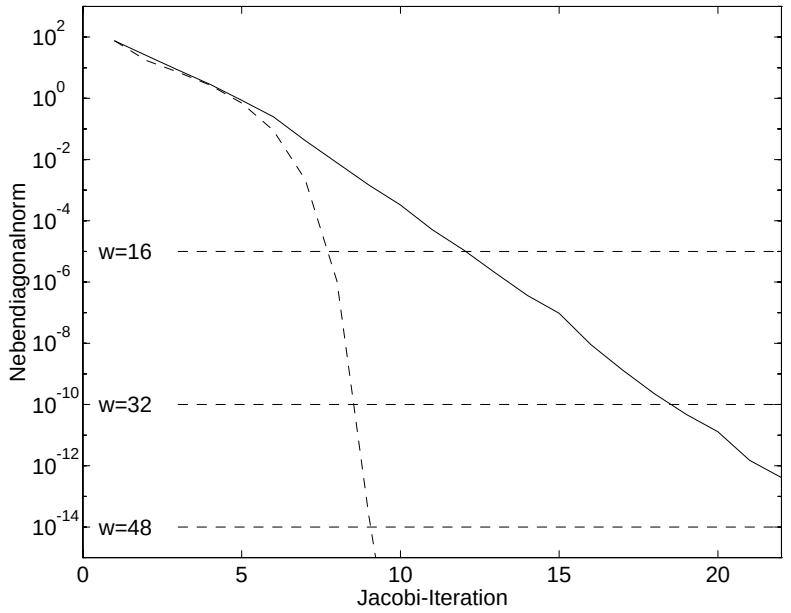

Figure 5: Reduction of off-diagonal norm vs. sweep number for random matrix of dimension $n = 70$ for conventional Jacobi algorithm with CORDIC (solid line) and one angle Jacobi algorithm (dashed line).

$$\mathbf{A} = \begin{bmatrix} 1 & 2 \\ 2 & 5 \end{bmatrix}$$

$m = 1: \quad \ell = 2 \qquad \Phi = 28.0725 \qquad \mathbf{A} = \begin{bmatrix} 0.2249 & \Leftrightarrow 0.5467 \\ \Leftrightarrow 0.5467 & 5.7751 \end{bmatrix}$

$m = 2: \quad \ell = 4 \qquad \Phi = \Leftrightarrow 7.5127 \qquad \mathbf{A} = \begin{bmatrix} 0.1759 & 0.1559 \\ 0.1559 & 5.8241 \end{bmatrix}$

$m = 3: \quad \ell = 6 \qquad \Phi = 1.7903 \qquad \mathbf{A} = \begin{bmatrix} 0.1716 & \Leftrightarrow 0.0207 \\ \Leftrightarrow 0.0207 & 5.8284 \end{bmatrix}$

$m = 4: \quad \ell = 9 \qquad \Phi = \Leftrightarrow 0.2238 \qquad \mathbf{A} = \begin{bmatrix} 0.1716 & 0.0013 \\ 0.0013 & 5.8284 \end{bmatrix}$

$m = 5: \quad \ell = 13 \qquad \Phi = 0.0140 \qquad \mathbf{A} = \begin{bmatrix} 0.1716 & 0 \\ 0 & 5.8241 \end{bmatrix}$

$m = 6: \quad \ell = 18 > b = 16$

Figure 6: Iterative application of the new algorithm for a 2×2 matrix.

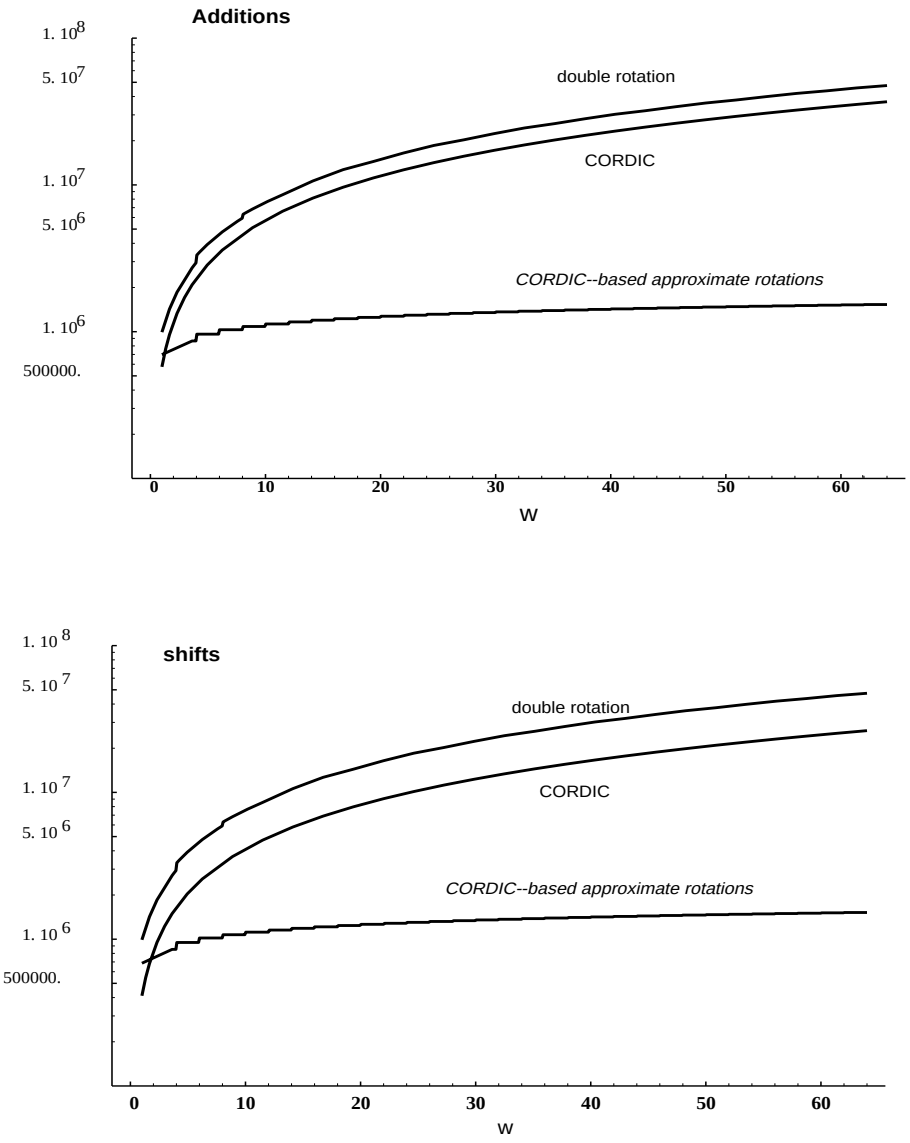

Figure 7: Comparison of shift and add complexities vs. precision b of exact Jacobi with one-angle approach for $n = 70$.