On an Implementation of Two-Sided Jacobi Method

Sho Araki¹, Masami Takata², Kinji Kimura³, and Yoshimasa Nakamura¹
¹Graduate School of Informatics, Kyoto University, Kyoto, Kyoto, JAPAN
²Research Group of Information and Communication Technology for Life, Nara Women’s University, Nara, Nara, JAPAN
³Department of Electrical and Electronics Engineering, University of Fukui, Fukui, Fukui, JAPAN

Abstract—The Jacobi method for singular value decomposition can compute all singular values and singular vectors with high accuracy. Previously published studies have reported that Jacobi’s method is more accurate than the QR algorithm. The computation cost in the Jacobi method is higher than that of the computation method, which combines the QR method and bidiagonalization using the Householder transformation. However, the computation cost is insignificant for very small matrices. Moreover, the Jacobi method can be implemented as an embedded system such as FPGA because of its simple operation pattern. Based on the Jacobi method, one-sided and two-sided Jacobi methods are proposed. The one-sided method has already been implemented in LAPACK. There are still many parts which can be improved in the implementation of the two-sided Jacobi method. Thus, in this paper, we improve the two-sided Jacobi method. We confirmed through experiments that the two-sided Jacobi method has a shorter computation time and exhibits a higher accuracy than the one-sided Jacobi method for small matrices.

Keywords: singular value decomposition, one-sided Jacobi method, two-sided Jacobi method, False position method, secant method, fused multiply–accumulate

1. Introduction

Many mathematical applications require a generalized eigenvalue formula comprising a symmetric matrix and a positive definite symmetric matrix, although these applications use only some eigenvalues and the corresponding eigenvectors. The Sakurai-Sugiura method [15] is known as a truncated eigenvalue decomposition and uses a column space. To compute the column space, a rectangular matrix should be decomposed using a singular value decomposition. Generally, a given matrix is transformed into a bidiagonal matrix by using the Householder transformation [3] as a preprocessing method. In [1], a computation method for column space, adopted to a bidiagonal matrix, has been proposed. The method combines the DQDS (differential qd with shift) [7], [13] and OQDS (orthogonal qd with shift) methods [12]. The Sakurai-Sugiura method is sufficient only for computing column space, which is based on the left singular vectors, in a given upper bidiagonal matrix. Because the row space in a lower bidiagonal matrix is equal to the column space in the upper bidiagonal matrix, it can be computed based on the right singular vectors, achieved by using the OQDS method, which was proposed in [1].

To reduce the computation costs and improve accuracy, the Sakurai-Sugiura method was modified by Imakura et al. [9]. The modified Sakurai-Sugiura method requires both the left and right singular vectors. The OQDS method can achieve high accuracy when only column space is required. Thus, the OQDS method is compatible with the original Sakurai-Sugiura method. However, when the left singular vectors of the lower bidiagonal matrix is computed by the OQDS method, the OQDS method requires a matrix twice as large as the given matrix size. The left singular vectors of a matrix are obtained by extracting the smaller matrix from the larger matrix. Consequently, it is not guaranteed that the computed left singular vectors in the lower bidiagonal matrix will have high orthogonality. Hence, to implement the modified Sakurai-Sugiura method with high accuracy, it is necessary to establish a method with high accuracy for singular value decomposition, which can address all singular values and left and right singular vectors.

James Demmel and Kresimir Veselic reported in their paper that the Jacobi method is more accurate than QR [4]. As the Jacobi method is for singular value decomposition, one-sided and two-sided Jacobi methods have been proposed [5], [6], [2], [8], [10]. The one-sided Jacobi method was implemented in LAPACK [11]. There are still many parts which can be improved in the implementation of the two-sided Jacobi method. Thus, in this paper, we improve the two-sided Jacobi method. Experimental results confirmed that the two-sided Jacobi method has shorter computation time and higher accuracy than the one-sided Jacobi method for small matrices.

2. Target matrices

The two-sided Jacobi method for eigenvalue decomposition can compute the eigenvalues and eigenvectors of a real symmetric matrix. More precisely, it is also possible to extend the target matrix to Hermitian matrix. Actually, the two-sided Jacobi method for singular value decomposition can even be designed to perform computations on
complex matrices of any size. However, in this paper, we have considered only the real upper triangular matrices. By preprocessing using the QR and LQ decompositions in the case of rectangular matrices, singular value decomposition of the rectangular matrices can be reduced to that of upper triangular matrices. Moreover, since we can easily extend our method to an upper complex matrix, a singular value decomposition using the two-sided Jacobi method is designed for allowing computations on real upper triangular matrices.

3. Singular value decomposition using two-sided Jacobi method

3.1 Outline

Let $J^{(i)}$, $K^{(i)}$, $N^{(i)}$, and $M^{(i)}$ be the products of rotation matrices. Let $R^{(i)}$ and $L^{(i)}$ be set to a real upper and lower triangular matrix, respectively. In a singular value decomposition using two-sided Jacobi method, eqs.(1) and (2) are computed repeatedly.

\[ K^{(i)}R^{(i)}J^{(i)} = L^{(i)} , \]
\[ N^{(i)}L^{(i)}M^{(i)} = R^{(i+1)} , \quad i = 0, 1, \ldots \]  \hspace{1cm} (1)
\hspace{1cm} (2)

By these iterative computations, $R^{(i)}$ and $L^{(i)}$ converge into a diagonal matrix. In the convergence, the left singular vector $U$ and the right singular vector $V$ can be computed as follows.

\[
U = \begin{pmatrix} K^{(0)} \top & N^{(0)} \top & K^{(1)} \top & N^{(1)} \top & \cdots & K^{(m-1)} \top & N^{(m-1)} \top \end{pmatrix} ,
\]
\[
V = J^{(0)}M^{(0)}J^{(1)}M^{(1)} \cdots J^{(m-1)}M^{(m-1)} , \]  \hspace{1cm} (3)
\hspace{1cm} (4)

where $m$ is the iteration number in the case of convergence. Here, the matrix multiplication in eqs. (3) and (4) is accomplished by Givens rotations. Fig.1 shows that $R^{(i)}$ and $L^{(i)}$ are stored together in the upper triangular matrix. In the case of Fig.1, memory allocation is not needed for $R^{(i)}$ and $L^{(i)}$ as separate matrices. Hence, $R^{(i)}$ and $L^{(i)}$ can be computed in the same memory area itself.

As shown in eq. (7), $R_{j,k}$ is converted to 0 by using the rotation matrices $P$ and $Q$. Here, $I$ means an identity matrix.

\[
P = \begin{pmatrix} I & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{pmatrix} , \]  \hspace{1cm} (5)
\[
Q = \begin{pmatrix} I & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{pmatrix} , \]  \hspace{1cm} (6)

By repeating the eq.(7), $R^{(i)}$ can be transformed into $L^{(i)}$. However, it should be noted that the eq. (7) is not the computation of $L^{(i)}$ from $R^{(i)}$. Thus, the eq. (7) is not expressed using $R_{j,j}^{(i)}$, $R_{j,k}^{(i)}$, $R_{k,k}^{(i)}$, $J_{j,j}^{(i)}$, and $L_{k,k}^{(i)}$. Since $P$ and $Q$ are rotation matrices, $\theta_1$ and $\theta_2$ satisfy $c_1 = \cos \theta_1$, $s_1 = \sin \theta_1$, $c_2 = \cos \theta_2$, and $s_2 = \sin \theta_2$. Hereafter, we will discuss only those element parts in which the values show a change.

\[
\begin{pmatrix} c_1 & s_1 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{pmatrix} \begin{pmatrix} R_{j,j} \cdots R_{j,k} \cdots \cdots \cdots \cdots \cdots \cdots \end{pmatrix} \begin{pmatrix} c_2 & -s_2 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{pmatrix} = \begin{pmatrix} \hat{R}_{j,j} & \cdots & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{pmatrix} . \]  \hspace{1cm} (7)
\hspace{1cm} (8)

To compute $L^{(i)}$ from $R^{(i)}$, the eq.(8) needs to be repeated many times. In iterative procedures, an ordering strategy for erasing the off-diagonal elements, explained in sec.3.2, is adopted. We use the same procedure to obtain $R^{(i+1)}$ from $L^{(i)}$.

Computation of $c_1$, $s_1$, $c_2$, $s_2$, $R_{j,j}$, and $R_{k,k}$ from $R_{j,j}, R_{j,k}, R_{k,k}$ is explained in sec.3.3, 3.4 and 3.5.
3.2 Ordering strategy and convergence criterion

In the ordering strategy, off-diagonal elements in an upper triangular matrix $R^{(i)}$ are reduced to 0. Because the non-zero elements appear in the lower triangular part, that part is set as the lower triangular matrix $L^{(i)}$. The details are as follows: If $|R_{1,1}^{(0)}| \geq |R_{n,n}^{(0)}|$, we use the following strategy. The off-diagonal elements are reduced to 0 in the order of elements $(1,2)(1,3), \ldots, (1,n), (2,3), (2,4), \ldots, (n-2,n-1), (n-2,n), (n-1,n)$. Then, the off-diagonal elements in the lower triangular matrix $L^{(i)}$ are reduced to 0 in the order of elements $(2,1)(3,1), \ldots, (n,1), (3,2), (4,2), \ldots, (n-1,n-2), (n,n-2), (n,n-1)$. If $|R_{1,1}^{(0)}| < |R_{n,n}^{(0)}|$, we use the following strategy. The off-diagonal elements are reduced to 0 in the order of elements $(n-1,n)(n-2,n), \ldots, (1,n), (n-2,n-1), (n-3,n-1), \ldots, (1,3), (1,2)$. Then, the off-diagonal elements in the lower triangular matrix $L^{(i)}$ are reduced to 0 in the order of elements $(n,n-1)(n-2,n-2), \ldots, (n,1), (n-1,n-2), (n-1,n-3), \ldots, (3,1), (2,1)$. By using the two-sided Jacobi method, all the off-diagonal elements converge to 0. Computationally, since the number of iterations is limited to a finite number, the off-diagonal elements may not be an exact 0. Therefore, in case the eq.(9) is satisfied, the element is set to $R_{j,k} \leftarrow 0$.

$$|R_{j,k}| \leq \varepsilon \sqrt{|R_{j,j}| \times |R_{k,k}|}, \quad (9)$$

Once all the off-diagonal elements converge to 0, the iteration is terminated.

3.3 Implementation method using arctangent function

Unlike in the one-sided Jacobi method, singular value decomposition using the two-sided Jacobi method requires many operations to decide $c_1, s_1, c_2,$ and $s_2$. In numerical computations, performing such a large number of operations introduces numerous errors into the variables under computation. Therefore, we propose to implement the method using the arctangent function. In the proposed implementation method, the number of operations for computing $c_1, s_1, c_2,$ and $s_2$ is decreased using $\tan^{-1}, \theta_1,$ and $\theta_2$.

Here, $c_1, s_1, c_2,$ and $s_2$ are computed using $\tan^{-1}, \theta_1,$ and $\theta_2$:

$$\alpha = \tan^{-1}\left(\frac{R_{j,k}}{R_{j,j} - R_{k,k}}\right), \quad (10)$$

$$\beta = \tan^{-1}\left(-\frac{R_{j,k}}{R_{j,j} + R_{k,k}}\right), \quad (11)$$

$$\theta_1 = \frac{1}{2} (\alpha + \beta), \quad \theta_2 = \frac{1}{2} (\alpha - \beta), \quad (12)$$

$$c_1 = \cos (\theta_1), \quad s_1 = \sin (\theta_1), \quad (13)$$

$$c_2 = \cos (\theta_2), \quad s_2 = \sin (\theta_2). \quad (14)$$

Here, $-\frac{\pi}{4} \leq \theta_1 \leq \frac{\pi}{4}$ and $-\frac{\pi}{4} \leq \theta_2 \leq \frac{\pi}{4}$. Then, the computed $c_1, s_1, c_2,$ and $s_2$ are substituted in eqs.(15) and (16):

$$u = c_1 + c_2, \quad (15)$$

$$\hat{R}_{j,j} = R_{j,j} + \frac{s_2}{u} \times R_{j,k}, \quad \hat{R}_{k,k} = R_{k,k} - \frac{s_1}{u} \times R_{j,k}. \quad (16)$$

The fused multiply–accumulate can be adopted in the double underline part of the equations. It reduces the error of the final operation result by performing a product-sum operation in one instruction without rounding up the integration in the middle, and is, therefore, important to achieve high accuracy. In the case that $|\theta_0|$ is much larger than $|\theta_j|(i = 1, \ldots, q)$, the method based on $T_q = \sum_{i=1}^{q} x_i$ and $S_q = x_0 + T_q$ is suitable for computing $S_q = \sum_{i=0}^{q} x_i$. The process is adopted in the eq.(16).

3.4 Rutishauser’s implementation method

By using Rutishauser’s implementation method [14] in the two-sided Jacobi method for eigenvalue decomposition, the fused multiply–accumulate, which can achieve high accuracy, prevents mixing of errors into $c_1, s_1, c_2,$ and $s_2$. $t_1$ and $t_2$ is decided as follows:

$$h_1 = \frac{R_{j,j} - R_{k,k}}{R_{j,k}}, \quad f_1 = \sqrt{1 + (h_1)^2}, \quad t_1 = \frac{1}{h_1 \pm f_1}, \quad (17)$$

$$h_2 = \frac{R_{j,j} + R_{k,k}}{R_{j,k}}, \quad f_2 = \sqrt{1 + (h_2)^2}, \quad t_2 = \frac{1}{h_2 \pm f_2}. \quad (18)$$

Here, the signs of $f_1$ and $f_2$ have to be matched with the signs of $h_1$ and $h_2$, respectively. Then, by using $t_1$ and $t_2$,

$$v_1 = 1 - t_1 \times t_2, \quad w_1 = t_1 + t_2, \quad (19)$$

$$u_1 = \max(|v_1|, |w_1|), \quad \frac{v_1}{u_1}, \frac{w_1}{u_1} \quad (20)$$

$$c_1 = \sqrt{\frac{v_1}{u_1}^2 + \frac{w_1}{u_1}^2}, \quad s_1 = \sqrt{\frac{v_1}{u_1}^2 + \frac{w_1}{u_1}^2}, \quad (21)$$

and

$$v_2 = 1 + t_1 \times t_2, \quad w_2 = t_1 - t_2, \quad (22)$$

$$u_2 = \max(|v_2|, |w_2|), \quad \frac{v_2}{u_2}, \frac{w_2}{u_2} \quad (23)$$

$$c_2 = \sqrt{\frac{v_2}{u_2}^2 + \frac{w_2}{u_2}^2}, \quad s_2 = \sqrt{\frac{v_2}{u_2}^2 + \frac{w_2}{u_2}^2}, \quad (24)$$

are computed. Finally, the eqs.(25) and (26) can be computed using $c_1, s_1, c_2,$ and $s_2$,

$$u = c_1 + c_2, \quad (25)$$

$$\hat{R}_{j,j} = R_{j,j} + \frac{s_2}{u} \times R_{j,k}, \quad \hat{R}_{k,k} = R_{k,k} - \frac{s_1}{u} \times R_{j,k}. \quad (26)$$
Algorithm 1 Implementation of the Givens rotation
1: \( f \leftarrow |x| \)
2: \( g \leftarrow |y| \)
3: \( t \leftarrow \max(f, g) \)
4: if \( t = 0 \) then
5: \( \cos(\theta) \leftarrow 1 \)
6: \( \sin(\theta) \leftarrow 0 \)
7: \( \sqrt{x^2 + y^2} \leftarrow 0 \)
8: else
9: \( u \leftarrow f/t \)
10: \( v \leftarrow g/t \)
11: if \( f \geq g \) then
12: \( r \leftarrow \sqrt{1 + v^2} \)
13: \( \cos(\theta) \leftarrow u/r \)
14: \( \sin(\theta) \leftarrow v/r \)
15: \( \sqrt{x^2 + y^2} \leftarrow r \times t \)
16: else
17: \( r \leftarrow \sqrt{1 + u^2} \)
18: \( \cos(\theta) \leftarrow u/r \)
19: \( \sin(\theta) \leftarrow v/r \)
20: \( \sqrt{x^2 + y^2} \leftarrow r \times t \)
21: end if
22: end if

The fused multiply–accumulate can be adopted in double underlined part in the equations.

3.5 Implementation method using Givens rotation

3.5.1 Implementation of the Givens rotation

Consider the Givens rotation:

\[
\begin{align*}
\cos(\theta) &= \frac{x}{\sqrt{x^2 + y^2}}, \\
\sin(\theta) &= \frac{y}{\sqrt{x^2 + y^2}}.
\end{align*}
\tag{27}
\]

Here, for computing \( \cos(\theta) \), \( \sin(\theta) \), and \( \sqrt{x^2 + y^2} \), Algorithm 1 is adopted. To avoid overflow and underflow, the Givens rotation should be implemented as the Algorithm 1. The fused multiply-accumulate can be adopted in the double-underlined part of the lines 12 and 17 of Algorithm 1.

3.5.2 Detail of the implementation

For computing \( c_1, s_1, c_2, \) and \( s_2 \), Algorithm 2 is adopted. Here, the function \( \text{SIGN}(A, B) \) returns the value of \( A \) with the sign of \( B \). Then, the eqs. (28) and (29) are computed using \( c_1, s_1, c_2, \) and \( s_2 \).

\[
\begin{align*}
\hat{R}_{j,j} &= \hat{R}_{j,j} + \frac{s_2}{u} \times R_{j,k}, \\
\hat{R}_{k,k} &= \hat{R}_{k,k} - \frac{s_1}{u} \times R_{j,k}.
\end{align*}
\tag{28}
\]

The fused multiply-accumulate can be adopted in double underlined part in the equations.

Algorithm 2 Implementation method using the Givens rotation
1: \( h_1 \leftarrow R_{j,j} - \hat{R}_{k,k} \)
2: \( g_1 \leftarrow |R_{j,k}| \)
3: \( f_1 \leftarrow |h_1| + \sqrt{h_1^2 + R_{j,k}^2} \quad \text{The Givens rotation is adopted in underlined part} \)
4: \( g_1 \leftarrow \text{SIGN}(R_{j,k}, R_{j,k}/h_1) \)
5: \( h_2 \leftarrow R_{j,j} + \hat{R}_{k,k} \)
6: \( f_2 \leftarrow |h_2| + \sqrt{h_2^2 + R_{j,k}^2} \quad \text{The Givens rotation is adopted in underlined part} \)
7: \( g_2 \leftarrow \text{SIGN}(R_{j,k}, -R_{j,k}/h_2) \)
8: if \( f_1 \geq f_2 \) then
9: \( t_1 \leftarrow g_1/f_1 \)
10: \( \hat{c}_1 \leftarrow -t_1 \times g_2 + f_2 \)
11: \( \hat{s}_1 \leftarrow t_1 \times f_2 + g_2 \)
12: Compute \( c_1 \) and \( s_1 \) using the Givens rotation for \( x \leftarrow \hat{c}_1 \) and \( y \leftarrow \hat{s}_1 \)
13: \( \hat{c}_2 \leftarrow t_1 \times g_2 + f_2 \)
14: \( \hat{s}_2 \leftarrow t_1 \times f_2 - g_2 \)
15: Compute \( c_2 \) and \( s_2 \) using the Givens rotation for \( x \leftarrow \hat{c}_2 \) and \( y \leftarrow \hat{s}_2 \)
16: else
17: \( t_2 \leftarrow g_2/f_2 \)
18: \( \hat{c}_1 \leftarrow -g_1 \times t_2 + f_1 \)
19: \( \hat{s}_1 \leftarrow g_1 \times t_2 + g_2 \)
20: Compute \( c_1 \) and \( s_1 \) using the Givens rotation for \( x \leftarrow \hat{c}_1 \) and \( y \leftarrow \hat{s}_1 \)
21: \( \hat{c}_2 \leftarrow g_1 \times t_2 + f_1 \)
22: \( \hat{s}_2 \leftarrow f_1 \times t_2 + g_1 \)
23: Compute \( c_2 \) and \( s_2 \) using the Givens rotation for \( x \leftarrow \hat{c}_2 \) and \( y \leftarrow \hat{s}_2 \)
24: end if

3.6 Addition of sorting function to the two-sided Jacobi method

In the case \( R_{1,1}^{(0)} \geq R_{n,n}^{(0)} \), if \( \hat{R}_{j,j} < \hat{R}_{k,k} \) and \( s_1 > 0 \) are satisfied, we set \( c_1 \leftarrow s_1 \) and \( s_1 \leftarrow -c_1 \), which means \( \theta_1 \leftarrow \theta_1 - \frac{\pi}{2} \). And, if \( |\hat{R}_{j,j}| < |\hat{R}_{k,k}| \) and \( s_1 \leq 0 \) are satisfied, we set \( c_1 \leftarrow -s_1 \) and \( s_1 \leftarrow c_1 \), which means \( \theta_1 \leftarrow \theta_1 + \frac{\pi}{2} \). If \( |\hat{R}_{j,j}| < |\hat{R}_{k,k}| \) and \( s_2 > 0 \) are satisfied, we set \( c_2 \leftarrow s_2 \) and \( s_2 \leftarrow -c_2 \), which means \( \theta_2 \leftarrow \theta_2 - \frac{\pi}{2} \), and, if \( |\hat{R}_{j,j}| < |\hat{R}_{k,k}| \) and \( s_2 \leq 0 \) are satisfied, we set \( c_2 \leftarrow -s_2 \) and \( s_2 \leftarrow c_2 \), which means \( \theta_2 \leftarrow \theta_2 + \frac{\pi}{2} \). If \( \frac{\pi}{2} \) is subtracted for or is added to both \( \theta_1 \) and \( \theta_2 \), then we set \( \hat{R}_{j,j} \leftarrow \hat{R}_{k,k}, \hat{R}_{k,k} \leftarrow \hat{R}_{j,j} \). Otherwise, we set \( \hat{R}_{j,j} \leftarrow -\hat{R}_{k,k}, \hat{R}_{k,k} \leftarrow -\hat{R}_{j,j} \).

In the case \( R_{1,1}^{(0)} < R_{n,n}^{(0)} \), if \( \hat{R}_{j,j} \geq \hat{R}_{k,k} \) and \( s_1 > 0 \) are satisfied, we set \( c_1 \leftarrow s_1 \) and \( s_1 \leftarrow -c_1 \), which means \( \theta_1 \leftarrow \theta_1 + \frac{\pi}{2} \). And, if \( |\hat{R}_{j,j}| \geq |\hat{R}_{k,k}| \) and \( s_1 \leq 0 \) are satisfied, we set \( c_1 \leftarrow -s_1 \) and \( s_1 \leftarrow c_1 \), which means \( \theta_1 \leftarrow \theta_1 - \frac{\pi}{2} \). If \( \frac{\pi}{2} \) is subtracted from or is added to both \( \theta_1 \) and \( \theta_2 \), then we set \( \hat{R}_{j,j} \leftarrow \hat{R}_{k,k}, \hat{R}_{k,k} \leftarrow \hat{R}_{j,j} \). Otherwise, we set \( \hat{R}_{j,j} \leftarrow -\hat{R}_{k,k}, \hat{R}_{k,k} \leftarrow -\hat{R}_{j,j} \).
we set $c$ satisfied, we set $c$ both we can correct variable representing both of $x$ and, if $\hat{R}_{j,j} > \bar{R}_{k,k}$ and $s_2 > 0$ are satisfied, we set $c_2 \rightarrow s_2$ and $s_2 \leftarrow c_2$, which means $\theta_2 \leftarrow \theta_2 + \frac{\pi}{2}$. If $\frac{\pi}{2}$ is subtracted for or is added to both $\theta_1$ and $\theta_2$, then we set $\hat{R}_{j,j} \leftarrow \bar{R}_{k,k}, \bar{R}_{k,k} \leftarrow \hat{R}_{j,j}$. Otherwise, we set $\hat{R}_{j,j} \leftarrow -\bar{R}_{k,k}, \bar{R}_{k,k} \leftarrow \hat{R}_{j,j}$.

Furthermore, by adding the above operation, the two-sided Jacobi method becomes to has the function of sorting from larger singular values to smaller singular values. Please note that after the above operation, $c_1$ and $c_2$ are still nonnegative.

### 4. Correction of $c_1$, $s_1$, $c_2$, or $s_2$

By using the Rutishauser’s implementation method [14], we can correct $c_1$, $s_1$, $c_2$, or $s_2$. Let $\hat{c}$, $\hat{s}$, $\tilde{c}$, and $\tilde{s}$ be a variable representing both $c_1$ and $c_2$, a variable representing both $s_1$ and $s_2$, a result of correction for $c_1$ and $c_2$, and a result of correction for $s_1$ and $s_2$, respectively.

#### 4.1 False position method

Fig. 2 shows the image of the false position method. In the initial setting, $x_1$ and $x_2$ have different values. The sign of $f(x_1)$ is set to be different from that of $f(x_2)$.

In the false position method, $x_M$ in the eq.(30) is set to a new position to compute the real root $x$ in $f(x) = 0$;

$$x_M = x_1 \times f(x_2) - x_2 \times f(x_1) \over f(x_2) - f(x_1).$$

(30)

Here, in case the sign of $f(x_1)$ is equal to that of $f(x_M)$, then $x_1 \leftarrow x_M$. On the other hand, in case the sign of $f(x_2)$ is equal to that of $f(x_M)$, then $x_2 \leftarrow x_M$.

As shown in fig. 2, $x_M$ is set to a new $x_1$.

#### 4.2 Secant method

Fig. 3 shows the image of the secant method.

In the secant method, the following recurrence relation is adopted in order to compute the real root $x$ in $f(x) = 0$:

$$x_{n+1} = x_n - f(x_n) \times {x_n - x_{n-1} \over f(x_n) - f(x_{n-1})}.$$  

(31)

From the initial setting $x_0$ and $x_1$, the sequence of $x_2, x_3, \cdots$ converges to the real root $x$, as the point sequence is computed in order.

#### 4.3 Correction method

Theoretically, $\hat{c}^2 + \hat{s}^2 = 1$ is satisfied. However, computationally, the equation is not satisfied because of the effect of a rounding error. Therefore, we propose a correction method for $\hat{c}$ and $\hat{s}$.

Assuming that $\hat{s}$ is correct, $\hat{c}$ is decided by

$$x^2 + \hat{s}^2 = 1.$$  

(32)

Assuming that $\hat{c}$ is correct, $\hat{s}$ is computed using

$$\hat{c}^2 + x^2 = 1.$$  

(33)

Equations (32) and (33) can be used properly by introducing $\tilde{c} = \cos \theta$ and $\tilde{s} = \sin \theta$. For the case $-\frac{\pi}{4} \leq \theta \leq \frac{\pi}{4}$, eq.(32) is used, whereas eq.(33) is adopted for $\frac{\pi}{4} < \theta < \frac{\pi}{2}$ or $-\frac{\pi}{2} < \theta < -\frac{\pi}{4}$. In a singular value decomposition using the two-sided Jacobi method, $\hat{c} \geq 0$ is satisfied. Therefore, we can assume $-\frac{\pi}{4} \leq \theta \leq \frac{\pi}{4}$.

When the nonlinear single equation $f(x) = 0$ and the initial numbers $x_0$ and $x_1$ are given, $x_2$, which is the result at one iteration of the secant method, is exactly equal to $x_M$ achieved by the false position method:

$$x_2 = x_0 f(x_1) - x_1 f(x_0) \over f(x_1) - f(x_0).$$  

(34)

In the case $-\frac{\pi}{4} \leq \theta \leq \frac{\pi}{4}$, $\hat{c}$ is recomputed using the eq.(32). The case $-\frac{\pi}{4} \leq \theta \leq \frac{\pi}{4}$ can be considered equivalent to $\hat{c} \geq |\hat{s}|$. In order to compute $\hat{c}$, the initial numbers are set to $x_0 = 1$ and $x_1 = \hat{c}$. When $f(x) = x^2 + \hat{s}^2 - 1$,

$$\hat{c} = \frac{(\hat{c}^2 + \hat{s}^2 - 1) - \hat{c} \hat{s}^2}{(\hat{c}^2 + \hat{s}^2 - 1) - (\hat{s}^2)} = 1 - \hat{s} \times \frac{\hat{s}}{1 + \hat{c}}.$$  

(35)
is obtained, where $\hat{c}$ is more suitable for satisfying $f(x) = x^2 + s^2 - 1$. The Givens rotation for vectors $x$ and $y$ is defined as
\[
x \leftarrow \hat{c}x + \hat{s}y, \quad y \leftarrow -\hat{s}x + \hat{c}y.
\]
When not using $\hat{c}$ but $\hat{s}$,
\[
z_1 = \frac{\hat{s}}{1 + \hat{c}},
\]
\[
x \leftarrow (1 - \hat{s} \times z_1) x + \hat{s} y = \hat{s} \left(\frac{z_1 x + y}{1 + \hat{c}}\right) + x,
\]
\[
y \leftarrow -\hat{s} x + (1 - \hat{s} \times z_1) y = -\hat{s} \left(\frac{z_1 x + y}{1 + \hat{c}}\right) + y.
\]
is obtained.

The case $\frac{-\pi}{2} < \theta < \frac{-\pi}{4}$ can be regarded as being equivalent to $\frac{-\pi}{2} < |\theta|$ and $\frac{-\pi}{2} < |\xi|$. In order to compute $\hat{s}$, the initial numbers are set to $x_0 = 1$ and $x_1 = \hat{s}$. When $f(x) = x^2 + x^2 - 1$,
\[
\hat{s} = \frac{(c^2 + s^2 - 1) - \hat{s}c^2}{c^2 + s^2 - 1 - \hat{s}^2} = 1 - c \times \hat{c} \times \hat{s},
\]
is obtained, where $\hat{s}$ is more suitable for satisfying $f(x) = x^2 + x^2 - 1$. When not using $\hat{s}$ but $\hat{s}$, the Givens rotation for vectors $x$ and $y$ can be represented as follows:
\[
z_2 = \frac{\hat{c}}{1 + \hat{s}},
\]
\[
x \leftarrow \hat{c} x + (1 - \hat{c} \times z_2) y = \hat{c} \left(\frac{z_2 y + x}{1 + \hat{s}}\right) + y,
\]
\[
y \leftarrow -(1 - \hat{c} \times z_2) x + \hat{c} y = \hat{c} \left(\frac{z_2 x + y}{1 + \hat{s}}\right) - x.
\]
The case $\frac{-\pi}{4} < \theta < \frac{-\pi}{2}$ can be regarded as being equivalent to $\hat{c} < |\theta|$ and $\hat{s} < 0$. To compute $\hat{s}$, the initial numbers are set to $x_0 = -1$ and $x_1 = \hat{s}$. When $f(x) = x^2 + x^2 - 1$,
\[
\hat{s} = \frac{(c^2 + s^2 - 1) - \hat{s}c^2}{c^2 + s^2 - 1 - \hat{s}^2} = -1 + c \times \hat{c} \times \hat{s},
\]
is obtained, where $\hat{s}$ is more suitable for satisfying $f(x) = x^2 + x^2 - 1$. When not using $\hat{s}$ but $\hat{s}$, the Givens rotation for vectors $x$ and $y$ can be represented as follows:
\[
z_3 = \frac{\hat{c}}{1 - \hat{s}},
\]
\[
x \leftarrow \hat{c} x + (-1 + \hat{c} \times z_3) y = \hat{c} \left(\frac{z_3 y + x}{1 - \hat{s}}\right) - y,
\]
\[
y \leftarrow -(-1 + \hat{c} \times z_3) x + \hat{c} y = \hat{c} \left(\frac{z_3 x + y}{1 - \hat{s}}\right) + x.
\]

Please note that the fused multiply–accumulate can be adopted in the double underlined part.

### Table 1: Experimental Environment

<table>
<thead>
<tr>
<th>CPU</th>
<th>Intel(R) Xeon(R) Silver 4116 @ 2.10GHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM</td>
<td>192 GB</td>
</tr>
<tr>
<td>OS</td>
<td>Ubuntu 18.04 LTS</td>
</tr>
<tr>
<td>Compiler</td>
<td>gfortran 7.4.0</td>
</tr>
<tr>
<td>Options</td>
<td>-O3 -mtune=native -march=native</td>
</tr>
<tr>
<td>Software</td>
<td>Lapack 3.8.0</td>
</tr>
<tr>
<td>Precision</td>
<td>single precision</td>
</tr>
</tbody>
</table>

### 5. Experiments

We checked if the two-sided Jacobi method (arc tan and Rutishauser versions) has shorter computation time and higher accuracy than the one-sided Jacobi method implemented in LAPACK [11] for small matrices. Table 1 shows the experimental environment. We used eight matrices for the comparison:

- $A_1$ (dimension size: $500 \times 500$, an upper triangular matrix)
- $A_2$ (dimension size: $1000 \times 1000$, an upper triangular matrix)
- $A_3$ (dimension size: $1500 \times 1500$, an upper triangular matrix)
- $A_4$ (dimension size: $2000 \times 2000$, an upper triangular matrix)
- $A_5$ (dimension size: $500 \times 500$, an upper triangular matrix)
- $A_6$ (dimension size: $1000 \times 1000$, an upper triangular matrix)
- $A_7$ (dimension size: $1500 \times 1500$, an upper triangular matrix)
- $A_8$ (dimension size: $2000 \times 2000$, an upper triangular matrix)

In $A_1$, $A_2$, $A_3$, and $A_4$, all elements are set to random numbers in $[0, 1]$ generated by a uniform random number generator. In $A_5$, $A_6$, $A_7$, and $A_8$, all elements are set to 1.

The two-sided Jacobi method (arc tan and Rutishauser versions) has shorter computation time and higher accuracy than the one-sided Jacobi method implemented in LAPACK [11] for the $500 \times 500$, $1000 \times 1000$, and $1500 \times 1500$ upper triangular matrices with elements generated by a uniform random number generator and also for the $500 \times 500$ and $1000 \times 1000$ upper triangular matrix in which all elements are 1. The performance results are given in Tables 2 and 3.

### 6. Conclusion

We confirmed through experiments that the two-sided Jacobi method (arc tan and Rutishauser versions) has shorter computation time and higher accuracy than the one-sided Jacobi method for smaller matrices.

For future work, we plan to apply our two-sided Jacobi Method to implement the modified Sakurai-Sugiura method [9].
### Table 2: Comparison of Jacobi SVD Algorithms(I)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>One-sided Jacobi</th>
<th>Two-sided Jacobi</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[U^T U - I]_F$</td>
<td>$1.94 \times 10^{-4}$</td>
<td>$4.34 \times 10^{-9}$</td>
</tr>
<tr>
<td>$[V^T V - I]_F$</td>
<td>$8.20 \times 10^{-5}$</td>
<td>$4.32 \times 10^{-5}$</td>
</tr>
<tr>
<td>$[A - UΣV^\top]_F$</td>
<td>$8.13 \times 10^{-4}$</td>
<td>$3.73 \times 10^{-4}$</td>
</tr>
<tr>
<td>Computation time [s]</td>
<td>1.777</td>
<td>1.185</td>
</tr>
</tbody>
</table>

### Table 3: Comparison of Jacobi SVD Algorithms(II)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Two-sided Jacobi</th>
<th>Two-sided Jacobi</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[U^T U - I]_F$</td>
<td>$4.33 \times 10^{-9}$</td>
<td>$4.34 \times 10^{-9}$</td>
</tr>
<tr>
<td>$[V^T V - I]_F$</td>
<td>$4.31 \times 10^{-5}$</td>
<td>$4.33 \times 10^{-5}$</td>
</tr>
<tr>
<td>$[A - UΣV^\top]_F$</td>
<td>$3.77 \times 10^{-4}$</td>
<td>$3.60 \times 10^{-4}$</td>
</tr>
<tr>
<td>Computation time [s]</td>
<td>1.134</td>
<td>1.126</td>
</tr>
</tbody>
</table>

### Acknowledgment

This work was supported by JSPS KAKENHI Grant Number JP17H02858 and JP17K00167.

### References


