Doctoral Dissertation

Orthogonal transformation based algorithms for singular value decomposition

Guidance

Professor Yoshimasa NAKAMURA

Sho ARAKI

Department of Applied Mathematics and Physics

Graduate School of Informatics

Kyoto University

February 2021

Abstract

Fast and accurate computation of matrix singular value decomposition is a major problem in the computer and computational sciences. The orthogonal transformation based algorithms, the OQDS algorithm and the Jacobi algorithm, attract attention as robust singular value decomposition algorithms. We introduce some applications for each algorithm, and propose new implementations of the OQDS algorithm and the Jacobi algorithms.

We discuss two applications and their implementations of the OQDS algorithm. One is the narrow band reduction approach and the other one is a combination method with the DQDS algorithm for obtaining column vectors of bidiagonal matrices. The narrow band reduction intends to improve the cache efficiency of the preliminary reduction. We formulate the OQDS algorithm for lower tridiagonal matrices and design a shift strategy and convergence criteria in order to compute the lower tridiagonal matrix produced as a result of the preliminary reduction. The combination method of the OQDS algorithm and the DQDS algorithm for obtaining column space of bidiagonal matrices is another application of the OQDS algorithm. We formulate the OQDS algorithm for computing singular vectors of bidiagonal matrices corresponding to the singular values obtained by the DQDS algorithm.

We also discuss about an implementation of the Jacobi algorithm as another orthogonal transformation based algorithm. An appropriate formulation of the Givens transformation with the fused multiply-add (FMA) instruction improve the speed and the accuracy both of the two-sided and the one-sided Jacobi algorithms. In addition, a selection of the rotation angle of the Givens rotation based on the knowledge of the OQDS algorithm gives a sorting feature of singular values to the Jacobi algorithm.

Through the results of numerical experiments, we show some advantages of the orthogonal transformation based algorithms, the OQDS algorithm and the Jacobi algorithm, in terms of the speed and the accuracy for the applications and consider an intrinsic relationship between the OQDS algorithm and the Jacobi algorithm.

Contents

1 Introduction

Fast and accurate computation of matrix singular value decomposition is a major problem in computer and computational science. For various types of problems, various methods have been proposed. Especially, if we want to obtain both of singular values and singular vectors with single thread computation, the QR algorithm[1] is the most famous and successful solver and a fine tuned implementation of the algorithm is provided by LAPACK[2]. On the other hand, there are some disadvantages of the QR algorithm reported such as inaccuracy for ill-conditioned matrices and cache inefficiency of the pre-process of the algorithm[3, 4].

In this thesis, we focus on the following two orthogonal transformations based solvers for singular value decomposition, the orthogonal quotient-difference algorithm with shift (OQDS algorithm)[5] and the Jacobi algorithm[6].

In order to overcome the inaccuracy of the QR algorithm for an ill-conditioned matrix, the quotient-difference algorithm (QD algorithm) by Rutishauser[7, 8] was reevaluated by Parlett and his coworkers[9, 10] in 1990'. The QD algorithm performs more accurate singular value computation for the ill-conditioned matrix than the QR algorithm. Differential quotient-difference algorithm with shift (DQDS algorithm)[10] is a practical implementation of the QD algorithm, which introduces suitable formulation for numerical computation and acceleration with an origin shift. Although the DQDS algorithm is widely used as the subroutine of the singular value computation solver `xBDSQR` provided in LAPACK, there is a disadvantage related to the preliminary reduction in common with the QR algorithm. In the `xBDSQR` routine, which uses the QR algorithm or the DQDS algorithm as its subroutines depending on whether the user requires the singular vectors or not, the input is bidiagonal matrix which is preliminary reduced from the original general real matrix with the Householder transformation. The QR algorithm and the DQDS algorithm perform fast and accurate singular value decomposition for the bidiagonal matrix however, the preliminary reduction has a higher computational complexity than that of the algorithms and therefore accounts the majority of the total computation time of singular value decomposition of general matrices. Householder bidiagonalization is often used as the preliminary reduction and its conventional implementation consists of an iteration of BLAS level 2 `xGEMV` (vector-matrix multiplication) routine. However, `xGEMV` routine requires data transfer from the main memory to cache for every vector multiplications and the latency is a strong bottleneck of the Householder bidiagonalization.

To reduce the data transfer, the narrow band reduction approach by using BLAS level 2.5 (two vectors-matrix multiplication) is proposed[11]. In the iteration of the Householder reductions, we can approximately cut in half of the data transfer by performing `xGEMM` (matrix-matrix multiplication) routine for two each vectors instead of `xGEMV` for each vectors. In this case, we obtain a lower tridiagonal matrix as the result of the reduction. Consequently, if we can compute the singular values of the lower tridiagonal matrix within a reasonable cost, it is expected that the total computation time for computing singular values of the original general real matrix is shortened.

The OQDS algorithm[5] is a variant of the QD algorithm and has capability for computing singular values and vectors of a lower tridiagonal matrix. The OQDS algorithm fully consists of orthogonal transformations and therefore is expected to perform accurate computation, however, there has been no practical implementation of the OQDS algorithm.

In this thesis, we propose a practical implementation of the OQDS algorithm for the lower tridiagonal matrix. For the practical implementation, we propose a formulation of Givens transformation and generalized Givens transformation, both of them are orthogonal transformations, which are the main components of the OQDS algorithm with considering loss of trailing digits and cancellation of significant digits. In addition, we propose a shift strategy which accelerates the convergence of the OQDS algorithm with estimating the minimum value of the singular values of the lower tridiagonal matrix. In order to estimate the minimum singular value, we propose a method for obtaining the trace of the inverse of a pentadiagonal matrix and then, compute the value of the generalized Newton shift of the lower tridiagonal matrix with the trace value. After sharpening the shift value with the Kato-Temple shift[12], we adopt the shift value to our shift strategy as an algebraic shift. Our shift strategy also contains the Gerschgorin shift[13] and the Laguerre[14] shift which are used in the DQDS algorithm. Then, we design convergence criteria based on Weyl's monotonicity theorem[15, 16] and apply the criteria for performing the deflation and splitting operations which terminate the OQDS algorithm. We perform some numerical experiments to confirm the implementation of the OQDS algorithm computes singular values of the lower tridiagonal matrices within a reasonable cost and reduces the total computation time of singular values computation of general real matrices.

We also focus on the capability of the OQDS algorithm for computing singular vectors likewise the QR algorithm. The orthogonal transformations consists of the OQDS algorithm keep column and row spaces of the original matrix so that the singular vectors are spontaneously obtained if we record the sequence of the transformations. However, for the lower tridiagonal matrix, the OQDS algorithm requires more number of iterations than that for the bidiagonal matrix because there are more off-diagonal entries to be converged through the iteration. The large number of iterations does not only slow down the convergence speed but also worsen the accuracy since the accumulation of the rounding error inevitable for numerical computation becomes large. The nonnegative off-diagonal entries also harden the estimation of the lower bound of the singular values which is required for the shift strategy and limit the effect. As a result, it is not fair to compare the accuracy of the singular values and vectors obtained by the OQDS algorithm for lower tridiagonal matrix to that even by the QR algorithm for bidiagonal matrix.

Then, we let the target matrix back to the bidiagonal matrix, propose a column space computation method for bidiagonal matrix with a combination of the OQDS algorithm and the DQDS algorithm. The application of the column space computation method is narrower than that of the general singular value decomposition but the Sakurai-Sugiura method[17], a parallel solver for polynomial eigenvalue problem, utilizes the column space

computation method as its subroutine therefore, it is expected that we can compute the non-linear eigenvalues and eigenvectors faster and more accurate if the combination method achieves a higher performance than the original OQDS algorithm. The main idea of the proposed combination method is that we compute the singular values of given bidiagonal matrix with the fast and accurate DQDS algorithm and then compute column vectors with the OQDS algorithm base on the obtained singular values. In order to implement this idea, we propose a new formulation of the OQDS algorithm for bidiagonal matrix accelerated with externally obtained singular values. The formulation also intends to suppress numerical errors. We also propose shift strategies for both of the OQDS algorithm and the DQDS algorithm. As the shift strategy of the DQDS algorithm, we adopt the generalized Rutishauser bound[18], the Johnson bound[19] and the Collatz inequality[20] based bound for estimating method of the minimum of the singular values in addition to the previously introduced method in the OQDS algorithm for the lower tridiagonal matrix. On the other hand, we also adopt the generalized Rutishauser bound and the Collatz inequality based bound which are not suitable for the lower tridiagonal matrix to the OQDS algorithm for bidiagonal matrix. Then, we perform a numerical experiment to confirm the effectiveness of the combination method of the DQDS algorithm and the OQDS algorithm.

Our proposed applications of the OQDS algorithm perform fast computation for each target problems. On the other hand, a high accuracy realized by the fully orthogonal transformation based formulation, which is the primary advantage of the OQDS algorithm, does not contribute to the accuracy of the entire algorithm because the preliminary Householder reduction may produce numerical error before applying the OQDS algorithm. Because of the reduction error, the accuracy of the entire algorithm hits the ceiling in accuracy no matter how the OQDS algorithm achieves. Then, we seek an application of the OQDS algorithm independent of the preliminary reduction. In fact, the OQDS algorithm for a general triangular matrix can be interpreted as a variant of Jacobi algorithm. Based on the interpretation, we try to improve the Jacobi algorithm with the knowledge we have acquired through the studies about the OQDS algorithm.

The Jacobi algorithm is a classical solver of singular value decomposition problem proposed by Jacobi before the growth of the electrical computer. In this thesis, we discuss two implementations of the Jacobi algorithm. One is the one-sided Jacobi algorithm and the other is the two-sided Jacobi algorithm. The two-sided Jacobi algorithm is a prototype of the Jacobi algorithm and is straightforwardly formulated as applying the Givens transformation from both side of the given general triangular matrix.

We propose an accurate computation method of the Givens transformation using fused multiply-add (FMA) instruction[21] in the formulation. The false position method[22] and the secant method[23] are used to correct the rotation angle of the Givens transformation. In addition to the accurate implementation method of the Givens transformation, we integrate a sorting feature of the singular values of the result matrix into the two-sided Jacobi algorithm. By introducing a selection method of the rotation angle of the Givens transformation based on the knowledge we have acquired through the studies of the OQDS algorithm, the Jacobi algorithm sorts singular values in descend-

ing order as the OQDS algorithm does while the conventional implementation generally gives singular values of the matrix in an irregular order. The selection of the rotation angle of the Givens transformation also accelerates the convergence of the two-sided Jacobi algorithm. As a result of these improvements, the numerical experiment shows that the two-sided Jacobi algorithm gets a higher performance than the conventional implementation in terms both of speed and accuracy.

According to the success of the two-sided Jacobi algorithm, we introduce the same improvements to the one-sided Jacobi algorithm. The one-sided Jacobi algorithm is more famous than the two-sided Jacobi algorithm, and and has been actively discussed not only in serial computation environment but in parallel computation environment[24]. The main difference from the two-sided Jacobi algorithm is multiplying the Givens rotation from only one side of the matrix as the name represents.

In addition to the accurate computation method of the Givens transformation which is introduced to the two-sided Jacobi algorithm, we propose an accurate computation method of vector norm used in the generation of Jacobi pairs which represent the iteration of the one-sided Jacobi algorithm. Then, our proposed implementation of the one-sided Jacobi algorithm shows a higher performance than the conventional implementation reported in [25] in a numerical experiment.

This thesis is organized as follows. Section 2 introduces the basic theory of the singular value decomposition and its representative application. In Section 3, we introduce the OQDS algorithm for lower tridiagonal matrix and its implementation techniques. A numerical result of the OQDS algorithm is also reported in this section. Section 4 expatiates upon the combination method of the OQDS algorithm and the DQDS algorithm for computing column space and demonstrates a numerical result of the method. Sections 5 and 6 describes the implementation details of the two-sided Jacobi algorithm and the one-sided Jacobi algorithm respectively. These sections also include numerical results of the algorithms. Then, the last section concludes whole of our studies.

2 Singular value decomposition

Singular value decomposition is one of fundamental factorizations in the domain of linear algebra. For a real or complex $m \times n$ matrix A, singular value decomposition is formulated as follows:

$$A = U\Sigma V^* \tag{1}$$

with $m \times m$ unitary matrix U, $m \times n$ rectangular diagonal matrix with non-negative real diagonal entries Σ and $n \times n$ unitary matrix V. If A is real matrix, U and V are real orthogonal matrices and satisfies $V^* = V^\top$. The column vectors of U are called left singular vectors and the column vectors of V are called right singular vectors as well. Though this form of factorization is not unique in general, Σ can be uniquely determined for A by sorting Σ_{ii} in descending order. U and V are not unique even in the case because the column vectors corresponding to multiple roots $\Sigma_{ii} = \Sigma_{jj}$ are exchangeable.

When the matrix A is factorized as (1),

$$AA^* = U\Sigma V^* V\Sigma^* U^* = U(\Sigma\Sigma^*)U^* \tag{2}$$
$$A^*A = V\Sigma^* U^* U\Sigma V^* = V(\Sigma^*\Sigma)V^* \tag{3}$$

are hold. Each (2) and (3) is the form of eigenvalue decomposition of positive semi-definite matrix AA^* and A^*A with the U and V composed of the eigenvectors of AA^* and A^*A, respectively. Singular value decomposition can be interpreted as a generalization of eigenvalue decomposition in this sense.

Singular value decomposition is applied to many practical problems such as signal processing, low-rank matrix approximation and data mining. For example, let Σ' be an $n' \times n'(n' < m, n)$ matrix which picks the elements σ_{11} to $\sigma_{n'n'}$ from an $m \times n$ matrix Σ then,

$$B = U'\Sigma' V^{*\prime}$$

gives an approximation of the matrix A.

3 Narrow Band Reduction Approach for Singular Value Decomposition with the OQDS Algorithm for Lower Tridiagonal Matrices

The QR algorithm and its related algorithms for singular value decomposition are rarely performed above general matrices but often bidiagonal matrices which are reduced from the given general matrices because the iterations of the algorithms have too high computational complexity to be applied to general dense matrices. In addition, the number of the iterations is larger than that of the reduced bidiagonal matrices and worsen accuracy because of cumulative effect of rounding error. Therefore, it is popular that the general matrix given is reduced into bidiagonal matrix and then, apply QR algorithm for the bidiagonal matrix. The two step computation makes it possible not only to reduce the number of the iteration but to enhance the precision of estimating lower bound of the singular values which is required to the shift method introduced the following subsection. On the other hand, bidiagonalization has a higher computational complexity than the QR algorithm and makes the computation time dominant among the whole of the computation. A narrow band reduction approach is an attempt to reduce the computation time of the dominant part[26]. By introducing the narrow band reduction instead of bidiagonalization, the reduction part is implemented as multiplication of matrix and vector which receives benefit of cache. In this approach, the latter part of the computation requires singular solver for narrow band matrix. We introduce the OQDS algorithm as the singular solver part and discuss the implementation details of the OQDS algorithm in this section.

For simplicity, it is noted that all matrix appears in this section are supposed to be real and input matrix is reduced into $n \times n$ square matrix.

3.1 OQDS algorithm

Classical OQDS algorithm is proposed by von Matt[5] as a modified and specialized form of the implicit Cholesky-LR algorithm for bidiagonal matrices. In this subsection, we explain the behavior and defect of the Cholesky-LR algorithm for lower triangular matrix.

Let

$$L = \begin{bmatrix} \alpha_1 & 0 & \dots & \dots & 0 \\ \beta_1 & \alpha_2 & \ddots & & \vdots \\ \gamma_1 & \beta_2 & \alpha_3 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ & \dots & \gamma_{n-2} & \beta_{n-1} & \alpha_n \end{bmatrix} \tag{4}$$

be an $n \times n$ lower triangular matrix. One step of the Cholesky-LR algorithm [27] with the shift τ^2 transforms the lower triangular matrix L into the upper triangular matrix U by

$$L^\top L - \tau^2 I = U^\top U. \tag{5}$$

Then, we set $L' := U^\top$. In this operation, the value of τ must be smaller than the minimum singular value of L to keep the positive definiteness of $L^\top L - \sigma^2 I$. It is not obvious how to obtain τ, but we explain the estimation method of the value in the following subsection. This procedure is called Cholesky-LR transformation.

By repeating the Cholesky-LR transformation iteratively, the diagonal elements of the matrix L converge to the singular values of the matrix L and the non-diagonal elements get into zero. The method for computing singular value by this iteration is called the Cholesky-LR algorithm.

However, it is known that the Cholesky decomposition shown in the algorithm is numerically unstable; the Cholesky decomposition may collapse by zero divide regardless of the shift value τ. For resolving this problem, we introduce the *implicit* Cholesky decomposition [5]. The implicit Cholesky decomposition is designed by using the generalized Givens transformation which can be performed numerically stably. The OQDS algorithm is formulated as the iteration of the implicit Cholesky-LR transformations and only composed of the orthogonal transformations what its name implies.

3.1.1 Implicit Cholesky decomposition

The implicit Cholesky decomposition computes an upper triangular matrix U from L and τ by an orthogonal transformation

$$Q \begin{bmatrix} L \\ 0 \end{bmatrix} = \begin{bmatrix} U \\ \tau I \end{bmatrix}, \tag{6}$$

where Q is a $2n \times 2n$ orthogonal matrix. It is readily verified that, for the same L and τ, the same U is obtained by (6) as by the Cholesky-LR transformation (5). The

orthogonal matrix Q is given by superposition of the Givens and the generalized Givens transformations on $\mathbb{R}^2$.

Definition 3.1 (Generalized Givens transformation [5])**.** *Let τ be a real number. The transformation on $\mathbb{R}^2$*

$$G \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} r \\ \tau \end{bmatrix} \tag{7}$$

by a 2×2 orthogonal matrix G is called the generalized Givens transformation if

$$r = \pm\sqrt{x_1^2 + x_2^2 - \tau^2}$$

and

$$\sigma^2 < x_1^2 + x_2^2.$$

Such a matrix G is uniquely determined by

$$G = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}, \tag{8}$$

$$\begin{bmatrix} c \\ s \end{bmatrix} = \frac{1}{x_1^2 + x_2^2} \begin{bmatrix} x_1 & x_2 \\ x_2 & -x_1 \end{bmatrix} \begin{bmatrix} r \\ \tau \end{bmatrix}. \tag{9}$$

It should be noted that the generalized Givens transformation is equal to the ordinary Givens transformation if $\tau = 0$. The procedure of the generalized Givens transformation is shown in Algorithm 1. The first step of the implicit Cholesky decomposition is a series of three orthogonal transformations:

$$G_1 \begin{bmatrix} \alpha_1 & & & \\ \beta_1 & \alpha_2 & & \\ \gamma_1 & \beta_2 & \ddots & \\ & \gamma_2 & \ddots & \\ & & \ddots & \\ 0 & & & \\ & 0 & & \\ & & \ddots & \\ & & & 0 \end{bmatrix} = \begin{bmatrix} \tilde{\alpha}_1 & & & \\ \beta_1 & \alpha_2 & & \\ \gamma_1 & \beta_2 & \ddots & \\ & \gamma_2 & \ddots & \\ & & \ddots & \\ \tau & & & \\ & 0 & & \\ & & \ddots & \\ & & & 0 \end{bmatrix}, \tag{10}$$

Algorithm 1 Generalized Givens transformation (rotg2(x_1, x_2, τ, c, s))

$scale := \max(|x_1|, |x_2|)$
if $scale = 0$ **then**
  $c := 1$
  $s := 0$
else
  $x_1 := x_1/scale$
  $x_2 := x_2/scale$
  $t := \tau/scale$
  $norm2 := x_1^2 + x_2^2$
  $r := \sqrt{norm2 - t^2}$
  $c := (x_1 \times r + x_2 \times t)/norm2$
  $s := (x_2 \times r - x_1 \times t)/norm2$
  $x_1 := scale \times r$
  $x_2 := \tau$
end if

$$G_2 \begin{bmatrix} \tilde{\alpha}_1 & & & \\ \beta_1 & \alpha_2 & & \\ \gamma_1 & \beta_2 & \ddots & \\ & \gamma_2 & \ddots & \\ & & \ddots & \\ \tau & & & \\ & 0 & & \\ & & \ddots & \\ & & & 0 \end{bmatrix} = \begin{bmatrix} \tilde{\tilde{\alpha}}_1 & \tilde{\beta}_1 & 0 & \\ 0 & \tilde{\alpha}_2 & & \\ \gamma_1 & \beta_2 & \ddots & \\ & \gamma_2 & \ddots & \\ & & \ddots & \\ \tau & & & \\ & 0 & & \\ & & \ddots & \\ & & & 0 \end{bmatrix}, \tag{11}$$

and then

$$G_n \begin{bmatrix} \tilde{\tilde{\alpha}}_1 & \tilde{\beta}_1 & 0 & \\ 0 & \tilde{\alpha}_2 & & \\ \vdots & \tilde{\beta}_2 & \ddots & \\ 0 & \tilde{\gamma}_2 & \ddots & \\ * & & \ddots & \\ \tau & & & \\ & 0 & & \\ & & \ddots & \\ & & & 0 \end{bmatrix} = \begin{bmatrix} \breve{\alpha}_1 & \breve{\beta}_1 & \breve{\gamma}_1 & \\ 0 & \tilde{\alpha}_2 & & \\ \vdots & \tilde{\beta}_2 & \ddots & \\ & \tilde{\gamma}_2 & \ddots & \\ 0 & & \ddots & \\ \tau & & & \\ & 0 & & \\ & & \ddots & \\ & & & 0 \end{bmatrix}. \tag{12}$$

In the first column, the first transformation (10) by G_1 generates the lower diagonal element τ. From the second (11) to the nth (12) by $G_2, \dots G_n$ vanish $\beta_1, \gamma_1, \dots$, respectively. Here, G_1 is the generalized Givens transformation for the first and the $(n+1)$th rows and $G_2, \dots, G_n$ are ordinary Givens transformations for the first and the second rows, the first and the third rows, respectively. After the $G_1, G_2, \dots, G_n$ transformations, we obtain the first column of the matrix on the right hand side of equation (6). Applying similar operations for second to nth columns successively, we obtain the upper triangular matrix U in the equation (6) and let the next L be $U^\top$ to continue the algorithm. The operations are numerically stable because all operations are formulated as the generalized Givens transformation.

Algorithm 2 summarizes the above procedures. The subroutines "rotg" and "rot" appeared in the Algorithm 2 are basic BLAS routines. If we call the "rotg(x_1, x_2, c, s)", the rotation angle of the Givens transformation is stored in the arguments c and s with cos and sin forms. The subroutine "rot(x_1, x_2, c, s)" applies the Givens transformation defined by the arguments c and s.

Algorithm 2 Implicit Cholesky Decomposition for an $n \times n$ lower triangular matrix L (icds(L))

$U := 0$
for $i = 1$ to n **do**
  $\check{\alpha}_i := \alpha_i$
  rotg2$(\check{\alpha}_i, 0, \tau, c, s)$

  *eliminate subdiagonal element
  rotg$(\check{\alpha}_i, \beta_i, c, s)$
  rot$(\check{\beta}_i, \beta_i, c, s)$

  $\vdots$

  *eliminate nth non-diagonal element
  rotg$(\check{\alpha}_i, \gamma_i, c, s)$
  rot$(\check{\beta}_i, \beta_i, c, s)$
  $\vdots$

end for
return $\check{L}$

3.2 Improvements of cash efficiency of Householder reduction

The QR algorithm is successful solver for singular value problem of general matrices but requires bidiagonalization before application. The bidiagonalization part has not only high computational complexity but also cache inefficiency because of its formulation

mainly using the multiplication among vector and vector. For this reason, we reduce the dense matrices into tridiagonal matrices which is formulated as the iterations of cache-efficient matrix-matrix multiplication. Figure 1 shows the flow of representative `xGESVD` singular solver routine using Householder bidiagonalization and QR algorithm in LAPACK [2]. In the next subsection, we introduce the well-known basic Householder

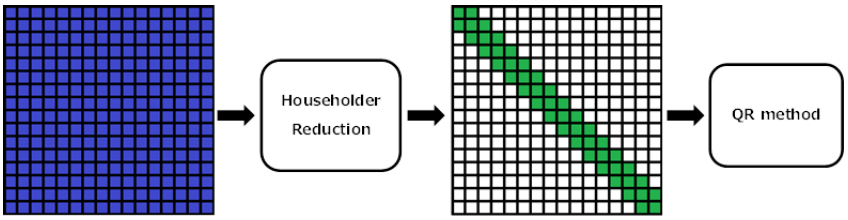


Figure 1: Flow of `xGESVD` singularsolver for matrix.

tridiagonalization method. The tridiagonalization method is also introduced as the bidiagonalization step of singular solver of positive definite matrix.

3.2.1 Basic Householder method

Algorithm 3 shows the basic tridiagonalization method [1, 28] using the Householder transformation. It should be noted that SIGN(x) means a sign of x. Like the following figure 2, the vector $d^{(k)}$ shown in this algorithm consists of $N-k$ elements located in $k+1$ to N-th row in k-th column and let $C^{(k)}$ be a submatrix which has $(k+1,k+1)$th element as upper left corner. In the computation for k-th step, we generate a reflection

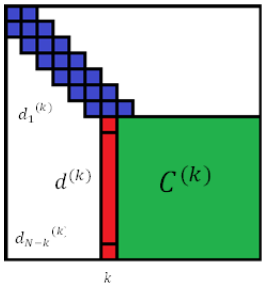


Figure 2: Matrix at the k-th step of the Householder method.

operator vector $\mathbf{u}^{(k)}$ and compute $\mathbf{p}^{(k)}$ and $\mathbf{q}^{(k)}$ by multiplying $C^{(k)}$ and $\mathbf{u}^{(k)}$. Then, we update $C^{(k)}$ by using $\mathbf{u}^{(k)}$ and $\mathbf{q}^{(k)}$. This update is called "rank-2 update" since this operation is adding two rank 1 matrices to $C^{(k)}$. This k-th step operation corresponds to the following Householder similarity transformation

$$C^{(k)} := (H^{(k)})^{-1} C^{(k)} H^{(k)}$$

where

$$H^{(k)} = I - \alpha^{(k)}\mathbf{u}^{(k)}{\mathbf{u}^{(k)}}^{\top}.$$

In the Householder method, matrix-vector multiplications and rank-2 updates cost $\frac{2}{3}N^3$ operations and occupy almost all operations in this method. However, this method is cache inefficient because these operations require $O(n^2)$ memory access for $O(n^2)$ operations if the size of $C^{(k)}$ is n.

Algorithm 3 Householder method

for $k = 1, N-2$ **do**

$\sigma^{(k)} = \sqrt{{\mathbf{d}^{(k)}}^{\top}\mathbf{d}^{(k)}}$

$\mathbf{u}^{(k)} = \left(d_1^{(k)} - \mathrm{SIGN}(d_1^{(k)})\sigma^{(k)}, d_2^{(k)}, \ldots, d_{N-k}^{(k)}\right)$

$\alpha^{(k)} = 2/\left\|\mathbf{u}^{(k)}\right\|_2$

$\mathbf{p}^{(k)} = \alpha^{(k)}C^{(k)}\mathbf{u}^{(k)}$

$\beta^{(k)} = \alpha^{(k)}{\mathbf{u}^{(k)}}^{\top}\mathbf{p}^{(k)}/2$

$\mathbf{q}^{(k)} = \mathbf{p}^{(k)} - \beta^{(k)}\mathbf{u}^{(k)}$

$C^{(k)} := C^{(k)} - \mathbf{u}^{(k)}{\mathbf{q}^{(k)}}^{\top} - \mathbf{q}^{(k)}{\mathbf{u}^{(k)}}^{\top}$

end for

3.2.2 Dongarra's algorithm

To resolve the cache inefficiency in the rank-2 update, Dongarra proposes multi-step algorithm in [3, 4]. Algorithm 4 shows this algorithm. In this algorithm, $(X)_i$ means the vector which is composed of i-th column of matrix X and $[A|B]$ means a matrix formed by putting matrix A and B side by side. For easy explanation, we assume that N is a multiple of L.

We don't apply the rank-2 updates for each step but the rank-2L update which all at once operates L step similarity transformation

$$C^{(K\times L)} := \left(H^{(K\times L)}\right)^{-1}\ldots\left(H^{((K-1)\times L+1)}\right)^{-1} \times C^{((K-1)\times L)} \times H^{((K-1)\times L+1)}\ldots H^{(K\times L)}.$$

This operation is one of matrix operator which has high rate of cache reuse. However, there are still many matrix-vector multiplications in this algorithm which has low cache reuse rate. Narrow-band reduction approach proposed by T. Imamura is one of solutions for this problem.

3.2.3 Narrow band reduction approach

To improve the cache efficiency caused by matrix-vector multiplication, T. Imamura applies a BLAS Level 2.5 (L2.5) matrix-narrow band matrix multiplication method instead

Algorithm 4 Dongarra's algorithm

for $K = 1, N/L$ **do**
  $U^{((K-1)\times L)} = \phi, Q^{((K-1)\times L)} = \phi$
  for $k = (K-1)\times L + 1, K \times L$ **do**
    $\mathbf{d}^{(k)} := \mathbf{d}^{(k)} - U^{(k-1)}(Q^{(k-1)^\top})_{k-(K-1)\times L} - Q^{(k-1)}(U^{(k-1)^\top})_{k-(K-1)\times L}$
    $\sigma^{(k)} = \sqrt{\mathbf{d}^{(k)^\top}\mathbf{d}^{(k)}}$
    $\mathbf{u}^{(k)} = \left(d_1^{(k)} - \mathrm{SIGN}(d_1^{(k)})\sigma^{(k)}, d_2^{(k)}, \ldots, d_{N-k}^{(k)}\right)$
    $\alpha^{(k)} = 2/\left\|\mathbf{u}^{(k)}\right\|_2$
    $\mathbf{p}^{(k)} = \alpha^{(k)}\left(C^{(k)} - U^{(k-1)}Q^{(k-1)^\top} - Q^{(k-1)}U^{(k-1)^\top}\right)\mathbf{u}^{(k)}$
    $\beta^{(k)} = \alpha^{(k)}\mathbf{u}^{(k)^\top}\mathbf{p}^{(k)}/2$
    $\mathbf{q}^{(k)} = \mathbf{p}^{(k)} - \beta^{(k)}\mathbf{u}^{(k)}$
    $U^{(k)} = \left[U^{(k-1)}|\mathbf{u}^{(k)}\right]$
    $Q^{(k)} = \left[Q^{(k-1)}|\mathbf{q}^{(k)}\right]$
  end for
  $C^{(k)} := C^{(K-1)\times L} - U^{(K\times L)}Q^{(K\times L)^\top} - Q^{(K\times L)}U^{(K\times L)^\top}$
end for

of BLAS Level 2 (L2) matrix-vector multiplication. Algorithm 5 shows narrow band reduction algorithm with BLAS L2.5. This algorithm is originally proposed as first step of multi-step tridiagonalization method by Bischof, and Wu, et al.[29, 30].

The algorithm is composed of the computation for $K = 1$ to $K = N/L - 1$ step. Figure 3 shows the definition of block vector $D^{(K)}$ and matrix $C^{(K)}$ in the K-th step. The

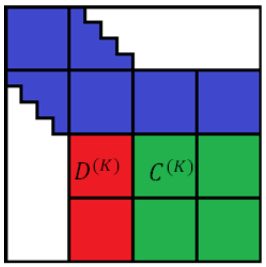


Figure 3: Matrix at the K-th step of the narrow band reduction method.

flow of each step is similar to the classical Householder method which is introduced in the former subsection but the vectors in the Householder method are exchanged to block vectors (or matrices of L bandwidth) and scalars are exchanged to $L \times L$ matrices. For K-th step processing, we first look at the block vector $D^{(K)}$ and obtain a block Householder reflector $I - U^{(K)}\alpha^{(K)}U^{(K)^T}$ which transforms $D^{(K)}$ to a block vector which has an upper triangular matrix as the first block and zero matrices as the latter matrices as shown as figure block. The $U^{(K)}$ and $\alpha^{(K)}$ are obtained easily by QR decomposition and WY-

Algorithm 5 Narrow band reduction algorithm

for $K = 1, N/L - 1$ **do**

*Generate block Householder reflector

Obtain a block Householder reflector $I - U^{(K)}\alpha^{(K)}{U^{(K)}}^\top$ transforms $D^{(K)}$ into a matrix which has upper triangular matrix as the first block and zero matrix as second and latter blocks.

$P^{(K)} = C^{(K)}U^{(K)}\alpha^{(K)}$
$\beta^{(K)} = \alpha^{(K)}{U^{(K)}}^\top {P^{(K)}}^\top/2$
$Q^{(K)} = P^{(K)} - U^{(K)}\beta^{(K)}$

$C^{(k)} := C^{(K)} - U^{(K)}{Q^{(K)}}^\top - Q^{(K)}{U^{(K)}}^\top$

end for

representation [31]. We next generate block vectors $P^{(K)}$ and $Q^{(K)}$ by multiplication $U^{(K)}$ to the matrix $C^{(K)}$ Then, we update the $C^{(K)}$ by $U^{(K)}$ and $Q^{(K)}$. The above operations are equivalent to the following block Householder transformation

$$C^{(K)} := \left(\tilde{H}^{(K)}\right)^{-1} C^{(K)}\tilde{H}^{(K)}$$

with the block Householder reflector

$$\tilde{H}^{(K)} := I - U^{(K)}\alpha^{(K)}{C^{(K)}}^\top.$$

Then, we complete the K-th block band reduction.

On these operations, matrix-vector multiplications and rank-2L updates occupy almost all of computation. We can compute these operations with BLAS L2.5 (computation among $L \times L$ matrices) then the cache reuse rate become higher if we set the bandwidth L larger. However, we get narrow band matrix instead of tridiagonal matrix if we use the narrow band reduction method therefore we cannot apply conventional tridiagonal eigen/singular solver directly and we have to adopt a solver for band matrices.

T. Imamura applies the divide and conquer method to the narrow band eigenvalue problem then get a higher performance than classical tridiagonalization and divide and conquer method. T. Imamura also reported that we get maximum performance if we reduce full-matrix to pentadiagonal matrix. However, the divide and conquer method on LAPACK may collapse and return error code for some ill-conditioned matrices. In this dissertation, we consider the other method to compute the singular value of positive definite pentadiagonal matrices. To solve the singular value problem, we introduce the OQDS algorithm for lower tridiagonal matrix which Cholesky-decomposed from the positive definite pentadiagonal matrix.

3.3 OQDS algorithm for lower tridiagonal matrices

The differential quotient difference with shifts algorithm (DQDS algorithm) is well-known method for singular value computation of tridiagonal (or decomposed bidiagonal)

matrices proposed by Fernando and Parlett and the algorithm indicates high performance regarding both speed and precision. However, it is hard to adapt the DQDS algorithm to narrow band matrices because the algorithm is highly specialized for bidiagonal matrices. Then, we try to adapt the OQDS algorithm to lower tridiagonal matrices. The OQDS algorithm proposed by von Matt has expandability for general triangular matrices as introduced in Subsection 3.1. Comparing with the DQDS algorithm as bidiagonal singular solver, OQDS algorithm has no practical advantage against the DQDS algorithm but theoretically, it is expected to perform high precision computation for general triangular matrices.

In this section, we introduce the shift strategy and convergence criteria required for the application of OQDS algorithm for lower tridiagonal matrices.

3.4 Shift strategy

In the implicit Cholesky decomposition described in Section 3.1.1, proper choice of the shift value τ significantly accelerates convergence of the OQDS algorithm, same as other LR-based algorithms. The shift value τ must be smaller than the minimum singular value of the matrix L to keep the positive-definiteness of $U^\top U$ while each singular value $\sigma_1, \sigma_2, \ldots, \sigma_{n-1}, \sigma_n$ decreases to $\sqrt{{\sigma_1}^2 - \tau^2}, \sqrt{{\sigma_2}^2 - \tau^2}, \ldots, \sqrt{{\sigma_n}^2 - \tau^2}$ by shift τ. Besides, it is reported that LR-based algorithms converge in proportion to the value of σ_{n-1}/σ_n then we can accelerate the convergence if we can estimate accurate lower bound of minimum singular value and reduce the value of σ_n by the shift. The reduction of the number of iterations until convergence not only makes the algorithm faster but also more accurate by the reduction of rounding error accumulates. Estimation method of lower bound of minimum singular value for lower tridiagonal matrices is not discussed actively while many strategies proposed for bidiagonal matrices that is applied to DQDS algorithm. Therefore, we propose a method to estimate the lower bound of the minimum singular value of the lower tridiagonal matrix L or the minimum eigenvalue of $L^\top L$.

In this section, we first introduce the classical Gerschgorin shift and next discuss the Algebraic shift for lower tridiagonal matrices.

3.4.1 Gerschgorin shift

Gerschgorin shift is a simple estimation method of lower bound of minimum eigenvalue based on the following theorem.

Theorem 3.1 (Gerschgorin [13]). *For an $n \times n$ matrix $A = (a_{ij})$, let us define*

$$R_i := \sum_{k \neq i} |a_{ik}|. \tag{13}$$

Then, for any eigenvalue λ of A, there exists an integer i such as

$$|\lambda - a_{ii}| \leq R_i. \tag{14}$$

If the matrix A is positive-definite symmetric, $\min(a_{ii} - R_i)$ gives a lower bound of the eigenvalues since all the eigenvalues of A are positive real number.

The Gerschgorin shift has the following two advantages;

- Low computational complexity
  - We just compute the summation of absolute of diagonal and nondiagonal elements.
- Tolerance to round off error
  - No recursion formula exists in Gerschgorin shift.

Because of the advantages, the Gerschgorin shift fits diagonal dominant matrices such as bidiagonal and lower tridiagonal matrices. Algorithm 6 summarize the computation of Gerschgorin shift of $L^\top L$.

3.4.2 Algebraic shift

The Gerschgorin shift has low computational complexity therefore computes a lower bound fast but for the matrices whose diagonal elements are not dominant such as the lower tridiagonal matrices, the Gerschgorin shift cannot return sharp shift value. On a preliminary experiment, we cannot compute singular values of large scale lower tridiagonal matrices by the OQDS algorithm if we only use the Gerschgorin shift. To resolve the problem, we apply Algebraic shift proposed by T. Yamashita et al. in [35]. Algebraic shift is composed of three shift; the generalized Newton, Laguerre and Kato-Temple shift obtained by the trace value of inverse matrix and it is reported by T. Yamashita that for bidiagonal matrices, the shift returns extremely sharp lower bound. In this subsection, we consider the application of the Algebraic shift to lower tridiagonal matrices. In the first, we introduce the method of obtaining trace value of pentadiagonal matrices and the Generalized Newton shift simply obtained by the trace.

3.4.3 A method for obtaining the trace of inverse of pentadiagonal matrices and generalized Newton shift

For a positive-definite symmetric matrix A and an arbitrary positive integer p, the value of $(\mathrm{tr}(A^{-p}))^{-1/p}$ is a lower bound of the eigenvalues of A. The lower bound is known as Generalized Newton shift.

Then, finding the value of $\mathrm{tr}((L^\top L)^{-p})$, we get a lower bound of singular values of L. We consider a differential method of computing the value of $\mathrm{tr}((L^\top L)^{-p})$ in this subsection.

Algorithm 6 Gerschgorin shift (gerschgorin(L))

$\tau := \alpha_{n-1}^2 + \beta_{n-2}^2 + \gamma_{n-3}^2 - |\alpha_{n-3}\gamma_{n-3}| - |\beta_{n-3}\gamma_{n-3} + \alpha_{n-2}\beta_{n-2}|$
if $\tau \leq 0$ **then**
  return 0
end if
$tmp := \alpha_{n-2}^2 + \beta_{n-3}^2 + \gamma_{n-4}^2 - |\alpha_{n-4}\gamma_{n-4}| - |\beta_{n-4}\gamma_{n-4} + \alpha_{n-3}\beta_{n-3}| - |\beta_{n-3}\gamma_{n-3} + \alpha_{n-2}\beta_{n-2}|$
if $tmp \leq 0$ **then**
  return 0
else if $tmp < \tau$ **then**
  $\tau := tmp$
end if
for i = N - 2 to 3 **do**
  $tmp := \alpha_i^2+\beta_{i-1}^2+\gamma_{i-2}^2-|\alpha_{i-2}\gamma_{i-2}|-|\beta_{i-1}\gamma_{i-1} + \alpha_i\beta_i|-|\beta_{i-2}\gamma_{i-2} + \alpha_{i-1}\beta_{i-1}|-|\alpha_i\gamma_i|$
  if $tmp \leq 0$ **then**
    return 0
  else if $tmp < \tau$ **then**
    $\tau := tmp$
  end if
end for
$tmp := \alpha_2^2 + \beta_1^2 - |\alpha_1\beta_1| - |\beta_1\gamma_1 + \alpha_2\beta_2| - |\alpha_2\gamma_2|$
if $tmp \leq 0$ **then**
  return 0
else if $tmp < \tau$ **then**
  $\tau := tmp$
end if
$tmp := \alpha_1^2 - |\alpha_1\beta_1| - |\alpha_1\gamma_1|$
if $tmp \leq 0$ **then**
  return 0
else if $tmp < \tau$ **then**
  $\tau := tmp$
end if

Let $\bar{L}$ be an $n \times n$ lower tridiagonal matrix,

$$\bar{L} = \begin{bmatrix} \bar{\alpha}_1 & & & & \\ \bar{\beta}_1 & \bar{\alpha}_2 & & & \\ \bar{\gamma}_1 & \bar{\beta}_2 & \bar{\alpha}_3 & & \\ & \ddots & \ddots & \ddots & \\ & & \bar{\gamma}_{n-2} & \bar{\beta}_{n-1} & \bar{\alpha}_n \end{bmatrix} \tag{15}$$

determined from L with shift s by

$$\bar{L}\bar{L}^\top = LL^\top - sI. \tag{16}$$

The relationships among elements are given by

$$\bar{\alpha}_i^2 + \bar{\beta}_{i-1}^2 + \bar{\gamma}_{i-2}^2 = \alpha_i^2 + \beta_{i-1}^2 + \gamma_{i-2}^2 - s, \tag{17}$$

$$\bar{\beta}_{i-2}\bar{\gamma}_{i-2} + \bar{\alpha}_{i-1}\bar{\beta}_{i-1} = \beta_{i-2}\gamma_{i-2} + \alpha_{i-1}\beta_{i-1}, \tag{18}$$

$$\bar{\alpha}_{i-2}\bar{\gamma}_{i-2} = \alpha_{i-2}\gamma_{i-2}. \tag{19}$$

Differentiating equations (17)–(19) with respect to s, we obtain

$$2\bar{\alpha}_i\bar{\alpha}_i' + 2\bar{\beta}_{i-1}\bar{\beta}_{i-1}' + 2\bar{\gamma}_{i-2}\bar{\gamma}_{i-2}' = -1, \tag{20}$$

$$\bar{\beta}_{i-2}'\bar{\gamma}_{i-2} + \bar{\beta}_{i-2}\bar{\gamma}_{i-2}' + \bar{\alpha}_{i-1}'\bar{\beta}_{i-1} + \bar{\alpha}_{i-1}\bar{\beta}_{i-1}' = 0, \tag{21}$$

$$\bar{\alpha}_{i-2}'\bar{\gamma}_{i-2} + \bar{\alpha}_{i-2}\bar{\gamma}_{i-2}' = 0. \tag{22}$$

Note that the α_i, β_i, γ_i are independent of s but the $\bar{\alpha}_i$, $\bar{\beta}_i$, $\bar{\gamma}_i$ are not. Differentiating once more, we get

$$2\bar{\alpha}_i'^2 + 2\bar{\alpha}_i\bar{\alpha}_i'' + 2\bar{\beta}_{i-1}'^2 + 2\bar{\beta}_{i-1}\bar{\beta}_{i-1}'' + 2\bar{\gamma}_{i-2}'^2 + 2\bar{\gamma}_{i-2}\bar{\gamma}_{i-2}'' = 0, \tag{23}$$

$$\bar{\alpha}_{i-2}''\bar{\gamma}_{i-2} + 2\bar{\alpha}_{i-2}'\bar{\gamma}_{i-2}' + \bar{\alpha}_{i-2}\bar{\gamma}_{i-2}'' = 0, \tag{24}$$

$$\bar{\beta}_{i-2}''\bar{\gamma}_{i-2} + 2\bar{\beta}_{i-2}'\bar{\gamma}_{i-2}' + \bar{\beta}_{i-2}\bar{\gamma}_{i-2}'' + \bar{\alpha}_{i-1}''\bar{\beta}_{i-1} + 2\bar{\alpha}_{i-1}'\bar{\beta}_{i-1}' + \bar{\alpha}_{i-1}\bar{\beta}_{i-1}'' = 0. \tag{25}$$

Let us write the eigenvalues of the matrix $LL^\top$ by $\lambda_1, \lambda_2, \cdots, \lambda_n$. Then, the characteristic polynomial of the matrix $\bar{L}\bar{L}^\top$

$$\begin{aligned} f(s) &= \det(LL^\top - sI) \\ &= (\lambda_1 - s)(\lambda_2 - s)\cdots(\lambda_n - s), \end{aligned} \tag{26}$$

since the matrix L is triangle, it is expressed by

$$f(s) = \bar{\alpha}_1 \bar{\alpha}_2 \cdots \bar{\alpha}_n. \tag{27}$$

Let us define

$$g(s) := -\frac{f'(s)}{f(s)} = -2\frac{\bar{\alpha}_1'}{\bar{\alpha}_1} - 2\frac{\bar{\alpha}_2'}{\bar{\alpha}_2} - \cdots - 2\frac{\bar{\alpha}_n'}{\bar{\alpha}_n}, \tag{28}$$

$$\begin{aligned} h(s) &:= g'(s) \\ &= -2\frac{\bar{\alpha}_1''\bar{\alpha}_1 - \bar{\alpha}_1'^2}{\bar{\alpha}_1^2} - \cdots - 2\frac{\bar{\alpha}_n''\bar{\alpha}_n - \bar{\alpha}_n'^2}{\bar{\alpha}_n^2} \end{aligned} \tag{29}$$

so that $g(0) = \mathrm{tr}((L^\top L)^{-1})$ and $h(0) = \mathrm{tr}((L^\top L)^{-2})$. Each $\bar{\alpha}_i$ tends to α_i as $s \to 0$. Hence, we can calculate the value of $\bar{\alpha}_i'$, $\bar{\beta}_i'$, $\bar{\gamma}_i'$, $\bar{\alpha}_i''$, $\bar{\beta}_i''$, $\bar{\gamma}_i''$ at $s = 0$ from α_i, β_i, γ_i by using (20)–(25), and then $g(0)$ and $h(0)$ by (28) and (29). It is clear by the definition of $g(0)$ and $h(0)$ that the values are always nonnegative without numerical error (in infinite-precision arithmetic).

3.4.4 Laguerre shift

If we already have the value of $\mathrm{tr}((LL^\top)^{-1})$ and $\mathrm{tr}((LL^\top)^{-2})$, we could improve the sharpness of the shift by $O(1)$ operation. Laguerre shift is one of the methods to improve the shift value.

Theorem 3.2 (Laguerre [14])**.** *For an $n \times n$ positive-definite symmetric pentadiagonal matrix $B = LL^\top$, let θ be the following value:*

$$\theta := \frac{n}{\mathrm{tr}\,(B^{-1}) + \sqrt{(n-1)\left(n\,\mathrm{tr}\,(B^{-2}) - \mathrm{tr}\,(B^{-1})^2\right)}}.$$

Then, the θ is a lower bound of the eigenvalues of B which is greater than $\mathrm{Tr}(B^{-1})^{-1}$ and $\mathrm{Tr}(B^{-2})^{-1/2}$.

If the value $n\,\mathrm{tr}\left(B^{-2}\right) - \mathrm{tr}\left(B^{-1}\right)^2$ is negative, Laguerre shift is useless. In that case, we adopt the generalized Newton shift.

3.4.5 Kato-Temple shift

There is another lower bound estimation method, Kato-temple shift.

Theorem 3.3 (Kato-Temple [12])**.** *For an $n \times n$ symmetric matrix A_n, let A_{n-1} denote the submatrix of A_n obtained by deleting the last row and column. For any lower bound*

λ^* of the eigenvalues of A_{n-1}, and for any $\mathbf{x} \in \mathbb{R}^n$, $\|\mathbf{x}\| = 1$, let $\rho = \mathbf{x}^\top A\mathbf{x}$. Then, if $\rho < \lambda^*$, the value

$$\rho - \frac{\|A_n\mathbf{x} - \rho\mathbf{x}\|^2}{\lambda^* - \rho} \leq \lambda_{\min}(A_n)$$

gives a lower bound of the eigenvalues of A_n.

We choose $\mathbf{x} = (0, \ldots, 0, 1)^\top$. The method requires λ^* which is a lower bound for the submatrix A_{n-1}, but the generalized Newton method enables us to find the lower bound of A_{n-1} in computation of the lower bound of A_n. Consequently, we obtain one more improved shift value by $O(1)$ operation.

The procedure of the proposed shift composed by the generalized Newton, Laguerre and Kato-Temple is shown in Algorithm 7. We adopt the largest value of them.

Then, one step of the OQDS algorithm works as Algorithm 8.

3.5 Convergence criteria

We can stop the OQDS algorithm where the non-diagonal elements of matrices L become so small but it requires many iterations to converge all non-diagonal elements into zero. So many iterations not only decelerate the algorithm but also reduce the precision of obtained singular value. Therefore, we stop the algorithm in an appropriate number of iteration. In this section, we introduce deflation and splitting for the OQDS algorithm and design convergence criteria for these operations.

3.5.1 Deflation and splitting

Deflation and splitting are familiar stopping method for QD-based algorithms. Deflation is an operation isolating sufficiently converged σ_n from matrix L and restarting the OQDS algorithm for L_{n-1} submatrix of L as the following figure 4. Splitting is an

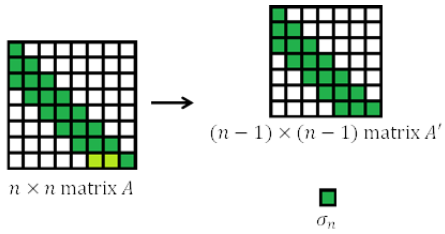


Figure 4: deflation

operation separating matrix L to submatrices $L^{(1)}$ and $L^{(2)}$ as the figure 5 below where corresponding off-diagonal elements converges sufficiently. Through these operations, not only we can stop the algorithm adequately but also accelerates the algorithm by effective shift and parallel computing for split matrices. However, it is nontrivial how to assess a series of matrices generated by the iterative process of the OQDS algorithm

Algorithm 7 Algebraic shift for lower tridiagonal matrix L (algshift(L))

$\alpha'_1 := -1/(2\alpha_1)$, $\beta'_1 := -\alpha'_1\beta_1/\alpha_1$, $\gamma'_1 := -\alpha'_1\gamma_1/\alpha_1$
$\alpha'_2 := (-\beta_1\beta'_1 - 0.5)/\alpha_2$, $\beta'_2 := -(\gamma'_1\beta_1 + \gamma_1\gamma'_1 + \alpha'_2\beta_2)/\alpha_2$
$\alpha'_3 := -(1 + 2 \times \gamma_1\gamma'_1 + 2\beta_2\beta'_2)/(2\alpha_3)$
$\alpha''_1 := -{\alpha'_1}^2/\alpha_1$, $\beta''_1 := -(\alpha''_1\beta_1 + 2\alpha'_1\beta'_1)/\alpha_1$, $\gamma''_1 := -(\alpha''_1\gamma_1 + 2\alpha'_1\gamma'_1)/\alpha_1$
$\alpha''_2 := -({\beta'_1}^2 + \beta_1\beta''_1 + {\alpha'_2}^2)/\alpha_2$, $\beta''_2 := -(\gamma''_1\beta_1 + 2\gamma'_1\beta'_1 + \gamma_1\beta''_1 + \alpha''_2\beta_2 + 2\alpha'_2\beta'_2)/\alpha_2$
$\alpha''_3 := -({\gamma'_1}^2 + \gamma_1\gamma''_1 + {\beta'_2}^2 + \beta_2\beta''_2 + {\alpha'_3}^2)/\alpha_3$
for $i = 4$ to N **do**
  $\gamma'_{i-2} := -\alpha'_{i-2}\gamma_{i-2}/\alpha_{i-2}$
  $\beta'_{i-1} := -(\beta'_{i-2}\gamma_{i-2} + \beta_{i-2}\gamma'_{i-2} + \alpha'_{i-1}\beta_{i-1})/\alpha_{i-1}$
  $\alpha'_i := -(1 + 2\beta_{i-1}\beta'_{i-1} + 2\gamma_{i-2}\gamma'_{i-2})/(2\alpha_i)$
  $\gamma''_{i-2} := -(\alpha''_{i-2}\gamma_{i-2} + 2\alpha'_{i-2}\gamma'_{i-2})/\alpha_{i-2}$
  $\beta''_{i-1} := -(\beta''_{i-2}\gamma_{i-2} + 2\beta'_{i-2}\gamma'_{i-2}$
    $+\beta_{i-2}\gamma''_{i-2} + \alpha''_{i-1}\beta_{i-1} + 2\alpha'_{i-1}\beta'_{i-1})/\alpha_{i-1}$
  $\alpha''_i := -({\alpha'_i}^2 + {\beta'_{i-1}}^2 + \beta_{i-1}\beta''_{i-1} + {\gamma'_{i-2}}^2 + \gamma_{i-2}\gamma''_{i-2})/\alpha_i$
end for
$tr1 := 0$
for $i = 1$ to $N - 1$ **do**
  $tr1 := tr1 - (2\alpha'_i/\alpha_i)$
end for
$tr2 := 0$
for $i = 1$ to $N - 1$ **do**
  $tr2 := tr2 - 2(\alpha''_i\alpha_i - {\alpha'_i}^2)/\alpha_i^2$
end for
$\lambda^* := 1/sqrt(tr2)$
$tmp := n \times tr2 - tr1^2$
if $tmp > 0$ **then**
  $\lambda^* := \max(\lambda^*, n/(tr1 + \sqrt{(n-1) \times tmp}))$
end if
$tr1 := tr1 - (2\alpha'_N/\alpha_N)$
$tr2 := tr2 - 2(\alpha''_N\alpha_N - {\alpha'_N}^2)/\alpha_N^2$
$\tau := 1/sqrt(tr2)$
$x := (0, \ldots, 0, 1)^\top$
$\rho := \mathbf{x}^\top L_{n-1}\mathbf{x}$
if $\rho < \lambda^*$ **then**
  $\tau := \max(\tau, \rho - \|A_n\mathbf{x} - \rho\mathbf{x}\|^2/(\lambda^* - \rho))$
end if
$tmp := n \times tr2 - tr1^2$
if $tmp > 0$ **then**
  $\tau := \max(\tau, n/(tr1 + \sqrt{(n-1) \times tmp}))$
end if
return τ

Algorithm 8 oqds step(oqds(L, T))

$flag := 0$
if $flag = 0$ **then**
  $\tau :=$ algshift(L)
else if $flag := 1$ **then**
  $\tau :=$ gerschgorin(L)
else
  $\tau := 0$
end if
if $\tau^2 + T^2 = T^2$ **then**
  $\check{L} :=$ icds$(L, 0)$
  $L := \check{L}$
else
  $\check{L} :=$ icds(L, τ)
  if $\check{\alpha} \neq \check{\alpha}$ **then**
    $flag := flag + 1$
  else
    $T := \sqrt{T^2 + \tau^2}$
    $L := \check{L}$
  end if
end if

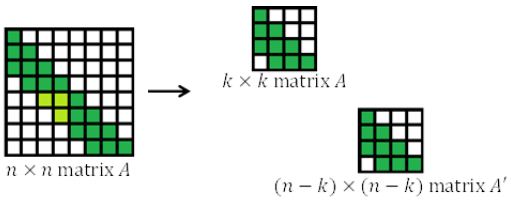


Figure 5: splitting

converges sufficiently. In the next subsection, we consider the situation that we can perform the deflation or splitting properly where the values of off-diagonal and second off-diagonal elements of lower tridiagonal matrices are so small. Specifically, we estimate the perturbation of eigenvalues of $L^\top L$ and $LL^\top$ by Weyl's theorem.

3.5.2 1-norm convergence criteria

Let us write

$$\hat{L} := L - \beta_k \mathbf{e}_{k+1} \mathbf{e}_k{}^\top$$

which is the matrix equal to L except for zero at $(k+1, k)$-entry. Then

$$L^\top L = \hat{L}^\top \hat{L} + E_1, \tag{30}$$
$$LL^\top = \hat{L}\hat{L}^\top + E_2 \tag{31}$$

hold, where the perturbation matrices

$$E_1 := \beta^2 \mathbf{e}_k \mathbf{e}_k{}^\top + \alpha_{k+1}\beta_k \left(\mathbf{e}_k \mathbf{e}_{k+1}^\top + \mathbf{e}_{k+1}\mathbf{e}_k^\top \right) + \beta_k \gamma_{k-1} \left(\mathbf{e}_{k-1}\mathbf{e}_k^\top + \mathbf{e}_k \mathbf{e}_{k-1}^\top \right), \tag{32}$$

$$E_2 := \beta^2 \mathbf{e}_{k+1} \mathbf{e}_{k+1}{}^\top + \alpha_k \beta_k \left(\mathbf{e}_{k-1}\mathbf{e}_k^\top + \mathbf{e}_k \mathbf{e}_{k-1}^\top \right) + \beta_k \gamma_k \left(\mathbf{e}_{k+1}\mathbf{e}_k^\top + \mathbf{e}_k \mathbf{e}_{k+1}^\top \right). \tag{33}$$

Theorem 3.4 (Weyl's monotonicity theorem [15, 16])**.** *For an $n \times n$ positive-definite matrix A, let $\lambda_i(A)$ denote the ith largest eigenvalue of A. Then, there exist reals u_i and v_i such that*

$$\lambda_i \left(L^\top L \right) = \lambda_i \left(\hat{L}^\top \hat{L} \right) + u_i \left\| E_1 \right\|_p, \tag{34}$$
$$\lambda_i \left(LL^\top \right) = \lambda_i \left(\hat{L}\hat{L}^\top \right) + v_i \left\| E_2 \right\|_p \tag{35}$$

where $|u_i| \leq 1$, $|v_i| \leq 1$.

From the definitions (32) and (33) of E_1 and E_2, we have

$$\|E_1\|_1 = \|E_1\|_\infty = |\beta_k| \left(|\alpha_{k+1}| + |\beta_k| + |\gamma_{k-1}| \right), \tag{36}$$
$$\|E_2\|_1 = \|E_2\|_\infty = |\beta_k| \left(|\alpha_k| + |\beta_k| + |\gamma_k| \right). \tag{37}$$

By Weyl's monotonicity theorem, we thus get the numerical deflation or splitting criterion to neglect a off-diagonal element β_k:

$$T + |\beta_k| \left(|\beta_k| + \min \left(|\alpha_{k+1}| + |\gamma_{k-1}|, |\alpha_k| + |\gamma_k| \right) \right) \simeq T, \tag{38}$$

where '$\simeq$' means that the left-hand side and the right-hand side are numerically equal and T is the square summation of shift values previously applied. We assume that β_k is so small and negligible provided that (38) holds numerically.

Similarly, we get the numerical criterion for neglecting a second off-diagonal element γ_k. On the setting of

$$\hat{L} := L - \gamma_k \mathbf{e}_{k+2} \mathbf{e}_k{}^\top,$$

the perturbation matrices are given by

$$E_1' := \gamma^2 \mathbf{e}_k \mathbf{e}_k{}^\top + \alpha_{k+2}\gamma_k \left(\mathbf{e}_{k+2}\mathbf{e}_k^\top + \mathbf{e}_k \mathbf{e}_{k+2}^\top \right) + \beta_{k+1}\gamma_k \left(\mathbf{e}_{k+1}\mathbf{e}_k^\top + \mathbf{e}_k \mathbf{e}_{k+1}^\top \right),$$
$$E_2' := \gamma^2 \mathbf{e}_{k+2} \mathbf{e}_{k+2}{}^\top + \alpha_k\gamma_k \left(\mathbf{e}_{k-2}\mathbf{e}_k^\top + \mathbf{e}_k \mathbf{e}_{k-2}^\top \right) + \beta_k\gamma_k \left(\mathbf{e}_{k-1}\mathbf{e}_k^\top + \mathbf{e}_k \mathbf{e}_{k-1}^\top \right).$$

Then, by evaluating the 1- and ∞-norms of these matrices, we obtain the criterion for neglecting a second off-diagonal element γ_k as follows:

$$T + |\gamma_k| \left(|\gamma_k| + \min \left(|\alpha_{k+2}| + |\beta_{k+1}|, |\alpha_k| + |\beta_k| \right) \right) \simeq T. \tag{39}$$

For the matrices in iteration, we perform deflation and splitting as follows:

1. If β_{n-1} and γ_{n-2} in the last row satisfy the criteria (38) and (39), then we deflate the matrix by deleting the last row and column.

2. If β_{k-1}, γ_{k-1} and γ_{k-2} satisfy the criteria (38) and (39), then we split the matrix into two submatrices formed by rows and columns 1 to $k-1$ and k to n, respectively.

3.5.3 2-norm convergence criteria

In the previous subsection, we design convergence criteria by estimating the perturbation of equation (34) and (35) with 1-norm. Although the choice of norm dimension p is arbitrary, we can assess the convergence more sharply if we obtain 2-norm of perturbation matrices E_1 and E_2. In other words, we can perform deflation and splitting faster with 2-norm convergence criteria. 2-norm of a matrix is equal to the maximum absolute of eigenvalues, however, E_1 and E_2 are three-by-three matrices and we generally have to solve complicated formula to obtain the eigenvalues. Moreover, there are no formula to obtain eigenvalues for larger matrices if we consider the algorithm for wider band matrices. However, in case of 32, characteristic polynomial of E_1 is

$$f(E_1) = x^3 - x^2\beta_k^2 - x(\beta_k^2\gamma_{k-1}^2 + \beta_k^2\alpha_{k+1}^2)$$

hence we can factorize and reduce the cubic characteristic polynomial to quadratic polynomial as follows

$$f(E_1) = x(x^2 - x\beta_k^2 + (\beta_k^2\gamma_{k-1}^2 + \beta_k^2\alpha_{k+1}^2)).$$

Then, we obtain the following 2-norm convergence criterion

$$T + \frac{1}{2} |\beta_k| \left(|\beta_k| + \sqrt{\beta_k^2 + \min(4\alpha_{k+1}^2 + 4\gamma_{k-1}^2, 4\alpha_k^2 + 4\gamma_k^2)} \right) \simeq T. \tag{40}$$

Similarly, for E_2, we obtain the criterion

$$T + \frac{1}{2}|\gamma_k| \left(|\gamma_k| + \sqrt{\gamma_k^2 + \min(4\alpha_{k+2}^2 + 4\beta_{k+1}^2, 4\alpha_k^2 + 4\beta_k^2)} \right) \simeq T. \tag{41}$$

By using these 2-norm criteria, the OQDS algorithm may not run faster because 2-norm criteria require square root computation though the criteria allow to perform deflation and splitting faster.

3.6 Numerical experiments

We perform some numerical experiments in order to confirm that the proposed singular solver with narrow band reduction approach performs faster computation than conventional method. In other word, if the decrease of computation time of tridiagonalization compared with conventional bidiagonalization surpasses the increase of singular value computation time of OQDS algorithm for lower tridiagonal matrices compared with that of DQDS algorithm for bidiagonal matrices, it can be said that the goal is achieved.

The numerical experiments were performed on a Linux PC with Intel Core i7 920 (Nehalem) 2.66GHz and DDR3-1066 12GB memory. Each program is compiled by Intel C/C++ compiler with -fast and -mkl option.

We apply the classical Householder reduction method to bidiagonalization while apply the narrow band reduction method to lower tridiagonalization, respectively. The `dgebd2` subroutine on LAPACK is used for bidiagonalization and hand coding reduction program with BLAS L2.5 is used for lower tridiagonalization. Table 1 show the computation time of each algorithm. The first row shows the size of matrices. The second and the third rows show the computation time taken by the classical Householder bidiagonalization and the block Householder lower tridiagonalization, respectively.

Table 1: Computation time of preprocessing [sec.]

matrix size	2000	4000	6000	8000	10000
bidiagonalization	9.554	78.877	261.779	637.437	1209.908
lower tridiagonalization (proposed)	3.991	33.514	110.846	267.367	518.655

We compare the time of singular value computation by DQDS for bidiagonal matrices and OQDS for lower tridiagonal matrices. It should be noted that: The DQDS algorithm were applied to random bidiagonal matrices and the proposed OQDS algorithm for lower tridiagonal matrices were applied to random lower tridiagonal matrix. We adopt the DBDSQR routine on LAPACK as DQDS algorithm while the OQDS is our hand coding program with 1-norm convergence criteria. Table 2 shows the computation time of each algorithm. The first row shows the size of matrices. The second and the third rows show the computation time of preprocessing taken by the classical Householder bidiagonalization for the DQDS and by the block Householder lower tridiagonalization for the extended OQDS algorithm.

Table 2: Computation time of singular value computation [sec.]

matrix size	2000	4000	6000	8000	10000
DQDS	0.191	0.633	1.364	2.424	3.698
extended OQDS	1.146	4.119	8.641	15.166	22.312

We perform more experiments for larger matrices to confirm the scalability of proposed OQDS algorithm for lower tridiagonal matrices. The first row in Table 3 shows the size of matrices. The second row shows the computation time taken by proposed OQDS algorithm for lower tridiagonal matrices.

Table 3: Computation time of singular value computation (larger size) [sec.]

matrix size	20000	40000	60000	80000	100000
DQDS	12.469	42.563	97.847	167.703	247.550
extended OQDS	84.974	316.973	703.159	1266.273	1862.898

3.7 Discussion about the narrow band approach of singular value decomposition with the extended OQDS algorithm

The first experiment (Table 1) shows a significant effectiveness of the proposed method: the computation time for the narrow band reduction takes less time than half of the classical Householder reduction for matrices of the same size. The result suggests that the bottlenecks of the Householder reduction exist in matrix-vector multiplications of the Householder transformation and rank-2 updates. Then the performance can be improved by introducing the narrow band reduction method. In addition to less computational complexity of the narrow band reduction, we can apply cache efficient BLAS level 2.5 matrix-vectors operations instead of BLAS level 2 matrix-vector operations. The method demonstrates remarkably high performance even the block bandwidth $L = 2$. In the lower tridiagonalization, we compute the multiplication among matrix and two vectors simultaneously therefore we reduce the number of DRAM access which consumes a vast amount of time. As a result, we can perform the tridiagonalization in the time of less than 50 percent of bidiagonalization.

On the next experiment (Table 2, 3), the proposed OQDS algorithm costs a longer time to compute singular values of lower tridiagonal matrices than that by the DQDS algorithm to compute singular values of bidiagonal matrices. However, the preprocessing has the computational complexity $O(n^3)$ for matrix size n while singular value computation has $O(n^2)$ complexity. Therefore a vast amount of the computation time is consumed by the preprocessing. In fact, for the same size of matrix, the preprocessing requires much longer time than singular value computation (Table 1). For this reason, the total time for computing singular values of full matrices by the proposed method is

significantly shorter than for conventional Householder bidiagonalization plus the DQDS algorithm.

Moreover, we confirm the scalability of the proposed OQDS algorithm for larger matrices by the experiment in Table 3. The computation time of table are then in proportion to the square of matrix size n. It shows that we can apply the method to larger problems.

On the other hand, we found some disadvantages of the approach. The first is that the computational accuracy is limited to that of Householder tridiagonalization. Our attempt to improve the accuracy of the OQDS algorithm does not contribute to total accuracy containing tridiagonalization and singular value computation. Moreover, it is difficult to compare the accuracy of the computed singular values because there is no test matrix of the tridiagonal form. Therefore, we could not perform a control experiment for the two algorithms of different input form, including preliminary reduction, in order to find the bottleneck of the accuracy. The another disadvantage is the number of iterations of the OQDS algorithm. The increase of the iteration is within expectation since input matrix has more non-zero entries, however, the larger band width does not only harden the estimation of lower bound of minimum singular values but also depresses the success rate of shift operation. Generalized Givens transformation introduced in Section 3.1.1 might produce negative diagonal entry even if the shift value τ satisfies $\tau < \min(\sigma_i)$. By this problem, however precise we estimate the lower bound of minimum singular value, the acceleration by the shift strategy is limited to the availability of the generalized Givens transformation.

For those reasons, we gave up more improve of this approach and fumbled for other application of the OQDS algorithm focusing its ability to compute singular vectors.

4 Column Space Computation Method by Using the Combination of DQDS and OQDS Algorithms

The propose an combination method of DQDS and OQDS algorithms which compute column space of bidiagonal matrix. The method is an applications of OQDS algorithm focusing its ability to compute singular vectors. The target matrix is bidiagonal as same as DQDS algorithm, and utilize the computational stability instead of the expandability to band matrix which introduced the above section. In this method, DQDS algorithm is used for computing singular values and OQDS algorithm is used for singular vectors.

4.1 DQDS algorithm for computing singular values

We introduce the outline of DQDS algorithm before the proposed method.

Algorithm 9 DQDS algorithm

```
d_1 := q_1 − s;
for k := 1, 2, ··· , n − 1 do
    q̂_k := d_k + e_k;
    if SAFMIN × q̂_k < q_{k+1}  and  SAFMIN × q_{k+1} < q̂_k then
        ê_k := (q_{k+1}/q̂_k) e_k;
        d_{k+1} := (q_{k+1}/q̂_k) d_k − s;
    else
        ê_k := (e_k/q̂_k) q_{k+1};
        d_{k+1} := (d_k/q̂_k) q_{k+1} − s;
    end if
end for
q̂_n := d_n;
```

Let B be an $n \times n$ bidiagonal matrix:

$$B = \begin{bmatrix} \sqrt{q_1} & \sqrt{e_1} & & \\ & \sqrt{q_2} & \ddots & \\ & & \ddots & \sqrt{e_{n-1}} \\ & & & \sqrt{q_n} \end{bmatrix}. \tag{42}$$

In the single iteration of DQDS algorithm, the bidiagonal matrix B is transformed into the $n \times n$ upper bidiagonal matrix,

$$\hat{B} = \begin{bmatrix} \sqrt{\hat{q}_1} & \sqrt{\hat{e}_1} & & \\ & \sqrt{\hat{q}_2} & \ddots & \\ & & \ddots & \sqrt{\hat{e}_{n-1}} \\ & & & \sqrt{\hat{q}_n} \end{bmatrix}. \tag{43}$$

Algorithm 9 shows the operation sequence of DQDS algorithm. Here, a variable s means a shift value. Let Σ be a diagonal matrix arranged in descending order of singular values. After the repetition of the above operation in the DQDS algorithm, $\hat{B}$ converges into a diagonal matrix D. Then $\Sigma_{kk} = \sqrt{D_{kk} + S}$ $(k = 1, \cdots, n)$, where S is the sum of the value of shift s, is satisfied. Similar to OQDS algorithm, DQDS algorithm is also accelerated its convergence by introducing the shift.

In DQDS algorithm, all variable must be non-negative to keep the positive definiteness of original matrix. If $d_k = 0$, then imply $d_{k+1} < 0$ because $s > 0$ and the DQDS algorithm lose the constraint. On the other hand, in the case that the last diagonal entry $\hat{q}_n = d_n = 0$, one iteration in the DQDS algorithm can be terminated correctly. Consequently, the value of shift s can be set to not only less than the minimum eigenvalue $\lambda_{\min}\left(B^\top B\right)$ in $B^\top B$ but also equal to the minimum eigenvalue.

The variable d_k in the DQDS algorithm can be formulated to the form of $f \times g + h$ which is optimized as the fused multiply-add operation and provides high precision computation. DQDS algorithm achieves high accuracy by the introducing of the form.

When DQDS algorithm is terminated, the diagonal elements are arranged in descending order of the singular values. Considering its feature, in the case that $q_1 < q_n$, elements in B are replaced as follows:

$$B_r = \begin{bmatrix} \sqrt{q_n} & \sqrt{e_{n-1}} & & \\ & \sqrt{q_{n-1}} & \ddots & \\ & & \ddots & \sqrt{e_1} \\ & & & \sqrt{q_1} \end{bmatrix}. \tag{44}$$

Even with this replacement, singular values of B_r are equal to that of B. Thus, after the replacement that $B \leftarrow B_r$, DQDS algorithm is adopted to B_r.

In a singular value computation with shift, the summation of shift values is added to the obtained diagonal elements. The strategy to give the value of shift is expressed in Section 4.1.1. Let $s^{(i)}$ set the value of shift s in the i-th iteration. The summation of the shift values S is computed as follows:

$$S = \sum_{i=1}^{i_0} s^{(i)}. \tag{45}$$

In Equation (45), a tiny value may be added to a large value. In this case, loss of trailing digits occurs. To avoid this problem, we introduce the double-double arithmetic[33], which expresses one number using two double precision floating point numbers.

4.1.1 Convergence acceleration with origin shift

We introduce shift strategy which accelerate the convergence of DQDS algorithm likewise the case of OQDS algorithm for lower tridiagonal matrices introduced in Section 3.4. As shown above, shift value must be less than or equal to the minimum of eigenvalues of B.

Let s be an arbitrary positive number. The DQDS algorithm can be formulated as Equation (46), which consists only of d_k.

$$\begin{aligned} d_1 &= q_1 - s, \\ d_k &= q_k \frac{d_{k-1}}{(d_{k-1} + e_{k-1})} - s, \quad (k = 2, \cdots, n). \end{aligned} \tag{46}$$

In the context of d_k, the following operation is repeated for s.

1. If $d_k \leq 0 (k = 1, \ldots, n)$, $s' \leftarrow \max(d_k + s, 0)$.
2. If s' is equal to s in floating-point computation, $s' \leftarrow 0, d_k \leftarrow 0$.
3. After the above operation, if $d_n \geq 0$, then we end the repetition.
4. Moreover, in order to prevent s from becoming too small, $s \leftarrow \max(s', 0.75s)$
5. Recomputation eq.(46) using the above s.

Even if s might not be a lower bound of $\lambda_{\min}\left(B^\top B\right)$, by repeating the above operation multiple times, s is almost always transformed into positive number, which satisfies a lower bound of $\lambda_{\min}\left(B^\top B\right)$. We can utilize the above technique not only in Equation (46) but also in the DQDS algorithm.

The design of the shift strategy, which gives the value of shift s in the DQDS algorithm is widely discussed and becomes too complicated to explain in this dissertation. In this section, we do not explain the aggressive shift [34] which is implemented in the `DLASQ` routine of LAPACK [2], but our proposed shift strategy.

In our proposed shift strategy, the combination of the following estimation methods of lower bound of the minimum eigenvalue.

- Generalized Rutishauser bound [18]
- Maximum value of Laguerre bound [5], Newton bound, generalized Newton bound [35]
- A bound based on the Collatz inequality [20]
- Johnson bound [19]

We introduce mathematical definitions and the combination of them.

The generalized Rutishauser bound is expressed as following. Let s set to $\lambda_{\min}\left(F^\top F\right)$, where

$$F = \begin{bmatrix} \sqrt{q_{n-1}} & \sqrt{e_{n-1}} \\ 0 & \sqrt{q_n} \end{bmatrix}. \tag{47}$$

$\lambda_{\min}\left(F^\top F\right)$ is an upper bound of $\lambda_{\min}\left(B^\top B\right)$. Through appropriate update procedure introduced in Section 4.1.1, s becomes a positive number and a tight lower bound of $\lambda_{\min}\left(B^\top B\right)$. In the generalized Rutishauser bound, only in the case that $d_k > 0\,(k = 1, \ldots, n-1)$ and $d_n < 0$, s is adopted as the lower bound of $\lambda_{\min}\left(B^\top B\right)$. Of course, if $d_k > 0\,(k = 1, \ldots, n-1)$ and $d_n \geq 0$, $\lambda_{\min}\left(F^\top F\right)$ can be regarded as the suitable amount of shift. When k satisfies $d_k \leq 0\,(k = 1, \ldots, n-1)$, the other bounds is used as the value of shift.

Using

$$a = \mathrm{tr}\left(\left(BB^\top\right)^{-1}\right), \tag{48}$$

and

$$b = \mathrm{tr}\left(\left(BB^\top\right)^{-2}\right), \tag{49}$$

the Laguerre bound, the Newton bound, and the generalized Newton bound are defined as follows:

- Laguerre bound

$$L = \frac{n}{a + \sqrt{(n-1)\,(nb - a^2)}} \tag{50}$$

- Newton bound

$$N = a^{-1} \tag{51}$$

- Generalized Newton bound

$$GN = b^{-\frac{1}{2}} \tag{52}$$

Here, a and b are computed by the recurrence formula(53) and (54) [35].

$$a = \sum_{k=1}^{n} f_k, \tag{53}$$

$$\begin{cases} f_1 = \dfrac{1}{q_1}, \\ f_k = \dfrac{1}{q_k} + \dfrac{e_{k-1}}{q_k} f_{k-1}, \quad k = 2, \cdots, n, \end{cases}$$

$$b = \sum_{k=1}^{n} g_k, \tag{54}$$

$$\begin{cases} g_1 = f_1^2, \\ g_k = f_k^2 + \dfrac{e_{k-1}}{q_k} \left(g_{k-1} + f_{k-1}^2 \right), \quad k = 2, \cdots, n. \end{cases}$$

In theoretical,

$$N \leq GN \leq L, \tag{55}$$

is always satisfied but in a numerical computation, Equation (55) may not be satisfied because of rounding error. Therefore, we adopt $X = \max(N, GN, L)$ as the lower bound.

In the Collatz inequality, if all elements in an $n \times n$ matrix A are positive number and $\mathbf{v}$ is a vector, which consists of n positive elements, then,

$$\lambda_{\max}(A) \leq \max_k \frac{(A\mathbf{v})_k}{v_k}. \tag{56}$$

If all elements in B are positive number and K is defined as follows,

$$K = \begin{bmatrix} \sqrt{q_1} & & & \\ -\sqrt{e_1} & \sqrt{q_2} & & \\ & \ddots & \ddots & \\ & & -\sqrt{e_{n-1}} & \sqrt{q_n} \end{bmatrix}, \tag{57}$$

the lower bound of $\lambda_{\min}\left(B^\top B\right)$ is obtained using the following inequality:

$$\begin{aligned} \min_k \frac{\mathbf{v}_k}{\left((K^\top K)^{-1} \mathbf{v} \right)_k} &\leq \lambda_{\min}\left(K^\top K\right) \\ &= \lambda_{\min}\left(B^\top B\right), \end{aligned} \tag{58}$$

since all elements of $\left(K^\top K\right)^{-1}$ are positive number. $\mathbf{v}$ is generated by using the inverse iteration method [36]:

$$\mathbf{x} = \left(K^\top K\right)^{-1} (1, 1, \cdots, 1)^\top, \tag{59}$$

$$\mathbf{v} = \frac{\mathbf{x}}{\max_k \mathbf{x}_k}. \tag{60}$$

Moreover,

$$\frac{1}{\max_k \mathbf{x}_k} \leq \lambda_{\min}\left(K^\top K\right) = \lambda_{\min}\left(B^\top B\right), \tag{61}$$

can be adopted as another lower bound.

Johnson bound is the lower bound based on the following theorem. Let us set $C = \left(B^\top + B\right)/2$.

$$\lambda_{\min}(C) \leq \sqrt{\lambda_{\min}(B^\top B)}, \tag{62}$$

is satisfied. Therefore, through adopting Gerschgorin bound [13] to C, the lower bound $\lambda_{\min}\left(B^\top B\right)$ is obtained.

In our proposed shift strategy, at fast, the generalized Rutishauser bound is computed. When the positive lower bound of $\lambda_{\min}\left(B^\top B\right)$ can be obtained, the proposed shift strategy is terminated and the value of shift s sets to the lower bound. Otherwise, the Laguerre bound, the Newton bound, and the generalized Newton bound are computed. When

$$Z < 2 \times X, \tag{63}$$

is satisfied, X is adopted as the value of shift s. Here, Z means the upper bound of $\lambda_{\min}\left(B^\top B\right)$. Z is defined as $Z = \min(z_1, z_2, z_3)$:

- By using the small matrix F,

$$z_1 = \lambda_{\min}\left(F^\top F\right). \tag{64}$$

- By using the approximate amount g_k for Z,

$$z_2 = \frac{1}{\sqrt{\max_{k=1}^{n} g_k}}. \tag{65}$$

- The upper bound is obtained by using an optimization problem [37]. If, using a and b, an integer number m is determined within $m - 1 < a^2/b \leq m$, then,

$$z_3 = \frac{m}{a + \sqrt{(mb - a^2)/(m-1)}}, \tag{66}$$

  is the value for Z.

When Equation (63) is not satisfied, s is computed by using the bound based on the Collatz inequality. However, in the case that the value of shift s is not a positive number, the value of shift s is reset by using the Johnson bound.

Algorithm 10 DQD method

$d_1 := q_1$;
if $d_1 + S = S$ **then**
  $d_1 := 0$;
end if
for $k := 1, 2, \cdots, n-1$ **do**
  $\hat{q}_k := d_k + e_k$;
  if $\hat{q}_k = 0$ **then**
    $\hat{e}_k := 0,\ d_{k+1} := q_{k+1}$;
  else if $SAFMIN \times \hat{q}_k < q_{k+1}$ and $SAFMIN \times q_{k+1} < \hat{q}_k$ **then**
    $\hat{e}_k := (q_{k+1}/\hat{q}_k)\, e_k$;
    $d_{k+1} := (q_{k+1}/\hat{q}_k)\, d_k$;
  else
    $\hat{e}_k := (e_k/\hat{q}_k)\, q_{k+1}$;
    $d_{k+1} := (d_k/\hat{q}_k)\, q_{k+1}$;
  end if
  if $d_{k+1} + S = S$ **then**
    $d_{k+1} := 0$;
  end if
end for
$\hat{q}_n := d_n$;

4.1.2 Convergence criteria of the DQDS algorithm

There are three methods as convergence criteria.

In the first method, when the element e_k $(k = 1, \ldots, n-1)$ is extremely smaller than the sum of the value of shift S in Equation (45), the e_k can be regarded as 0. In DQDS algorithm, if e_k is equal to 0, the split operation [34] occurs and the given matrix is divided to two matrices. Moreover, when the element e_{n-1} is sufficiently smaller than the element $S+q_n$, the dimension size decreases by one. Thus, by checking $e_k (k = 1, \ldots, n-1)$ and e_{n-1}, the DQDS algorithm is terminated. However, only by using the first method, the number of iterations increases and rounding error accumulates.

In the second method, when the element d_k is extremely smaller than the sum of the value of shift S, the d_k can be regarded as 0. By using the algorithm 9 and $d_k = 0$, q_n becomes 0. Then, if $q_n = 0$, then $\hat{e}_{n-1} = 0$. Thus, the confirmation that d_k is extremely smaller than the sum of the value of shift S leads to convergence. Algorithm 10 shows the DQD method [10, 34], which is based on the DQDS algorithm without shift , with the second convergence method.

In the third method, we use the following recurrence relation,

$$\begin{aligned} d_1 &= q_1, \\ d_k &= q_k \frac{d_{k-1}}{(d_{k-1} + e_{k-1})}, \quad k = 2, \cdots, n, \end{aligned} \tag{67}$$

where the d_k in Equation (67) completely matches the d_k in the DQD algorithm. When the element e_{k-1} is extremely smaller than the element d_{k-1}, e_{k-1} can be regarded as 0. However, in implementation of iteration in the DQD algorithm, it is not efficient in terms of computation time. We notice $\hat{q}_k = d_k + e_k$. If $e_k \leq \varepsilon d_k = \varepsilon\left(\hat{q}_k - e_k\right)$, then e_{k-1} can be regarded as 0. Thus, we check,

$$e_k \leq \frac{\varepsilon}{(1+\varepsilon)}\hat{q}_k, \tag{68}$$

where ε is extremely smaller than 1. Instead of eq.(68), for checking whether e_k is sufficiently small or not, we should use $e_k \leq \varepsilon\hat{q}_k$. When e_k is extremely small, $\hat{e}_k$ is sufficiently small. Therefore, the given matrix can be divided.

4.2 OQDS algorithm for computing singular vectors

In singular value computation, DQDS algorithm and OQDS algorithm are mathematically equivalent. OQDS algorithm is theoretically suitable to compute of singular values with high accuracy in terms of relative errors but contains square root computation which worsen the accuracy of the algorithm. On the other hand, in the DQDS algorithm, square root computation is eliminated by variable transformation. The square root elimination gives DQDS algorithm an advantage against OQDS algorithm in terms of computation speed. However, the DQDS algorithm cannot compute singular vectors because the variable transformation in the formulation of DQDS algorithm destroys the correspondence between the variables and singular vectors. Thus, in this thesis, we use the OQDS algorithm to compute singular vectors.

Let $L^{(i)}$ be an $n \times n$ lower bidiagonal matrix,

$$L^{(i)} = \begin{bmatrix} \alpha_1^{(i)} & & & \\ \beta_1^{(i)} & \alpha_2^{(i)} & & \\ & \ddots & \ddots & \\ & & \beta_{n-1}^{(i)} & \alpha_n^{(i)} \end{bmatrix}, \tag{69}$$

and $U^{(i)}$ be an $n \times n$ upper bidiagonal matrix,

$$U^{(i)} = \begin{bmatrix} \gamma_1^{(i)} & \zeta_1^{(i)} & & \\ & \gamma_2^{(i)} & \ddots & \\ & & \ddots & \zeta_{n-1}^{(i)} \\ & & & \gamma_n^{(i)} \end{bmatrix}, \tag{70}$$

respectively.

In the OQDS algorithm, the following three operations are repeated for $L^{(i)}$ and $U^{(i)}$.

1. Compute a shift $\tau^{(i)}$ satisfying:

$$0 \leq \tau^{(i)} \leq \sigma_{\min}\left(L^{(i)}\right). \tag{71}$$

2. LU step

$$P^{(i)} \begin{bmatrix} L^{(i)} \\ t^{(i)} I_n \end{bmatrix} = \begin{bmatrix} U^{(i)} \\ t^{(i+1)} I_n \end{bmatrix}, \tag{72}$$

$$t^{(i+1)} = \sqrt{\left(t^{(i)}\right)^2 + \left(\tau^{(i)}\right)^2} \tag{73}$$

3. UL step

$$\begin{bmatrix} I_n & O \\ O & \left(Q^{(i)}\right)^\top \end{bmatrix} \begin{bmatrix} U^{(i)} \\ t^{(i+1)} I_n \end{bmatrix} Q^{(i)} = \begin{bmatrix} L^{(i+1)} \\ t^{(i+1)} I_n \end{bmatrix}. \tag{74}$$

The lower half of the matrix is trivial. Thus, only $U^{(i)}Q^{(i)} = L^{(i+1)}$ should be considered.

$P^{(i)}$ and $Q^{(i)}$ are a $2n \times 2n$ orthogonal matrix and an $n \times n$ orthogonal matrix, respectively. $P^{(i)}$ is computed by using the Givens rotation and the generalized Givens rotation [5]. $Q^{(i)}$ consists of the Givens rotation. $t^{(i_0)} I_n$ expresses a diagonal matrix having the same value. Let Σ be a diagonal matrix arranged in descending order of singular values. When $L^{(i_0)}$ and $t^{(i_0)} I_n$ converge to a diagonal matrix E and tI_n, if the split operation dose not occurs, $\Sigma_{kk} = \sqrt{E_{kk}^2 + t^2}$ $(k = 1, \cdots, n)$. To obtain right singular vectors, an orthogonal matrix V is computed with $V = Q^{(0)} \cdots Q^{(i_0 - 1)}$. Using the Givens rotation, we add tI_n to $E = L^{(i_0)}$. Then, $L^{(i_0)}$ and tI_n become Σ and the zero matrix, respectively. The orthogonal matrix U, which consists of left singular vectors, are not required in the Sakurai-Sugiura method. Thus, in this paper, we do not introduce $P^{(i)}$, which is adopted to compute U.

The summation of the value of shift is computed with

$$t^{(i+1)} = \sqrt{\left(t^{(i)}\right)^2 + \left(\tau^{(i)}\right)^2}. \tag{75}$$

To avoid loss of trailing digits, the double-double arithmetic should be adopted like the DQDS algorithm.

When the OQDS algorithm is terminated, if the split operation dose not occurs, the diagonal elements of E are arranged in descending order. Thus, in the case that $\alpha_1^{(i)} < \alpha_n^{(i)}$, by using the matrix Y,

$$Y = \begin{bmatrix} 0 & & & 1 \\ & & \iddots & \\ & \iddots & & \\ 1 & & & 0 \end{bmatrix}, \tag{76}$$

and $U^{(i)} \leftarrow YL^{(i)}Y$, all elements are rearranged in reverse order. $L^{(i)}$ and $U^{(i)}$ are an $n \times n$ lower bidiagonal matrix and an $n \times n$ upper bidiagonal matrix, respectively.

Algorithm 11 The LU step in the case of the value of shift $\tau^{(i)} > 0$

1: $\rho_1^{(i)} := \sqrt{\alpha_1^{(i)} - \tau^{(i)}}\sqrt{\alpha_1^{(i)} + \tau^{(i)}}$;
2: **for** $k := 1, 2, \cdots, n-1$ **do**
3: $\quad \gamma_k^{(i)} := \sqrt{\left(\rho_k^{(i)}\right)^2 + \left(\beta_k^{(i)}\right)^2}$, $\zeta_k^{(i)} := \frac{\beta_k^{(i)}}{\gamma_k^{(i)}}\alpha_{k+1}^{(i)}$;
4: $\quad \rho_{k+1}^{(i)} := \sqrt{\frac{\rho_k^{(i)}}{\gamma_k^{(i)}}\alpha_{k+1}^{(i)} - \tau^{(i)}}\sqrt{\frac{\rho_k^{(i)}}{\gamma_k^{(i)}}\alpha_{k+1}^{(i)} + \tau^{(i)}}$;
5: **end for**
6: $\gamma_n^{(i)} := \rho_n^{(i)}$;

4.2.1 LU and UL steps

The LU step is an operation that compose $L^{(i)}$ to $U^{(i)}$.

$$\begin{bmatrix} \rho_1^{(i)} & & \\ \beta_1^{(i)} & \alpha_2^{(i)} & \\ & \beta_2^{(i)} & \ddots \\ & & \ddots \\ t^{(i)} & & \\ & t^{(i)} & \\ & & \ddots \end{bmatrix} \xrightarrow{\text{O1}} \begin{bmatrix} & \rho_1^{(i)} & & \\ & \beta_1^{(i)} & \alpha_2^{(i)} & \\ & & \beta_2^{(i)} & \ddots \\ & & & \ddots \\ \sqrt{\left(t^{(i)}\right)^2 + \left(u^{(i)}\right)^2} & & & \\ & & t^{(i)} & \\ & & & \ddots \end{bmatrix}$$

$$\xrightarrow{\text{O2}} \begin{bmatrix} & \gamma_1^{(i)} & & \\ & & \zeta_1^{(i)} & \\ & & \rho_2^{(i)} & \\ & & \beta_2^{(i)} & \ddots \\ & & & \ddots \\ \sqrt{\left(t^{(i)}\right)^2 + \left(u^{(i)}\right)^2} & & & \\ & & t^{(i)} & \\ & & & \ddots \end{bmatrix} \xrightarrow{\text{O3}} \tag{77}$$

Algorithm 11 shows the LU step with the value of shift $u^{(i)}$ and elements of $L^{(i)}$ and $U^{(i)}$. In the computation of $\rho_{k+1}^{(i)}$, the fused multiply-add operation can be adopted effectively. Algorithm 12 shows the LU step in the case that $\tau^{(i)} = 0$.

Algorithm 12 The LU step in the case of the value of shift $\tau^{(i)} = 0$

$\rho_1^{(i)} := \alpha_1^{(i)}$;
for $k := 1, 2, \cdots, n-1$ **do**
  $\gamma_k^{(i)} := \sqrt{\left(\rho_k^{(i)}\right)^2 + \left(\beta_k^{(i)}\right)^2}$;
  if $\gamma_k^{(i)} = 0$ **then**
    $\zeta_k^{(i)} := 0,\ \rho_{k+1}^{(i)} := \alpha_{k+1}^{(i)}$;
  else
    $\zeta_k^{(i)} := \left(\beta_k^{(i)} / \gamma_k^{(i)}\right) \alpha_{k+1}^{(i)}$;
    $\rho_{k+1}^{(i)} := \left(\rho_k^{(i)} / \gamma_k^{(i)}\right) \alpha_{k+1}^{(i)}$;
  end if
end for
$\gamma_n^{(i)} := \rho_n^{(i)}$;

The UL step is an operation that composes $U^{(i)}$ to $L^{(i+1)}$ as following:

$$\begin{bmatrix} \gamma_1^{(i)} & \zeta_1^{(i)} & & \\ & \gamma_2^{(i)} & \zeta_2^{(i)} & \\ & & \ddots & \ddots \\ & & & \ddots \\ t^{(i)} & & & \\ & t^{(i)} & & \\ & & \ddots & \\ & & & \ddots \end{bmatrix} \rightarrow \begin{bmatrix} \alpha_1^{(i+1)} & & & \\ \beta_1^{(i+1)} & \eta_2^{(i+1)} & \zeta_2^{(i)} & \\ & & \ddots & \ddots \\ & & & \ddots \\ t^{(i)} & & & \\ & t^{(i)} & & \\ & & \ddots & \\ & & & \ddots \end{bmatrix} \tag{78}$$

The eq.(78) can be carried out using the Givens rotation from the right hand side. Algorithm 13 shows the UL step with elements of $U^{(i)}$ and $L^{(i+1)}$.

In the OQDS algorithm, the Givens rotation is adopted to the LU and UL steps. Usually, in implementation, the Givens rotation can be computed by using `DROTG`, which is a level-1 routine in BLAS [38]. However, the Givens rotation with high precision, which is required from the OQDS algorithm, should be performed by using `DLARTG` in LAPACK.

4.2.2 Shift strategy for the OQDS algorithm

In the OQDS algorithm, the value of shift $u^{(i)}$ is computed using the follows:

1. generalized Rutishauser bound
2. a bound based on the Collatz inequality

Algorithm 13 UL step

1: $\eta_1^{(i)} := \gamma_1^{(i)}$;
2: **for** $k := 1, 2, \cdots, n-1$ **do**
3: $\quad \alpha_k^{(i+1)} := \sqrt{\left(\eta_k^{(i)}\right)^2 + \left(\zeta_k^{(i)}\right)^2}$;
4: $\quad$ **if** $\alpha_k^{(i+1)} = 0$ **then**
5: $\quad\quad \beta_k^{(i+1)} := 0,\ \eta_{k+1}^{(i)} := \gamma_{k+1}^{(i)}$;
6: $\quad$ **else**
7: $\quad\quad \beta_k^{(i+1)} := \left(\zeta_k^{(i)} / \alpha_k^{(i+1)}\right) \gamma_{k+1}^{(i)}$;
8: $\quad\quad \eta_{k+1}^{(i)} := \left(\eta_k^{(i)} / \alpha_k^{(i+1)}\right) \gamma_{k+1}^{(i)}$;
9: $\quad$ **end if**
10: **end for**
11: $\alpha_n^{(i)} := \eta_n^{(i)}$;

3. Johnson bound

Unlike the shift strategy for the DQDS algorithm in Section 4.1.1, no square root computation is required in the bound based on the Collatz inequality. Since the bound based on the Collatz inequality can be computed at high speed, the Laguerre bound, the Newton bound, and the generalized Newton bound that estimate the lower bound more roughly than Collatz method can be excluded. The strategy of the combination of the lower bounds in the OQDS algorithm is the same as that in the DQDS algorithm.

4.3 Convergence criteria of the OQDS algorithm

The first convergence criteria of the DQDS algorithm in Section 4.1.2 can be used as convergence methods of the OQDS algorithm through variable transformation.

In terms of the 2-norm, the third convergence criteria of the DQDS algorithm in Section 4.1.2 is defined. In the OQDS algorithm, in terms of the 1-norm, the convergence criteria are redefined as follows. If, in the following recurrence relation,

$$\begin{aligned} \mu_1 &= \alpha_1^{(i)}, \\ \mu_k &= \alpha_k^{(i)} \frac{\mu_{k-1}}{\left(\mu_{k-1} + \beta_{k-1}^{(i)}\right)}, \quad k = 2, \cdots, n, \end{aligned} \tag{79}$$

$\beta_{k-1}^{(i)}$ is extremely smaller than μ_{k-1}, then $\beta_{k-1}^{(i)}$ can be regarded as 0. As a result of preliminary experiment, the convergence criteria in terms of the 1-norm gives more accurate judgement than that of 2-norm. Thus, we use 1-norm convergence criteria in the subsequent experiments.

4.4 Proposed method for computing column space by using the combination of DQDS and OQDS algorithms

In this section, we propose a fast computation method of column space of the upper bidiagonal matrix. The DQDS algorithm is known as a fast computation method only for singular values. Thus, the DQDS algorithm is adopted to investigate the distribution of all singular values. From the distribution of singular values, the numerical rank is determined. After excluding the number K of singular values, which are very close to 0, from the size n of the matrix, the numerical rank is defined as $n - K$. If the split operation dose not occurs, the diagonal elements of E computed in the OQDS algorithm are arranged in descending order. Here, if off-diagonal elements do not become completely 0 in iterations, the OQDS algorithm may compute some right singular vectors corresponding to from the smallest singular value to K-th singular value. The OQDS algorithm can compute the null space of the lower bidiagonal matrix $L^{(0)}$ as K right singular vectors. In the OQDS algorithm, the computation cost of the null space is cheaper than that of all right singular vectors. While singular vectors corresponding to the null space are computed, the row space can be obtained as the complementary space of the null space. Here, the row space in the lower bidiagonal matrix $L^{(0)}$ is equal to the column space in the upper bidiagonal matrix $\left(L^{(0)}\right)^{\top}$. Therefore, the column space is obtained from the complementary space of K vectors corresponding to from the smallest singular value to K-th singular value in $V = Q^{(0)} \cdots Q^{(i_0-1)}$ of $L^{(0)}$. In other words, the column space is obtained from the first $n - K$-th vectors in V. Finally, singular values computed in the OQDS algorithm should be compared with that computed in the DQDS algorithm.

4.5 Numerical experiment

In order to confirm the effectiveness of the proposed method, we compare the orthogonality of the computed row space and the computation time in the conventional method. As a conventional method, we compute all right singular vectors with the OQDS algorithm and compare with that in the proposed method. Table 4 describes the experimental environment.

Table 4: Experimental Environment

CPU	Intel(R) Core(TM) CPU i3-7100 @ 3.90GHz
RAM	4 GB
OS	Ubuntu 16.04.3 LTS
Compiler	gcc version 5.4.0, gfortran version 5.4.0
Options	-O3 -mtune=native -march=native -Wall -fopenmp
Software	Intel Math Kernel Library 2018

We use the following bidiagonal matrix for the comparison:

$$\sigma_i := \sqrt{\varepsilon^{\frac{i-1}{n-1}}}, \quad i = 1, \cdots, n - t_0 \tag{80}$$

$$\sigma_i := \sqrt{\varepsilon^{2\frac{i-1}{n-1}}}, \quad i = n - t_0 + 1, \cdots, n \tag{81}$$

where σ_i means the value of singular-value. ε is set to $2.22044604925031 \times 10^{-16}$ as the machine epsilon of double precision (binary64) floating-point number defined in IEEE 754 and t_0 is set to 20. The dimension n of the matrix is 128. For this matrix, we can obtain the number K of singular values, which are very close to 0, from the result computed with the DQDS algorithm. From the relative gap $\hat{\sigma}_{i+1}/\hat{\sigma}_i$, where $\hat{\sigma}_i$ is computed as singular values with the DQDS algorithm, K is found to be t_0. Table 5 shows orthogonality $||V^\top V - I||_{\mathsf{F}}$ of the computed row space and the computation time.

Table 5: Orthogonality and Computation Time

	Conventional method	Proposed method
Orthogonality	1.23×10^{-14}	4.76×10^{-15}
Computation Time	5.27×10^{-4}	1.03×10^{-4}

4.6 Discussion about the column space computation method by using the combination of DQDS and OQDS algorithms

We proposed the method computing column vector of bidiagonal matrices by using the combination of DQDS and OQDS algorithms. As the result of the numerical experiment, our proposed combination method gives half digit more accurate orthogonality than conventional method in four times shorter computation time. On the other hand, the use case of this method, subroutine of Sakurai-Sugiura method, is very limited compared with general singular value decomposition. Because of the disadvantage of the convergence speed of OQDS algorithm, it is hard to surpass the performance of conventional singular value computation algorithm in general purpose.

Then, we focus the capability of OQDS algorithm for band matrices again and fumble for general singular value decomposition algorithm which does not requires preliminary reduction.

5 Singular value decomposition using the two-sided Jacobi algorithm

Jacobi algorithm is classical, iterative matrix diagonalization method originally proposed by Jacobi before the advent of computer science. The iteration of the Jacobi algorithm is very simple; eliminating non-diagonal element of matrix with Givens rotation one by

one. By repeating the rotation until the matrix becomes almost diagonal, we obtain the singular values of original matrix as the diagonal elements of rotated matrix because Givens rotation keeps the singular values.

The Jacobi algorithm can be interpreted as the application of OQDS algorithm for general triangular matrix if eliminate non-diagonal elements in an appropriate order. In this section, we discuss the improvements of the Jacobi algorithm from the perspective of the derivative of OQDS algorithm. The Jacobi algorithm is roughly divided into two types; one-sided Jacobi algorithm and two-sided Jacobi algorithm. We first discuss the two-sided Jacobi algorithm that is more intuitive and then, discuss the one-sided Jacobi algorithm.

5.1 Outline of two-sided Jacobi algorithm

Let $J^{(i)}$, $K^{(i)}$, $N^{(i)}$, and $M^{(i)}$ be the products of rotation matrices. Let $R^{(i)}$ and $L^{(i)}$ be real upper and lower triangular matrices, respectively. In a singular value decomposition using the two-sided Jacobi algorithm, the transformations described as Equations (82) and (83) are repeatedly computed.

$$K^{(i)}R^{(i)}J^{(i)} = L^{(i)}, \tag{82}$$

$$N^{(i)}L^{(i)}M^{(i)} = R^{(i+1)}, \quad i = 0, 1, \cdots . \tag{83}$$

By these iterative computations, both $R^{(i)}$ and $L^{(i)}$ converge into a diagonal matrices consist of the singular values of original triangular matrix. In the case of convergence, the left singular vector U and right singular vector V can be computed as follows:

$$U = \left(K^{(0)}\right)^\top \left(N^{(0)}\right)^\top \left(K^{(1)}\right)^\top \left(N^{(1)}\right)^\top \cdots \left(K^{(m-1)}\right)^\top \left(N^{(m-1)}\right)^\top, \tag{84}$$

$$V = J^{(0)}M^{(0)}J^{(1)}M^{(1)}\cdots J^{(m-1)}M^{(m-1)}, \tag{85}$$

where m denotes the iteration number in the case of convergence. Then, U, $\Sigma \coloneqq R^{(m)}$ and V satisfy $U\Sigma V = R^{(0)}$. The matrix multiplication in Equations (84) and (85) is accomplished using Givens rotations.

Figure 6 shows the storing method of $R^{(i)}$ and $L^{(i)}$ as a single upper triangular matrix. By using the storing method of Figure 6, memory allocation is not required for $R^{(i)}$ and $L^{(i)}$ as separate matrices in each iteration. In other words, $R^{(i)}$ and $L^{(i)}$ can be computed using the same memory area.

As shown in Equation (88), $R_{j,k}$ is converted to 0 by using the rotation matrices P

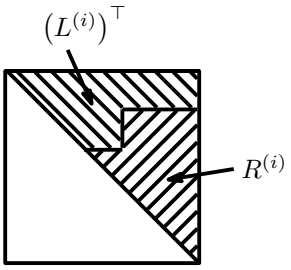


Figure 6: Space sharing of the upper triangular matrix

and Q. I denotes an identity matrix here.

$$P = \begin{bmatrix} I & 0 & \cdots & \cdots & 0 \\ 0 & c_1 & \cdots & s_1 & \vdots \\ \vdots & \vdots & I & \vdots & \vdots \\ \vdots & -s_1 & \cdots & c_1 & 0 \\ 0 & \cdots & \cdots & 0 & I \end{bmatrix}, \tag{86}$$

$$Q = \begin{bmatrix} I & 0 & \cdots & \cdots & 0 \\ 0 & c_2 & \cdots & -s_2 & \vdots \\ \vdots & \vdots & I & \vdots & \vdots \\ \vdots & s_2 & \cdots & c_2 & 0 \\ 0 & \cdots & \cdots & 0 & I \end{bmatrix}, \tag{87}$$

$$\begin{aligned} P \times & \begin{bmatrix} \ddots & \cdots & \cdots & \cdots & \cdots \\ \vdots & R_{j,j} & \cdots & R_{j,k} & \vdots \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \vdots & 0 & \cdots & R_{k,k} & \vdots \\ \cdots & \cdots & \cdots & \cdots & \ddots \end{bmatrix} \times Q \\ & = \begin{bmatrix} \ddots & \cdots & \cdots & \cdots & \cdots \\ \vdots & \hat{R}_{j,j} & \cdots & 0 & \vdots \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \vdots & 0 & \cdots & \hat{R}_{k,k} & \vdots \\ \cdots & \cdots & \cdots & \cdots & \ddots \end{bmatrix}. \end{aligned} \tag{88}$$

Repeating transformations of Equation (88), $R^{(i)}$ can be transformed to lower triangular matrix $L^{(i)}$. Because P and Q are rotation matrices, θ_1 and θ_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 whose values change.

$$\begin{bmatrix} c_1 & s_1 \\ -s_1 & c_1 \end{bmatrix} \begin{bmatrix} R_{j,j} & R_{j,k} \\ 0 & R_{k,k} \end{bmatrix} \begin{bmatrix} c_2 & -s_2 \\ s_2 & c_2 \end{bmatrix} = \begin{bmatrix} \hat{R}_{j,j} & 0 \\ 0 & \hat{R}_{k,k} \end{bmatrix}. \tag{89}$$

To compute $L^{(i)}$ from $R^{(i)}$, the transformation of Equation (89) is repeated many times. In the iterative procedures, an ordering strategy for erasing off-diagonal elements, as explained in Section 5.2, is adopted. We also use same procedure to obtain $R^{(i+1)}$ from $L^{(i)}$.

The detail of the computation of c_1, s_1, c_2, s_2, $\hat{R}_{j,j}$, and $\hat{R}_{k,k}$ from $R_{j,j}, R_{j,k}$ and $R_{k,k}$ is explained in Sections 5.3, 5.4, and 5.5.

5.2 Ordering strategy and convergence criterion

In the iteration of the Jacobi algorithm, the off-diagonal elements in an upper triangular matrix $R^{(i)}$ are reduced to 0. By the operation, some non-zero elements appear in the lower triangular part and that part is set as the off-diagonal elements of lower triangular matrix $L^{(i)}$.

The details are described as follows:

- If $\left|R^{(0)}_{1,1}\right| \geq \left|R^{(0)}_{n,n}\right|$, we use the following strategy.

  The off-diagonal elements are reduced to 0 in the following order of elements: $(1,2), (1,3), \cdots, (1,n), (2,3), (2,4), \cdots, (n-2,n), (n-1,n)$. Subsequently, the off-diagonal elements in the lower triangular matrix $L^{(i)}$ are reduced to 0 in the following order of elements: $(2,1), (3,1), \cdots, (n,1), (3,2), (4,2), \cdots, (n,n-2), (n,n-1)$.

- If $\left|R^{(0)}_{1,1}\right| < \left|R^{(0)}_{n,n}\right|$, we use the following strategy.

  The off-diagonal elements are reduced to 0 in the following order of elements: $(n-1,n), (n-2,n), \cdots, (1,n), (n-2,n-1), (n-3,n-1), \cdots, (1,3), (1,2)$. Subsequently, the off-diagonal elements in the lower triangular matrix $L^{(i)}$ are reduced to 0 in the following order of elements: $(n,n-1), (n,n-2), \cdots, (n,1), (n-1,n-2), (n-1,n-3), \cdots, (3,1), (2,1)$.

In other words, eliminate off-diagonal elements in the order of column prior static traveling from top left edge if the absolute value of the top left edge is larger than another edge, otherwise row prior static traveling from bottom right edge. The conditional instruction means that larger edge of diagonal element and its subsidiary off-diagonal elements are processed at first in order to help sorting of diagonal elements which introduced in Section 5.8.

In practical, we must stop the iteration along with an appropriate convergence criteria where the off-diagonal elements are sufficiently small but not exact 0. The proposed algorithm treat the element $R_{j,k}$ as 0 if Equation (90) is satisfied,

$$|R_{j,k}| \leq \varepsilon \sqrt{|R_{j,j}|} \times \sqrt{|R_{k,k}|}. \tag{90}$$

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

5.3 Implementation method using the arctangent function

Unlike in the one-sided Jacobi algorithm, the singular value decomposition using the two-sided Jacobi algorithm requires many operations to calculate c_1, s_1, c_2, and s_2.

5.3.1 Conventional implementation

We introduce the Upper-triangular, Left transformation first (UL) algorithm and the Upper-triangular, Right transformation first (UR) algorithm for computing c_1, s_1, c_2, and s_2 as a conventional method [25]. If $(R_{j,j} - R_{k,k})(R_{j,j} + R_{k,k})$ is greater than or equal to 0, the UL algorithm should be adopted as follows.

$$\theta_1 = \frac{1}{2}\tan^{-1}\left(\frac{2R_{k,k}}{(R_{j,j} - R_{k,k})(R_{j,j} + R_{k,k})/R_{j,k} + R_{j,k}}\right), \tag{91}$$

$$\theta_2 = \tan^{-1}\left(\frac{\underline{\underline{R_{j,k} + R_{k,k}\tan(\theta_1)}}}{R_{j,j}}\right). \tag{92}$$

Here, $-\frac{\pi}{4} \leq \theta_1 \leq \frac{\pi}{4}$ and $-\frac{\pi}{2} \leq \theta_2 \leq \frac{\pi}{2}$. Conversely, if $(R_{j,j} - R_{k,k})(R_{j,j} + R_{k,k})$ is negative, the UR algorithm should be adopted as follows.

$$\theta_2 = \frac{1}{2}\tan^{-1}\left(\frac{2R_{j,j}}{(R_{j,j} - R_{k,k})(R_{j,j} + R_{k,k})/R_{j,k} - R_{j,k}}\right), \tag{93}$$

$$\theta_1 = \tan^{-1}\left(\frac{\underline{\underline{R_{j,j}\tan(\theta_2) - R_{j,k}}}}{R_{k,k}}\right). \tag{94}$$

Here, $-\frac{\pi}{2} \leq \theta_1 \leq \frac{\pi}{2}$ and $-\frac{\pi}{4} \leq \theta_2 \leq \frac{\pi}{4}$. Then, we compute c_1, s_1, c_2, and s_2.

$$c_1 = \cos\theta_1,\ s_1 = \sin\theta_1, \tag{95}$$

$$c_2 = \cos\theta_2,\ s_2 = \sin\theta_2. \tag{96}$$

Then, the computed c_1, s_1, c_2, and s_2 are substituted in Equations (97) and (98);

$$u = c_1 + c_2, \tag{97}$$

$$\hat{R}_{j,j} = \underline{\underline{R_{j,j} + \frac{s_2}{u} \times R_{j,k}}},\ \hat{R}_{k,k} = \underline{\underline{R_{k,k} - \frac{s_1}{u} \times R_{j,k}}}. \tag{98}$$

For the double underlined part of the equations, FMA operation can be adopted. The operation reduces the error of the final result by performing a product–sum operation in one instruction without rounding up the integration in the middle and is important for the achievement of high accuracy.

5.3.2 Fused multiply-add operaion

Before the proposal of the implementation of the two-sided Jacobi algorithm, we introduce a brief summary of the FMA operation. The FMA is defined in IEEE 754, performs floating point multiply and add written as follows:

$$a \leftarrow a + (b \times c)$$

in one step, with a single rounding while the conventional floating point number operation requires two steps and two roundings[21]. It means that the FMA operation is fast and more accurate than the conventional floating point number operation. Our proposed implementation described in the following section intends to utilize the FMA operation in the calculation of the Givens transformation.

5.3.3 Proposed implementation

In numerical computations, performing such a large number of operations introduces numerous errors into the variables under computation. Therefore, we propose to implement a method using the arctangent function. In the proposed implementation, the number of operations for computing c_1, s_1, c_2, and s_2 is decreased by using the intermediate variables α and β.

Let

$$\alpha = \tan^{-1}\left(\frac{R_{j,k}}{R_{j,j} - R_{k,k}}\right), \tag{99}$$

$$\beta = \tan^{-1}\left(\frac{-R_{j,k}}{R_{j,j} + R_{k,k}}\right), \tag{100}$$

then c_1, s_1, c_2, and s_2 are computed using $\tan^{-1}$, θ_1, and θ_2 as follows:

$$\theta_1 = \frac{1}{2}(\alpha + \beta),\ \theta_2 = \frac{1}{2}(\alpha - \beta), \tag{101}$$

$$c_1 = \cos\theta_1,\ s_1 = \sin\theta_1, \tag{102}$$

$$c_2 = \cos\theta_2,\ s_2 = \sin\theta_2, \tag{103}$$

where $-\frac{\pi}{2} \leq \theta_1 \leq \frac{\pi}{2}$ and $-\frac{\pi}{2} \leq \theta_2 \leq \frac{\pi}{2}$. Then, the computed c_1, s_1, c_2, and s_2 are substituted in Equations (97) and (98); We define the above computation method as the faster version of the proposed implementation. The FMA operation can be adopted in the double underlined part of the equations.

In order to reduce the error of the decomposition $||A - U\Sigma V^\top||_\mathsf{F}$, we use the accurate

version of the proposed implementation as follows:

$$\alpha = \tan^{-1}\left(\frac{R_{j,k}}{R_{j,j} - R_{k,k}}\right), \tag{104}$$

$$\beta = \tan^{-1}\left(\frac{-R_{j,k}}{R_{j,j} + R_{k,k}}\right), \tag{105}$$

$$\theta_1 = \frac{1}{2}(\alpha + \beta),\ \theta_2 = \frac{1}{2}(\alpha - \beta), \tag{106}$$

$$v_1 = \cos\theta_1,\ w_1 = \sin\theta_1, \tag{107}$$

$$v_2 = \cos\theta_2,\ w_2 = \sin\theta_2. \tag{108}$$

We compute c_1 and s_1 using the Givens rotation for $x \leftarrow v_1$ and $y \leftarrow w_1$ and compute c_2 and s_2 using the Givens rotation for $x \leftarrow v_2$ and $y \leftarrow w_2$ in Section 5.5. Finally, Equations (97) and (98)can be computed.

5.4 Implementation method by Rutishauser

Integrating the implementation method proposed by Rutishauser [39] with the two-sided Jacobi algorithm for singular value decomposition, the FMA, which can achieve high accuracy, prevents the introduction of errors in c_1, s_1, c_2, and s_2.

In addition, t_1 and t_2 are decided as follows:

$$\gamma_1 = \frac{R_{j,j} - R_{k,k}}{R_{j,k}},\ t_1 = \frac{1}{\gamma_1 + \mathrm{SIGN}\left(\sqrt{\underline{\underline{1 + (\gamma_1)^2}}}, \gamma_1\right)}, \tag{109}$$

$$\gamma_2 = -\frac{R_{j,j} + R_{k,k}}{R_{j,k}},\ t_2 = \frac{1}{\gamma_2 + \mathrm{SIGN}\left(\sqrt{\underline{\underline{1 + (\gamma_2)^2}}}, \gamma_2\right)}, \tag{110}$$

where the function $\mathrm{SIGN}(A, B)$ returns the value of A with the sign of B. Subsequently, using t_1 and t_2,

$$\hat{v}_1 = \underline{\underline{1 - t_1 \times t_2}},\ \hat{w}_1 = t_1 + t_2, \tag{111}$$

and

$$\hat{v}_2 = \underline{\underline{1 + t_1 \times t_2}},\ \hat{w}_2 = t_1 - t_2, \tag{112}$$

are computed. We compute c_1 and s_1 using the Givens rotation for $x \leftarrow \hat{v}_1$ and $y \leftarrow \hat{w}_1$ and compute c_2 and s_2 using the Givens rotation for $x \leftarrow \hat{v}_2$ and $y \leftarrow \hat{w}_2$ as described in Section 5.5. Finally, Equations (97) and (98) can be computed

5.5 Implementation method using the Givens rotation

Consider the Givens rotation denoted as follows:

$$\cos\theta = \frac{x}{\sqrt{x^2+y^2}}, \sin\theta = \frac{y}{\sqrt{x^2+y^2}}. \tag{113}$$

Here, we executed Algorithm 14 to compute $\cos\theta$, $\sin\theta$, and $\sqrt{x^2+y^2}$. To avoid overflow and underflow, the Givens rotation should be implemented as Algorithm 14. If $\sqrt{x^2+y^2}$ is not required, $\sqrt{x^2+y^2} \leftarrow r \times f$ and $\sqrt{x^2+y^2} \leftarrow r \times g$ are not computed.

Algorithm 14 Implementation of the Givens rotation

1: **if** $x = 0$ **and** $y = 0$ **then**
2: $c \leftarrow 1$
3: $s \leftarrow 0$
4: $\sqrt{x^2+y^2} \leftarrow 0$
5: **else**
6: $f \leftarrow |x|$
7: $g \leftarrow |y|$
8: **if** $f \geq g$ **then**
9: $v \leftarrow y/f$
10: $r \leftarrow \sqrt{\underline{\underline{1+v^2}}}$
11: $c \leftarrow \mathrm{SIGN}(1/r, x)$
12: $s \leftarrow v/r$
13: $\sqrt{x^2+y^2} \leftarrow r \times f$
14: **else**
15: $u \leftarrow x/g$
16: $r \leftarrow \sqrt{\underline{\underline{1+u^2}}}$
17: $c \leftarrow u/r$
18: $s \leftarrow \mathrm{SIGN}(1/r, y)$
19: $\sqrt{x^2+y^2} \leftarrow r \times g$
20: **end if**
21: **end if**

The FMA can be adopted in the double-underlined part of lines 10 and 16 of Algorithm 14.

Algorithm 15 was executed to compute c_1, s_1, c_2, and s_2. Here, the function $\mathrm{SIGN}(A, B)$ returns the value of A with the sign of B. Subsequently, Equations (97) and (98) are computed.

5.6 Comparing the number of operations

Table 6 summarizes the comparison of the number of operations for computing c_1, s_1, c_2, and s_2 using the method explained the former sections.

Table 6: Comparing the number of operations

	conventional $\tan^{-1}$	fast $\tan^{-1}$	accurate $\tan^{-1}$	Rutishauser	Givens
add/sub	4	5	5	7	5
multiply	3	2	2	0	2
division	5	4	10	12	11
FMA	3	2	4	8	10
sqrt	0	0	2	4	4
$\tan^{-1}$	2	2	2	0	0
cos	2	2	2	0	0
sin	2	2	2	0	0

5.7 Implementation technique for a summation

If $|x_0|$ is considerably greater than $|x_i|(i = 1, \cdots, q)$, the method based on $T_q = \sum_{i=1}^{q} x_i$ and $S_q = x_0 + T_q$ is appropriate to compute $S_q = \sum_{i=0}^{q} x_i$. The computation process is adopted in Equation (98).

5.8 Introducing sort feature to the two-sided Jacobi algorithm

Unlike OQDS or DQDS algorithm, the elements of the obtained diagonal matrix of the general Jacobi algorithm are in an irregular order. However, in the convergence process of OQDS and DQDS algorithm, we found that the diagonal elements are sorted in the early stage and then, off-diagonal elements get smaller along with the corresponding diagonal element. We assume the sorting feature accelerates the convergence of LR-based singular value decomposition algorithm, and then try to introduce the feature to the Jacobi algorithm similar to OQDS algorithm.

From Equation (90), if both diagonal elements of 2×2 matrix are large, the non-diagonal elements are considered to be converging even if they are somewhat large. Thus, when the large diagonal elements are collected in the upper left corner, the singular values in the diagonal elements converge from the upper left to the lower right. As a result, it is possible to separate the entries that converge at the beginning from those that converge later. Table 7 summarizes the selection of the angle of Givens rotation that collect large diagonal elements to upper left and smaller elements to lower right.

In the case $\left|R_{1,1}^{(0)}\right| \geq \left|R_{n,n}^{(0)}\right|$, if $\left|\hat{R}_{j,j}\right| < \left|\hat{R}_{k,k}\right|$ 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 $\left|\hat{R}_{j,j}\right| < \left|\hat{R}_{k,k}\right|$ 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 $\left|\hat{R}_{j,j}\right| < \left|\hat{R}_{k,k}\right|$ 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 $\left|\hat{R}_{j,j}\right| < \left|\hat{R}_{k,k}\right|$ 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 from or added to both θ_1 and θ_2, we set $\hat{R}_{j,j} \leftarrow \hat{R}_{k,k}$, $\hat{R}_{k,k} \leftarrow \hat{R}_{j,j}$. Otherwise,

Table 7: Case list for function of sorting

cases								setting
$R^{(0)}_{1,1}$	$\geq$	$R^{(0)}_{n,n}$		$\hat{R}_{j,j}$	$<$	$\hat{R}_{k,k}$	$s_1 > 0$	$c_1 \leftarrow s_1$ and $s_1 \leftarrow -c_1$
$R^{(0)}_{1,1}$	$\geq$	$R^{(0)}_{n,n}$		$\hat{R}_{j,j}$	$<$	$\hat{R}_{k,k}$	$s_1 \leq 0$	$c_1 \leftarrow -s_1$ and $s_1 \leftarrow c_1$
$R^{(0)}_{1,1}$	$\geq$	$R^{(0)}_{n,n}$		$\hat{R}_{j,j}$	$<$	$\hat{R}_{k,k}$	$s_2 > 0$	$c_2 \leftarrow s_2$ and $s_2 \leftarrow -c_2$
$R^{(0)}_{1,1}$	$\geq$	$R^{(0)}_{n,n}$		$\hat{R}_{j,j}$	$<$	$\hat{R}_{k,k}$	$s_2 \leq 0$	$c_2 \leftarrow -s_2$ and $s_2 \leftarrow c_2$
$R^{(0)}_{1,1}$	$<$	$R^{(0)}_{n,n}$		$\hat{R}_{j,j}$	$\geq$	$\hat{R}_{k,k}$	$s_1 > 0$	$c_1 \leftarrow s_1$ and $s_1 \leftarrow -c_1$
$R^{(0)}_{1,1}$	$<$	$R^{(0)}_{n,n}$		$\hat{R}_{j,j}$	$\geq$	$\hat{R}_{k,k}$	$s_1 \leq 0$	$c_1 \leftarrow -s_1$ and $s_1 \leftarrow c_1$
$R^{(0)}_{1,1}$	$<$	$R^{(0)}_{n,n}$		$\hat{R}_{j,j}$	$\geq$	$\hat{R}_{k,k}$	$s_2 > 0$	$c_2 \leftarrow s_2$ and $s_2 \leftarrow -c_2$
$R^{(0)}_{1,1}$	$<$	$R^{(0)}_{n,n}$		$\hat{R}_{j,j}$	$\geq$	$\hat{R}_{k,k}$	$s_2 \leq 0$	$c_2 \leftarrow -s_2$ and $s_2 \leftarrow c_2$

we set $\hat{R}_{j,j} \leftarrow -\hat{R}_{k,k}$, $\hat{R}_{k,k} \leftarrow -\hat{R}_{j,j}$.

In the case $\left|R^{(0)}_{1,1}\right| < \left|R^{(0)}_{n,n}\right|$, if $\left|\hat{R}_{j,j}\right| \geq \left|\hat{R}_{k,k}\right|$ 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 $\left|\hat{R}_{j,j}\right| \geq \left|\hat{R}_{k,k}\right|$ 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 $\left|\hat{R}_{j,j}\right| \geq \left|\hat{R}_{k,k}\right|$ 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 $\left|\hat{R}_{j,j}\right| \geq \left|\hat{R}_{k,k}\right|$ 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 from or added to both θ_1 and θ_2, we set $\hat{R}_{j,j} \leftarrow \hat{R}_{k,k}$, $\hat{R}_{k,k} \leftarrow \hat{R}_{j,j}$. Otherwise, we set $\hat{R}_{j,j} \leftarrow -\hat{R}_{k,k}$, $\hat{R}_{k,k} \leftarrow -\hat{R}_{j,j}$.

By introducing the selection, the two-sided Jacobi algorithm get sorting feature of singular values in descending order. Notably, c_1 and c_2 are kept nonnegative.

5.9 Correction of c_1, s_1, c_2, or s_2

In order to achieve a higher accuracy for the Rutishauser's implementation method[39], we correct the values of c_1, s_1, c_2, or s_2 by using root finding method for the constraints of trigonometric function.

Let $\tilde{c}$, $\tilde{s}$, $\hat{c}$, and $\hat{s}$ be four variables that represent c_1 and c_2, s_1 and s_2, correction result for c_1 and c_2, and correction result for s_1 and s_2, respectively.

5.9.1 False position method

False position method is a classical root finding method depicted in Figure 7. As 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 given by Equation (114) is set as

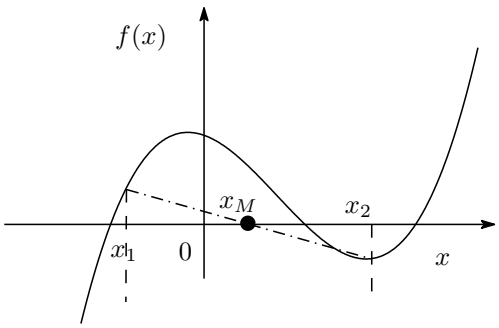


Figure 7: False position method

the next position to compute the real root x in $f(x) = 0$. One has the following:

$$x_M = \frac{x_1 \times f(x_2) - x_2 \times f(x_1)}{f(x_2) - f(x_1)}. \tag{114}$$

Here, if the sign of $f(x_1)$ is the same as that of $f(x_M)$, then $x_1 \leftarrow x_M$. Otherwise, if the sign of $f(x_2)$ is the same as that of $f(x_M)$, then $x_2 \leftarrow x_M$. Figure 7 represents the example case that x_M is set as the next value of x_1.

5.9.2 Secant method

Secant method is also used for the correction. The method is depicted in Figure 8. In

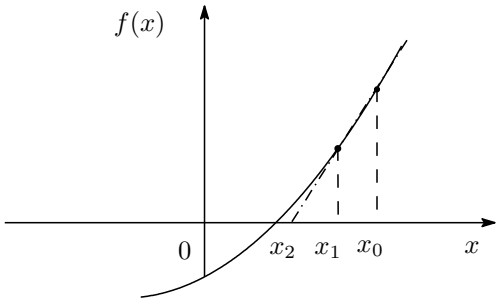


Figure 8: Secant method

secant method, x the real root of $f(x) = 0$ is given by the the following recurrence formula:

$$\begin{aligned} x_{n+1} &= x_n - f(x_n) \times \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} \\ &= \frac{x_{n-1} f(x_n) - x_n f(x_{n-1})}{f(x_n) - f(x_{n-1})}. \end{aligned} \tag{115}$$

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. Let $n = 2$ then Equation (115) is equal to Equation (114).

5.9.3 Correction method

Theoretically, for $c = \cos\theta$ and $s = \sin\theta$ obtained by the computation for the angle of Givens rotation, $c^2 + s^2 = 1$ is always satisfied. However, floating point number operation of finite length may produces rounding error and breaks the constraint. Correction method for c and s is introduced to minimize the rounding error by using root finding methods for the trigonometric constraint.

Assuming that s is correct, $\tilde{c}$ is calculated as follows:

$$\tilde{c}^2 + s^2 = 1. \tag{116}$$

On the other hand, assuming that c is correct, $\tilde{s}$ is computed as follows:

$$c^2 + \tilde{s}^2 = 1. \tag{117}$$

Equations (116) and (117) can be appropriately used by introducing $c = \cos\theta$ and $s = \sin\theta$. For the case where $-\frac{\pi}{4} \leq \theta \leq \frac{\pi}{4}$, Equation (116) is used, whereas Equation (117) is adopted if $\frac{\pi}{4} < \theta \leq \frac{\pi}{2}$ or $-\frac{\pi}{2} \leq \theta < -\frac{\pi}{4}$. For the singular value decomposition using the two-sided Jacobi algorithm, $c \geq 0$ is satisfied. Then, we can assume that $-\frac{\pi}{2} \leq \theta \leq \frac{\pi}{2}$.

When both the nonlinear single equation $f(x) = 0$ and initial numbers x_0 and x_1 are given, then x_2, which is the result of one iteration of the secant method, is equal to x_M achieved using the false position method. One has the following relation:

$$x_2 = \frac{x_0 f(x_1) - x_1 f(x_0)}{f(x_1) - f(x_0)}, \tag{118}$$

In the case where $-\frac{\pi}{4} \leq \theta \leq \frac{\pi}{4}$, $\tilde{c}$ is recomputed using Equation (116). The case where $-\frac{\pi}{4} \leq \theta \leq \frac{\pi}{4}$ can be considered as $c \geq |s|$. To compute $\tilde{c}$, the initial numbers are set to $x_0 = 1$ and $x_1 = c$. When $f(x) = x^2 + s^2 - 1$,

$$\tilde{c} = \frac{(c^2 + s^2 - 1) - cs^2}{(c^2 + s^2 - 1) - s^2} = 1 - s \times \frac{s}{1 + c}, \tag{119}$$

is obtained, where $\tilde{c}$ is more appropriate for satisfying $f(x) = x^2 + s^2 - 1$. The Givens rotation for vectors $\mathbf{x}$ and $\mathbf{y}$ is defined as follows:

$$\mathbf{x} \leftarrow \tilde{c}\mathbf{x} + s\mathbf{y},\ \mathbf{y} \leftarrow -s\mathbf{x} + \tilde{c}\mathbf{y}. \tag{120}$$

When not using c but $\tilde{c}$, Equations (97) and (98) can be written as

$$z_1 = \frac{s}{1+c}, \tag{121}$$

$$\mathbf{x} \leftarrow (1 - s \times z_1)\,\mathbf{x} + s\mathbf{y} = \underline{\underline{s\left(\underline{\underline{-z_1\mathbf{x} + \mathbf{y}}}\right) + \mathbf{x}}}, \tag{122}$$

$$\mathbf{y} \leftarrow -s\mathbf{x} + (1 - s \times z_1)\,\mathbf{y} = \underline{\underline{-s\left(\underline{\underline{z_1\mathbf{y} + \mathbf{x}}}\right) + \mathbf{y}}}. \tag{123}$$

The case wherein $\frac{\pi}{4} < \theta \leq \frac{\pi}{2}$ can be regarded as $c < |s|$ and $s \geq 0$. To compute $\tilde{s}$, the initial numbers are set to $x_0 = 1$ and $x_1 = s$. When $f(x) = c^2 + x^2 - 1$,

$$\tilde{s} = \frac{(c^2 + s^2 - 1) - sc^2}{(c^2 + s^2 - 1) - (c^2)} = 1 - c \times \frac{c}{1+s}, \tag{124}$$

is obtained, where $\tilde{s}$ is more appropriate for satisfying $f(x) = c^2 + x^2 - 1$. When not using s but $\tilde{s}$, the Givens rotation for vectors $\mathbf{x}$ and $\mathbf{y}$ is defined as follows:

$$z_2 = \frac{c}{1+s}, \tag{125}$$

$$\mathbf{x} \leftarrow c\mathbf{x} + (1 - c \times z_2)\,\mathbf{y} = \underline{\underline{c\left(\underline{\underline{-z_2\mathbf{y} + \mathbf{x}}}\right) + \mathbf{y}}}, \tag{126}$$

$$\mathbf{y} \leftarrow -(1 - c \times z_2)\,\mathbf{x} + c\mathbf{y} = \underline{\underline{c\left(\underline{\underline{z_2\mathbf{x} + \mathbf{y}}}\right) - \mathbf{x}}}. \tag{127}$$

The case where $-\frac{\pi}{2} \leq \theta < -\frac{\pi}{4}$ can be regarded as equivalent to $c < |s|$ and $s \leq 0$. To compute $\tilde{s}$, the initial numbers are set to $x_0 = -1$ and $x_1 = s$. When $f(x) = c^2 + x^2 - 1$,

$$\tilde{s} = \frac{-(c^2 + s^2 - 1) - sc^2}{(c^2 + s^2 - 1) - (c^2)} = -1 + c \times \frac{c}{1-s}, \tag{128}$$

is obtained, where $\tilde{s}$ is more appropriate for satisfying $f(x) = c^2 + x^2 - 1$. When not using s but $\tilde{s}$, the Givens rotation for vectors $\mathbf{x}$ and $\mathbf{y}$ is defined as follows:

$$z_3 = \frac{c}{1-s}, \tag{129}$$

$$\mathbf{x} \leftarrow c\mathbf{x} + (-1 + c \times z_3)\,\mathbf{y} = \underline{\underline{c\left(\underline{\underline{z_3\mathbf{y} + \mathbf{x}}}\right) - \mathbf{y}}}, \tag{130}$$

$$\mathbf{y} \leftarrow -(-1 + c \times z_3)\,\mathbf{x} + c\mathbf{y} = \underline{\underline{c\left(\underline{\underline{-z_3\mathbf{x} + \mathbf{y}}}\right) + \mathbf{x}}}. \tag{131}$$

Notably, the FMA can be adopted in the double underlined part.

5.10 Numerical experiments

In order to confirm the two-sided Jacobi algorithm (arctan and Rutishauser versions) has a shorter computation time and a higher accuracy than those of the one-sided Jacobi algorithm implemented in LAPACK [2], We perform numerical experiments. The experimental environment is mentioned in Table 8. We used the following eight matrices

Table 8: Experimental environment

CPU	Intel(R) Xeon(R) Silver 4116 @ 2.10 GHz (2 CPUs)
RAM	192 GB
OS	Ubuntu 20.04.1 LTS
Compiler	gfortran 9.3.0
Options	-O3 -mtune=native -march=native
Software	Lapack 3.9.0
Precision	single precision

for the comparison:

- A_1 (dimension size: 500×500, an upper triangular matrix)
- A_2 (dimension size: 1000×1000, an upper triangular matrix)
- A_3 (dimension size: 1500×1500, an upper triangular matrix)
- A_4 (dimension size: 2000×2000, an upper triangular matrix)

and

- A_5 (dimension size: 500×500, an upper triangular matrix)
- A_6 (dimension size: 1000×1000, an upper triangular matrix)
- A_7 (dimension size: 1500×1500, an upper triangular matrix)
- A_8 (dimension size: 2000×2000, an upper triangular matrix)

For A_1, A_2, A_3, and A_4, all the elements are set to random numbers $\in [0, 1]$ generated using a uniform random number generator. For A_5, A_6, A_7, and A_8, all the elements are set to 1. The results are shown in Table 9, Table 10 and Table 11.

5.11 Discussion about the proposed implementation of two-sided Jacobi algorithm

The results shown in Table 9, without the incorporation of our implementation technique which is introduced in Section 5.7, 5.8, and 5.9, represents that the proposed method achieved better performance than the proposed method in terms of the computation

Table 9: Comparison of Jacobi SVD algorithms (without the incorporation of our implementation technique)

	One-sided Jacobi	Two-sided Jacobi Conventional ($\tan^{-1}$)	Two-sided Jacobi Proposed (faster, $\tan^{-1}$)
A_1			
$\|\|U^\top U - I\|\|_\mathsf{F}$	$1.91 * 10^{-4}$	$1.11 * 10^{-4}$	$1.12 * 10^{-4}$
$\|\|V^\top V - I\|\|_\mathsf{F}$	$8.20 * 10^{-5}$	$1.03 * 10^{-4}$	$1.03 * 10^{-4}$
$\|\|A - U\Sigma V^\top\|\|_\mathsf{F}$	$6.33 * 10^{-4}$	$5.72 * 10^{-4}$	$5.69 * 10^{-4}$
Computation time[s]	1.140	0.882	0.741
A_2			
$\|\|U^\top U - I\|\|_\mathsf{F}$	$5.51 * 10^{-4}$	$3.18 * 10^{-4}$	$3.21 * 10^{-4}$
$\|\|V^\top V - I\|\|_\mathsf{F}$	$1.78 * 10^{-4}$	$2.87 * 10^{-4}$	$2.95 * 10^{-4}$
$\|\|A - U\Sigma V^\top\|\|_\mathsf{F}$	$2.54 * 10^{-3}$	$2.27 * 10^{-3}$	$2.53 * 10^{-3}$
Computation time[s]	10.246	8.160	6.358
A_3			
$\|\|U^\top U - I\|\|_\mathsf{F}$	$1.00 * 10^{-3}$	$6.25 * 10^{-4}$	$6.11 * 10^{-4}$
$\|\|V^\top V - I\|\|_\mathsf{F}$	$2.81 * 10^{-4}$	$5.62 * 10^{-4}$	$5.54 * 10^{-4}$
$\|\|A - U\Sigma V^\top\|\|_\mathsf{F}$	$4.12 * 10^{-3}$	$4.34 * 10^{-3}$	$4.96 * 10^{-3}$
Computation time[s]	38.645	35.668	28.868
A_4			
$\|\|U^\top U - I\|\|_\mathsf{F}$	$1.50 * 10^{-3}$	$9.70 * 10^{-4}$	$9.64 * 10^{-4}$
$\|\|V^\top V - I\|\|_\mathsf{F}$	$3.91 * 10^{-4}$	$9.20 * 10^{-4}$	$9.11 * 10^{-4}$
$\|\|A - U\Sigma V^\top\|\|_\mathsf{F}$	$6.77 * 10^{-3}$	$7.13 * 10^{-3}$	$7.15 * 10^{-3}$
Computation time[s]	92.253	133.499	113.791
A_5			
$\|\|U^\top U - I\|\|_\mathsf{F}$	$2.01 * 10^{-4}$	$1.47 * 10^{-4}$	$1.50 * 10^{-4}$
$\|\|V^\top V - I\|\|_\mathsf{F}$	$9.40 * 10^{-5}$	$1.66 * 10^{-4}$	$1.68 * 10^{-4}$
$\|\|A - U\Sigma V^\top\|\|_\mathsf{F}$	$9.22 * 10^{-4}$	$7.34 * 10^{-4}$	$9.32 * 10^{-4}$
Computation time[s]	0.989	0.993	0.922
A_6			
$\|\|U^\top U - I\|\|_\mathsf{F}$	$5.29 * 10^{-4}$	$4.67 * 10^{-4}$	$4.71 * 10^{-4}$
$\|\|V^\top V - I\|\|_\mathsf{F}$	$2.18 * 10^{-4}$	$5.18 * 10^{-4}$	$5.25 * 10^{-4}$
$\|\|A - U\Sigma V^\top\|\|_\mathsf{F}$	$2.03 * 10^{-3}$	$2.56 * 10^{-3}$	$2.80 * 10^{-3}$
Computation time[s]	8.402	8.556	8.253
A_7			
$\|\|U^\top U - I\|\|_\mathsf{F}$	$1.00 * 10^{-3}$	$9.38 * 10^{-4}$	$9.30 * 10^{-4}$
$\|\|V^\top V - I\|\|_\mathsf{F}$	$3.84 * 10^{-4}$	$1.05 * 10^{-3}$	$1.04 * 10^{-3}$
$\|\|A - U\Sigma V^\top\|\|_\mathsf{F}$	$3.46 * 10^{-3}$	$4.31 * 10^{-3}$	$4.94 * 10^{-3}$
Computation time[s]	30.260	39.966	39.056
A_8			
$\|\|U^\top U - I\|\|_\mathsf{F}$	$1.59 * 10^{-3}$	$1.54 * 10^{-3}$	$1.54 * 10^{-3}$
$\|\|V^\top V - I\|\|_\mathsf{F}$	$5.20 * 10^{-4}$	$1.69 * 10^{-3}$	$1.70 * 10^{-3}$
$\|\|A - U\Sigma V^\top\|\|_\mathsf{F}$	$5.95 * 10^{-3}$	$7.51 * 10^{-3}$	$7.15 * 10^{-3}$
Computation time[s]	78.082	154.763	150.870

Table 10: Comparison of Jacobi SVD algorithms (with the incorporation of our implementation technique)

	One-sided Jacobi	Two-sided Jacobi Conventional ($\tan^{-1}$)	Two-sided Jacobi Proposed (faster,$\tan^{-1}$)	Two-sided Jacobi Proposed (accurate,$\tan^{-1}$)
A_1				
$\|\|U^\top U - I\|\|_\mathsf{F}$	$1.91 * 10^{-4}$	$4.36 * 10^{-5}$	$4.30 * 10^{-5}$	$4.33 * 10^{-5}$
$\|\|V^\top V - I\|\|_\mathsf{F}$	$8.20 * 10^{-5}$	$4.32 * 10^{-5}$	$4.32 * 10^{-5}$	$4.31 * 10^{-5}$
$\|\|A - U\Sigma V^\top\|\|_\mathsf{F}$	$6.33 * 10^{-4}$	$3.74 * 10^{-4}$	$3.69 * 10^{-4}$	$3.70 * 10^{-4}$
Computation time[s]	1.140	0.819	0.712	0.698
A_2				
$\|\|U^\top U - I\|\|_\mathsf{F}$	$5.51 * 10^{-4}$	$8.90 * 10^{-5}$	$8.87 * 10^{-5}$	$8.92 * 10^{-5}$
$\|\|V^\top V - I\|\|_\mathsf{F}$	$1.78 * 10^{-4}$	$8.65 * 10^{-5}$	$8.65 * 10^{-5}$	$8.65 * 10^{-5}$
$\|\|A - U\Sigma V^\top\|\|_\mathsf{F}$	$2.54 * 10^{-3}$	$1.07 * 10^{-3}$	$1.06 * 10^{-3}$	$1.08 * 10^{-3}$
Computation time[s]	10.246	6.325	6.191	6.030
A_3				
$\|\|U^\top U - I\|\|_\mathsf{F}$	$1.00 * 10^{-3}$	$1.41 * 10^{-4}$	$1.40 * 10^{-4}$	$1.41 * 10^{-4}$
$\|\|V^\top V - I\|\|_\mathsf{F}$	$2.81 * 10^{-4}$	$1.30 * 10^{-4}$	$1.30 * 10^{-4}$	$1.30 * 10^{-4}$
$\|\|A - U\Sigma V^\top\|\|_\mathsf{F}$	$4.12 * 10^{-3}$	$2.02 * 10^{-3}$	$2.04 * 10^{-3}$	$2.07 * 10^{-3}$
Computation time[s]	38.645	28.324	27.150	26.822
A_4				
$\|\|U^\top U - I\|\|_\mathsf{F}$	$1.50 * 10^{-3}$	$1.96 * 10^{-4}$	$1.96 * 10^{-4}$	$1.96 * 10^{-4}$
$\|\|V^\top V - I\|\|_\mathsf{F}$	$3.91 * 10^{-4}$	$1.73 * 10^{-4}$	$1.73 * 10^{-4}$	$1.74 * 10^{-4}$
$\|\|A - U\Sigma V^\top\|\|_\mathsf{F}$	$6.77 * 10^{-3}$	$3.16 * 10^{-3}$	$3.18 * 10^{-3}$	$3.08 * 10^{-3}$
Computation time[s]	92.253	104.210	102.440	99.877
A_5				
$\|\|U^\top U - I\|\|_\mathsf{F}$	$2.01 * 10^{-4}$	$4.48 * 10^{-5}$	$4.45 * 10^{-5}$	$4.51 * 10^{-5}$
$\|\|V^\top V - I\|\|_\mathsf{F}$	$9.40 * 10^{-5}$	$4.51 * 10^{-5}$	$4.47 * 10^{-5}$	$4.48 * 10^{-5}$
$\|\|A - U\Sigma V^\top\|\|_\mathsf{F}$	$9.22 * 10^{-4}$	$5.52 * 10^{-4}$	$5.87 * 10^{-4}$	$5.55 * 10^{-4}$
Computation time[s]	0.989	0.789	0.757	0.737
A_6				
$\|\|U^\top U - I\|\|_\mathsf{F}$	$5.29 * 10^{-4}$	$9.12 * 10^{-5}$	$9.11 * 10^{-5}$	$9.18 * 10^{-5}$
$\|\|V^\top V - I\|\|_\mathsf{F}$	$2.18 * 10^{-4}$	$9.13 * 10^{-5}$	$9.15 * 10^{-5}$	$9.16 * 10^{-5}$
$\|\|A - U\Sigma V^\top\|\|_\mathsf{F}$	$2.03 * 10^{-3}$	$1.77 * 10^{-3}$	$1.74 * 10^{-3}$	$1.72 * 10^{-3}$
Computation time[s]	8.402	6.533	6.469	6.224
A_7				
$\|\|U^\top U - I\|\|_\mathsf{F}$	$1.00 * 10^{-3}$	$1.39 * 10^{-4}$	$1.39 * 10^{-4}$	$1.39 * 10^{-4}$
$\|\|V^\top V - I\|\|_\mathsf{F}$	$3.84 * 10^{-4}$	$1.38 * 10^{-4}$	$1.38 * 10^{-4}$	$1.38 * 10^{-4}$
$\|\|A - U\Sigma V^\top\|\|_\mathsf{F}$	$3.46 * 10^{-3}$	$3.52 * 10^{-3}$	$3.93 * 10^{-3}$	$3.51 * 10^{-3}$
Computation time[s]	30.260	29.807	29.257	28.888
A_8				
$\|\|U^\top U - I\|\|_\mathsf{F}$	$1.59 * 10^{-3}$	$1.86 * 10^{-4}$	$1.85 * 10^{-4}$	$1.85 * 10^{-4}$
$\|\|V^\top V - I\|\|_\mathsf{F}$	$5.20 * 10^{-4}$	$1.84 * 10^{-4}$	$1.84 * 10^{-4}$	$1.84 * 10^{-4}$
$\|\|A - U\Sigma V^\top\|\|_\mathsf{F}$	$5.95 * 10^{-3}$	$5.91 * 10^{-3}$	$5.88 * 10^{-3}$	$5.59 * 10^{-3}$
Computation time[s]	78.082	111.501	110.080	108.564

Table 11: Comparison of Jacobi SVD algorithms (with the incorporation of our implementation technique)

	Two-sided Jacobi Proposed Rutishauser	Two-sided Jacobi Proposed Givens rotation
A_1		
$\|\|U^\top U - I\|\|_\mathsf{F}$	$4.32 * 10^{-5}$	$4.34 * 10^{-5}$
$\|\|V^\top V - I\|\|_\mathsf{F}$	$4.30 * 10^{-5}$	$4.33 * 10^{-5}$
$\|\|A - U\Sigma V^\top\|\|_\mathsf{F}$	$3.77 * 10^{-4}$	$3.60 * 10^{-4}$
Computation time[s]	0.683	0.694
A_2		
$\|\|U^\top U - I\|\|_\mathsf{F}$	$8.93 * 10^{-5}$	$8.86 * 10^{-5}$
$\|\|V^\top V - I\|\|_\mathsf{F}$	$8.59 * 10^{-5}$	$8.61 * 10^{-5}$
$\|\|A - U\Sigma V^\top\|\|_\mathsf{F}$	$1.07 * 10^{-3}$	$1.06 * 10^{-3}$
Computation time[s]	6.032	6.265
A_3		
$\|\|U^\top U - I\|\|_\mathsf{F}$	$1.39 * 10^{-4}$	$1.40 * 10^{-4}$
$\|\|V^\top V - I\|\|_\mathsf{F}$	$1.29 * 10^{-4}$	$1.29 * 10^{-4}$
$\|\|A - U\Sigma V^\top\|\|_\mathsf{F}$	$2.01 * 10^{-3}$	$2.11 * 10^{-3}$
Computation time[s]	27.387	28.528
A_4		
$\|\|U^\top U - I\|\|_\mathsf{F}$	$1.95 * 10^{-4}$	$1.95 * 10^{-4}$
$\|\|V^\top V - I\|\|_\mathsf{F}$	$1.71 * 10^{-4}$	$1.72 * 10^{-4}$
$\|\|A - U\Sigma V^\top\|\|_\mathsf{F}$	$3.17 * 10^{-3}$	$3.14 * 10^{-3}$
Computation time[s]	104.007	104.847
A_5		
$\|\|U^\top U - I\|\|_\mathsf{F}$	$4.52 * 10^{-5}$	$4.48 * 10^{-5}$
$\|\|V^\top V - I\|\|_\mathsf{F}$	$4.52 * 10^{-5}$	$4.51 * 10^{-5}$
$\|\|A - U\Sigma V^\top\|\|_\mathsf{F}$	$5.71 * 10^{-4}$	$5.72 * 10^{-4}$
Computation time[s]	0.721	0.747
A_6		
$\|\|U^\top U - I\|\|_\mathsf{F}$	$9.16 * 10^{-5}$	$9.16 * 10^{-5}$
$\|\|V^\top V - I\|\|_\mathsf{F}$	$9.14 * 10^{-5}$	$9.13 * 10^{-5}$
$\|\|A - U\Sigma V^\top\|\|_\mathsf{F}$	$1.77 * 10^{-3}$	$1.77 * 10^{-3}$
Computation time[s]	6.250	6.226
A_7		
$\|\|U^\top U - I\|\|_\mathsf{F}$	$1.39 * 10^{-4}$	$1.39 * 10^{-4}$
$\|\|V^\top V - I\|\|_\mathsf{F}$	$1.38 * 10^{-4}$	$1.38 * 10^{-4}$
$\|\|A - U\Sigma V^\top\|\|_\mathsf{F}$	$3.52 * 10^{-3}$	$3.40 * 10^{-3}$
Computation time[s]	28.822	28.928
A_8		
$\|\|U^\top U - I\|\|_\mathsf{F}$	$1.85 * 10^{-4}$	$1.86 * 10^{-4}$
$\|\|V^\top V - I\|\|_\mathsf{F}$	$1.84 * 10^{-4}$	$1.84 * 10^{-4}$
$\|\|A - U\Sigma V^\top\|\|_\mathsf{F}$	$5.06 * 10^{-3}$	$4.96 * 10^{-3}$
Computation time[s]	108.220	109.734

speed. Furthermore, if we do not incorporate our implementation technique in Section 5.7, 5.8, and 5.9, the performance of the two-sided Jacobi algorithm is not better than that of the one-sided Jacobi algorithm implemented in LAPACK [2] in terms of computation time and accuracy.

The results shown in Table 10 and Table 11, made with the incorporation of our implementation technique in Section 5.7, 5.8, and 5.9 in Table 10 and Table 11, represents that the two-sided Jacobi algorithm (Givens rotation) has a shorter computation time and a higher accuracy than those of the one-sided Jacobi algorithm implemented in LAPACK for both the 500×500, 1000×1000, and 1500×1500 upper triangular matrices whose elements are generated using a uniform random number generator and also, for the 500×500, 1000×1000, and the 1500×1500 upper triangular matrix whose all elements are 1. The accuracy of the implemented two-sided Jacobi algorithm, which is applied using the arctangent function, the implementation procedure by Rutishauser, or Givens rotation, is higher than that of the one-sided Jacobi algorithm, for small matrices. Particularly, the orthogonality in the implemented two-sided Jacobi algorithm is better than that of the one-sided Jacobi algorithm. The smaller is the matrix size, the higher is the accuracy of the implemented two-sided Jacobi algorithm.

In test matrices A_4 and A_8, each of whose dimension size is 2000×2000, the computation time of the implemented two-sided Jacobi algorithm is slightly longer than that of the one-sided Jacobi algorithm. In the other test matrices, the computation time of the implemented two-sided Jacobi algorithm is shorter than that of the one-sided Jacobi algorithm.

The applications such as singular spectrum analysis requires only 100×100 or less dimensions of target matrices so that the proposed two-sided Jacobi algorithm is more suitable than the one-sided Jacobi algorithm. However, better performance for more larger matrices is desired then, we try to improve the one-sided Jacobi algorithm using the knowledge of the implementation of the two-sided Jacobi algorithm.

6 Singular value decomposition using the one-sided Jacobi method

The one-sided Jacobi algorithm is more popular algorithm than the two-sided Jacobi algorithm, and the difference is that the rotation matrices are multiplied only to right side of the target matrix while multiplied to both side in the two-sided Jacobi algorithm. We introduce the outline of the one-sided Jacobi algorithm and improvements of the implementation in this section.

6.1 Outline of the one-sided Jacobi method

Let A and J_i be $m \times n$ real matrix and the following Jacobi rotation of the $n \times n$ matrix, respectively.

$$J_i = \begin{bmatrix} I & 0 & \cdots & \cdots & 0 \\ 0 & \cos\theta_i & \cdots & -\sin\theta_i & \vdots \\ \vdots & \vdots & I & \vdots & \vdots \\ \vdots & \sin\theta_i & \cdots & \cos\theta_i & 0 \\ 0 & \cdots & \cdots & 0 & I \end{bmatrix}$$

From the right side of the matrix A, the rotations of $J_i (i = 1, \cdots)$ is effected so that the converged matrix B satisfies the column orthogonality $B^\top B = D$ (D is the diagonal matrix).

$$AJ_1J_2 \cdots \to B$$

Then, the norm of the column vector $\mathbf{b}_j (j = 1, \cdots, n)$ of the jth column of the matrix B is the singular value $\sigma_j (j = 1, \cdots, n)$ of the matrix A. If $V = J_1J_2 \cdots$, V is a matrix of the right singular vectors. For a column vector $\mathbf{b}_j$ with a positive value σ_j for $j = 1, \cdots, n$, if we divide a column vector $\mathbf{b}_j$ by the positive value σ_j, then each jth column vector of B is the jth left singular vector of the matrix A.

6.2 Conventional implementation for the one-sided Jacobi method

Algorithm 16 shows a pseudo code of the one-sided Jacobi algorithm[32]. It requires an $m \times n$ ($m \geq n$) real matrix A and a convergence-judgment threshold tol and ensures Σ, U, and V. The terms Σ, U, and V denote the matrices whose elements are singular values, left singular vectors, and right singular vectors in A, respectively. Generally, tol is set to $tol = \sqrt{m}\varepsilon$, where ε denotes a machine epsilon, which is adopted in `xGESVJ` routine implemented in LAPACK, considering the error in the inner-product computation.

The algorithm comprises a double loop with the main part. The outer loop repeats until the result of the inner loop converges. The inner loop performs through a subscript $pairs$, which is given using a subroutine `jacobi_pairs`. This generates a pair that contains at least one pair of all the integers from 1 to n. Thus, $pairs$ contains at least $n(n-1)/2$ entries. As an example, we introduce the $pairs$ obtained in the simplest subroutine `jacobi_pairs`,

$$\begin{aligned} pairs =& (1,2), (1,3), \cdots, (1,n), (2,3), (2,4), \cdots, (2,n), \cdots, \\ & (n-2, n-1), (n-2, n), (n-1, n). \end{aligned} \tag{132}$$

As of the two-sided Jacobi algorithm introduced in Section 5, correction of trigonometric values with false position method and secant method, and the FMA instruction improve the accuracy of Givens rotation.

6.3 Proposed Implementation

Six improvements have been made in the proposed implementation.

The first improvement is to avoid fatal bugs that could create an infinite loop. In the implementations using the Givens rotation, rounding errors may occur during orthogonalization while computing the *pair* of two vectors $(\mathbf{a}_j, \mathbf{a}_k)$. Therefore, when there is no effect of improving orthogonality in the *pair* of two vectors $(\mathbf{a}_j, \mathbf{a}_k)$, even upon performing orthogonalization, an infinite loop occurs. To avoid this infinite loop, the orthogonalization computation should be terminated when the orthogonality in all the *pair* becomes invariant.

As the second improvement, the De Rijk method [6] is adopted. In the subroutine `jacobi_pairs` of alg.16, the construction of *pair* is arbitrary and can be freely designed. The De Rijk method suggests an appropriate way to construct *pair*. As the subroutine `jacobi_pairs`, the proposed implementation swaps $\mathbf{a}_j$ with the longest one of $\mathbf{a}_k (k = i, \cdots, n)$. Thus, we can save the computation cost and increase the speed.

As the third improvement, the orthogonalization computation should be minimized. In the algorithm 16, the same computation is performed for all the vectors including those that possess sufficient orthogonality. In this case, it is not only redundant to orthogonalize vectors that previously have sufficient orthogonality but the effect of rounding errors increases by extra orthogonalization. Therefore, vector $\mathbf{a}_j$, which is orthogonal to all the other vectors in the previous iteration, is excluded from the orthogonalization computation. This exclusion can increase accuracy and speed.

The fourth improvement is to avoid overflow and underflow while computing x and y in algorithm 16. In the proposed implementation, lines 6 and 7 in algorithm 16 are properly scaled with the norm of the vectors.

The fifth improvement is also to avoid overflow and underflow. The computation for g may occurs overflow or underflow. To avoid this, $\hat{g}$ and $\mathbf{w}_j = \mathbf{a}_j \times t_j$ is introduced, where t_j is define as follows:

$$
\begin{aligned}
\rho_j &\equiv \begin{cases} \text{SAFMIN} & \sqrt{\hat{\beta}_{j,j}} < \text{SAFMIN} \\ \sqrt{\hat{\beta}_{j,j}} & \text{otherwise} \end{cases} \\
t_j &\equiv \frac{1}{\rho_j} \\
\mathbf{w}_j &\equiv \mathbf{a}_j \times t_j
\end{aligned}
$$

Here, $\sqrt{\hat{\beta}_{j,j}}$, which is described in 6.3.2, is the estimated norm of $\mathbf{a}_j$. Introducing this, the norm $\mathbf{w}_j$ becomes almost equal to 1. SAFMIN means such a safe minimum value that 1/SAFMIN does not overflow. Using the upper $\mathbf{w}_j$, we can compute $\hat{g} = \left(\mathbf{a}_j^\top \mathbf{a}_k\right) \times t_j = \mathbf{w}_j^\top \mathbf{a}_k$, which is adopted in 6.3.1.

The sixth improvement is to compute an accurate norm of a vector under the condition that the approximation of norm is known. That is described in 6.3.3.

6.3.1 Computation with high accuracy in $\cos\theta$ and $\sin\theta$

The Jacobi rotation matrix makes the off-diagonal components zero as follows:

$$J_1 \begin{pmatrix} \mathbf{a}_j^\top \mathbf{a}_j & \mathbf{a}_j^\top \mathbf{a}_k \\ \mathbf{a}_j^\top \mathbf{a}_k & \mathbf{a}_k^\top \mathbf{a}_k \end{pmatrix} J_2 = \begin{pmatrix} \hat{\beta}_{j,j} & 0 \\ 0 & \hat{\beta}_{k,k} \end{pmatrix}, \tag{133}$$

where,

$$J_1 = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}, \tag{134}$$

$$J_2 = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}. \tag{135}$$

The values of $\cos\theta$ and $\sin\theta$ in Equations (134) and (135) are computed in the conventional implementation as follows:

$$x = \mathbf{a}_j^\top \mathbf{a}_j, \tag{136}$$

$$y = \mathbf{a}_k^\top \mathbf{a}_k, \tag{137}$$

$$f = \frac{1}{2}(x - y), \tag{138}$$

$$g = \mathbf{a}_j^\top \mathbf{a}_k, \tag{139}$$

$$t = \frac{g}{\left(f + \text{sign}\left(\sqrt{g^2 + f^2}, f\right)\right)}, \tag{140}$$

$$r = \sqrt{1 + t^2}, \tag{141}$$

$$\cos\theta = \frac{1}{r}, \tag{142}$$

$$\sin\theta = \frac{t}{r}, \tag{143}$$

$$\hat{\beta}_{j,j} = \mathbf{a}_j^\top \mathbf{a}_j + t \times \mathbf{a}_j^\top \mathbf{a}_k, \tag{144}$$

$$\hat{\beta}_{k,k} = \mathbf{a}_k^\top \mathbf{a}_k - t \times \mathbf{a}_j^\top \mathbf{a}_k. \tag{145}$$

In the proposed implementation, instead of f and g, we use $\hat{f}$ and $\hat{g}$ to avoid divergence. The following equations are adopted to compute $\hat{f}$ and $\hat{g}$:

$$\delta_j = s_j \times t_j, \tag{146}$$
$$\delta_k = s_k \times t_j, \tag{147}$$
$$\begin{aligned}\hat{f} =& \frac{1}{2}\left(\sqrt{\mathbf{a}_j^\top \mathbf{a}_j} - \sqrt{\mathbf{a}_k^\top \mathbf{a}_k}\right) \\ & \times \left(\sqrt{\mathbf{a}_j^\top \mathbf{a}_j} \times t_j + \sqrt{\mathbf{a}_k^\top \mathbf{a}_k} \times t_j\right),\end{aligned} \tag{148}$$
$$= \frac{1}{2}\left(s_j - s_k\right)\left(\delta_j + \delta_k\right), \tag{149}$$
$$\hat{g} = \left(\mathbf{a}_j^\top \mathbf{a}_k\right) \times t_j = \mathbf{w}_j^\top \mathbf{a}_k, \tag{150}$$
$$t = \frac{\hat{g}}{\left(\hat{f} + \mathrm{sign}\left(\sqrt{\hat{g}^2 + \hat{f}^2}, \hat{f}\right)\right)}, \tag{151}$$
$$r = \sqrt{\underline{\underline{1 + t^2}}}, \tag{152}$$
$$\cos\theta = \frac{1}{r}, \tag{153}$$
$$\sin\theta = \frac{t}{r}, \tag{154}$$
$$\hat{\beta}_{j,j} = s_j^2 + t \times \hat{g}/t_j, \tag{155}$$
$$\hat{\beta}_{k,k} = s_k^2 - t \times \hat{g}/t_j, \tag{156}$$

where s_j and s_k denote the norm of $\mathbf{a}_j$ and the norm of $\mathbf{a}_k$, respectively. The FMA instruction can be adopted in the double underlined part of these equations. For computing $\sqrt{\hat{g}^2 + \hat{f}^2}$, we employ `xLARTG` implemented in LAPACK.

6.3.2 Approximate computation of the norm of vectors

$\sqrt{\hat{\beta}_{j,j}}$ and $\sqrt{\hat{\beta}_{k,k}}$ represent the approximated norm of $\hat{\mathbf{a}}_j$ and the approximated norm of $\hat{\mathbf{a}}_k$, respectively.

$$\sqrt{\hat{\beta}_{j,j}} \approx ||\hat{\mathbf{a}}_j||_2, \tag{157}$$
$$\sqrt{\hat{\beta}_{k,k}} \approx ||\hat{\mathbf{a}}_k||_2, \tag{158}$$

where

$$\hat{\mathbf{a}}_j = \cos\theta \mathbf{a}_j + \sin\theta \mathbf{a}_k, \tag{159}$$
$$\hat{\mathbf{a}}_k = -\sin\theta \mathbf{a}_j + \cos\theta \mathbf{a}_k. \tag{160}$$

$\sqrt{\hat{\beta}_{j,j}}$ and $\sqrt{\hat{\beta}_{k,k}}$ are computed as follows,

$$\sqrt{\hat{\beta}_{j,j}} = \sqrt{s_j^2 + t \times (\hat{g}/t_j)}, \tag{161}$$

$$= s_j \times \sqrt{\underline{\underline{1 + t \times (\hat{g}/(s_j \times \delta_j))}}}, \tag{162}$$

$$\sqrt{\hat{\beta}_{k,k}} = \sqrt{s_k^2 - t \times (\hat{g}/t_j)}, \tag{163}$$

$$= s_k \times \sqrt{\underline{\underline{1 - t \times (\hat{g}/(s_k \times \delta_k))}}}, \tag{164}$$

where s_j and s_k denote the norm of $\mathbf{a}_j$ and the norm of $\mathbf{a}_k$, respectively. The FMA instruction can be adopted in the double underlined part of these equations. If $\underline{\underline{1 - t \times (\hat{g}/(s_k \times \delta_k))}}$ is not positive, we must use `xNRM2` implemented in LAPACK for computing the norm of the vector $\mathbf{a}_k$.

6.3.3 Accurate computation of the norm of vectors

Let $\mathbf{x}$ be the vector whose norm you compute.

$$\mathbf{x} = (x_1, x_2, \ldots, x_m)\,. \tag{165}$$

Under the condition that the approximation of the norm α is known, the underflow and overflow can be avoided by using the following computation.

$$||\mathbf{x}||_2 = \alpha \times \sqrt{\left(\frac{x_1}{\alpha}\right)^2 + \left(\frac{x_2}{\alpha}\right)^2 + \cdots + \left(\frac{x_m}{\alpha}\right)^2} \tag{166}$$

Concretely, the implementation is as follows.

$$\beta \equiv \begin{cases} \text{SAFMIN} & \alpha < \text{SAFMIN} \\ \alpha & \text{otherwise} \end{cases}, \tag{167}$$

$$\gamma \equiv \frac{1}{\beta}, \tag{168}$$

$$\ell_0 = 0, \tag{169}$$

$$\ell_1 = \underline{\underline{\ell_0 + (\gamma x_1)^2}}, \tag{170}$$

$$\ell_2 = \underline{\underline{\ell_1 + (\gamma x_2)^2}}, \cdots, \tag{171}$$

$$\ell_m = \underline{\underline{\ell_{m-1} + (\gamma x_m)^2}}, \tag{172}$$

$$||\mathbf{x}||_2 = \beta \times \sqrt{\ell_m}. \tag{173}$$

The FMA instruction can be adopted in the double underlined part of these equations.

6.4 Numerical experiments

We evaluate the proposed implementation to see if it had a higher accuracy than those of the QR algorithm, the OQDS algorithm introduced in Section 4, and `xGESVJ` implemented in LAPACK-3.9.0, which is a routine for the one-sided Jacobi algorithm. Because the test matrices are upper triangular, the Householder transformation is adopted as a preprocessing method for the QR and OQDS algorithm.

Table 12 summarizes the experimental environment.

Table 12: Experimental Environment

CPU	Intel(R) Core(TM) i7-9700 CPU 3.00GHz
OS	Ubuntu 20.04.1 LTS
RAM	16GB
Cache	12MB Intel(R) Smart Cache
Compiler	gfortran 9.3.0
Options	-O3 -mtune=native -march=native
Software	Lapack-3.9.0
Precision	single precision

We use four real random upper triangular matrices, whose dimensions are 500×500, 1000×1000, 1500×1500, and 2000×2000, respectively. The following Frobenius norms are used to evaluate the computation errors:

$$||A - U\Sigma V^\top||_\mathsf{F}, \tag{174}$$

$$||U^\top U - I||_\mathsf{F}, \tag{175}$$

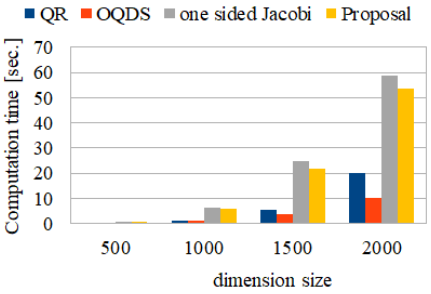
$$||V^\top V - I||_\mathsf{F}. \tag{176}$$

It should be noted that the Jacobi algorithm is not suited for the input matrices. The Jacobi algorithm is originally designed for diagonal dominant matrices therefore, the input matrices which have nonnegative values in their every entries are unfavorable assumptions for the Jacobi algorithm. For this reason, we should compare the relative performance between the conventional one-sided Jacobi algorithm and the proposed one-sided Jacobi algorithm. The results for the computation time of the QR and the OQDS algorithms are just references.

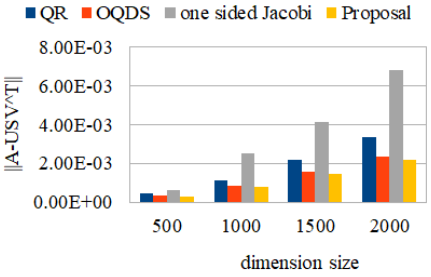
Figure 9 shows the performance results. From figure 9(a), it is shown that the proposed implementation is faster than `xGESVJ`. Despite the computation cost in the Givens rotation with high accuracy is higher than that in the Jacobi rotation in `xGESVJ`, the proposed implementation achieves a higher computation speed with the reduction of redundant computation. From figure 9(b) and (c), it is evident that $||A - U\Sigma V^\top||_\mathsf{F}$ and $||U^\top U - I||_\mathsf{F}$ in the proposed implementation is the highest among all the implementations. It is caused that the QR and OQDS methods are affected by rounding errors due to preprocessing via the Householder transformation; the Givens rotation of the proposed implementation with high accuracy is better than that of `xGESVJ`. Especially, from figure 9(c), we can see that the orthogonality of left singular vectors in the proposed

Table 13: Comparison of SVD algorithms

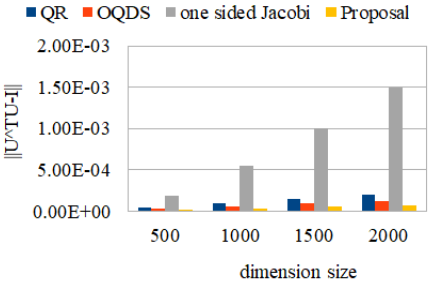
	QR	OQDS	One-sided Jacobi Conventional	One-sided Jacobi Proposed
A_1				
$\|\|U^\top U - I\|\|_\mathsf{F}$	$5.03 * 10^{-5}$	$3.14 * 10^{-5}$	$1.92 * 10^{-4}$	$1.85 * 10^{-5}$
$\|\|V^\top V - I\|\|_\mathsf{F}$	$4.79 * 10^{-5}$	$3.07 * 10^{-5}$	$8.62 * 10^{-5}$	$4.15 * 10^{-5}$
$\|\|A - U\Sigma V^\top\|\|_\mathsf{F}$	$4.27 * 10^{-4}$	$3.40 * 10^{-4}$	$6.85 * 10^{-4}$	$2.77 * 10^{-4}$
Computation time[s]	0.22	0.196	0.947	0.787
A_2				
$\|\|U^\top U - I\|\|_\mathsf{F}$	$9.79 * 10^{-5}$	$6.14 * 10^{-5}$	$5.84 * 10^{-4}$	$3.79 * 10^{-5}$
$\|\|V^\top V - I\|\|_\mathsf{F}$	$9.63 * 10^{-5}$	$6.35 * 10^{-5}$	$1.75 * 10^{-4}$	$8.31 * 10^{-5}$
$\|\|A - U\Sigma V^\top\|\|_\mathsf{F}$	$1.11 * 10^{-3}$	$8.72 * 10^{-4}$	$2.93 * 10^{-3}$	$7.67 * 10^{-4}$
Computation time[s]	1.915	1.708	8.088	7.477
A_3				
$\|\|U^\top U - I\|\|_\mathsf{F}$	$1.54 * 10^{-4}$	$9.25 * 10^{-5}$	$1.03 * 10^{-3}$	$5.72 * 10^{-5}$
$\|\|V^\top V - I\|\|_\mathsf{F}$	$1.42 * 10^{-4}$	$9.42 * 10^{-5}$	$2.82 * 10^{-4}$	$1.25 * 10^{-4}$
$\|\|A - U\Sigma V^\top\|\|_\mathsf{F}$	$2.15 * 10^{-3}$	$1.56 * 10^{-3}$	$4.67 * 10^{-3}$	$1.48 * 10^{-3}$
Computation time[s]	7.292	5.755	30.652	27.8
A_4				
$\|\|U^\top U - I\|\|_\mathsf{F}$	$2.05 * 10^{-4}$	$1.23 * 10^{-4}$	$1.53 * 10^{-3}$	$7.75 * 10^{-5}$
$\|\|V^\top V - I\|\|_\mathsf{F}$	$1.93 * 10^{-4}$	$1.27 * 10^{-4}$	$3.88 * 10^{-4}$	$1.68 * 10^{-4}$
$\|\|A - U\Sigma V^\top\|\|_\mathsf{F}$	$3.39 * 10^{-3}$	$2.38 * 10^{-3}$	$7.02 * 10^{-3}$	$2.15 * 10^{-3}$
Computation time[s]	26.213	14.11	73.345	70.558



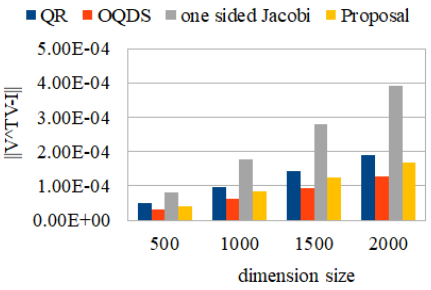
(a) Computation time (s)



(b) $||A - U\Sigma V^\top||_\mathsf{F}$



(c) $||U^\top U - I||_\mathsf{F}$



(d) $||V^\top V - I||_\mathsf{F}$

Figure 9: Comparison

implementation is approximately 10 times more accurate than that in `xGESVJ`. Table 13 shows the comparison in $||V^\top V - I||_\mathsf{F}$. From figure 9(d), we say that $||V^\top V - I||_\mathsf{F}$ in the proposed implementation is smaller than that in the QR method and `xGESVJ`. From table 13, $||V^\top V - I||_\mathsf{F}$ in the proposed implementation is slightly larger than that in the OQDS method but not considerably different.

6.5 Discussion about singular value decomposition using the one-sided Jacobi algorithm

We introduce six improvements to the one-sided Jacobi algorithm in this section. In the implementation, accurate Givens rotation and the FMA instruction were adopted. We also resolved the problem that overflow and underflow may occur in the conventional method[32] with careful implementation. Because the Givens rotation with high accuracy requires considerable computation time, we reduced redundant computation to speed up the operation. As a result of the numerical experiments, the proposed one-sided Jacobi algorithm has a higher performance than the conventional implementation provided in LAPACK in terms of speed and accuracy as shown in Figure 9. Especially, the improvement for the accuracy is notable; in terms of decomposition, the proposed implementation consistently improves the performance left singular vectors and right singular vectors. Also compared with the QR and the OQDS algorithms, the proposed one-sided Jacobi algorithm gives a higher orthogonality of left singular vectors despite of unfavorable assumption. It is expected that the application which requires only left singular vectors, such as Sakurai–Sugiura method[17] prefers the proposed one-sided Jacobi algorithm to the QR or the OQDS algorithms. On the other hand, if the input matrix is small such as the target of singular spectrum analysis, the one-sided Jacobi algorithm might be a best choice among the algorithms because of the high accuracy.

Algorithm 15 Implementation method using the Givens rotation

$f_1 \leftarrow R_{j,j} - R_{k,k}$
$f_1 \leftarrow f_1 + \mathrm{SIGN}\left(\underline{\sqrt{f_1^2 + R_{j,k}^2}}, f_1\right)$ The Givens rotation is adopted in the underlined part
if $f_1 \geq 0$ **then**
  $g_1 \leftarrow R_{j,k}$
else
  $g_1 \leftarrow -R_{j,k}$
  $f_1 \leftarrow -f_1$
end if
$f_2 \leftarrow R_{j,j} + R_{k,k}$
$f_2 \leftarrow f_2 + \mathrm{SIGN}\left(\underline{\sqrt{f_2^2 + R_{j,k}^2}}, f_2\right)$ The Givens rotation is adopted in the underlined part
if $f_2 \geq 0$ **then**
  $g_2 \leftarrow -R_{j,k}$
else
  $g_2 \leftarrow R_{j,k}$
  $f_2 \leftarrow -f_2$
end if
if $f_1 \geq f_2$ **then**
  $t_1 \leftarrow g_1 / f_1$
  $\hat{c}_1 \leftarrow \underline{\underline{-t_1 \times g_2 + f_2}}$
  $\hat{s}_1 \leftarrow \underline{\underline{t_1 \times f_2 + g_2}}$
  $\hat{c}_2 \leftarrow \underline{\underline{t_1 \times g_2 + f_2}}$
  $\hat{s}_2 \leftarrow \underline{\underline{t_1 \times f_2 - g_2}}$
else
  $t_2 \leftarrow g_2 / f_2$
  $\hat{c}_1 \leftarrow \underline{\underline{-g_1 \times t_2 + f_1}}$
  $\hat{s}_1 \leftarrow \underline{\underline{f_1 \times t_2 + g_1}}$
  $\hat{c}_2 \leftarrow \underline{\underline{g_1 \times t_2 + f_1}}$
  $\hat{s}_2 \leftarrow \underline{\underline{-f_1 \times t_2 + g_1}}$
end if
Compute c_1 and s_1 using the Givens rotation for $x \leftarrow \hat{c}_1$ and $y \leftarrow \hat{s}_1$
Compute c_2 and s_2 using the Givens rotation for $x \leftarrow \hat{c}_2$ and $y \leftarrow \hat{s}_2$

Algorithm 16 Pseudo code of the one-sided Jacobi method

Require: $A = [\mathbf{a}_1 \quad \mathbf{a}_2 \quad \cdots \quad \mathbf{a}_n], tol$
Ensure: (U, Σ, V)
1: $V := [\mathbf{v}_1 \quad \mathbf{v}_2 \quad \cdots \quad \mathbf{v}_n] := I_{n,n}$
2: **repeat**
3: $maxt := 0$
4: $pairs :=$ `jacobi_pairs`()
5: **for** (j, k) in $pairs$ **do**
6: $x := \mathbf{a}_j^\top \mathbf{a}_j$
7: $y := \mathbf{a}_k^\top \mathbf{a}_k$
8: $g := \mathbf{a}_j^\top \mathbf{a}_k$
9: $\tau := |g|/\sqrt{x \times y}$
10: $maxt := \max(maxt, \tau)$
11: **if** $\tau > tol$ **then**
12: Computation of Jacobi rotation
13: $f := (x - y)/2$
14: $t := g/\left(f + \text{sign}\left(\sqrt{g^2 + f^2}, f\right)\right)$
15: $r := \sqrt{1 + t^2}$
16: $\cos\theta := 1/r$
17: $\sin\theta := t/r$
18: Effect of Jacobi rotation
19: $\mathbf{q} := \mathbf{a}_j$
20: $\mathbf{a}_j := \cos\theta\mathbf{q} + \sin\theta\mathbf{a}_k$
21: $\mathbf{a}_k := -\sin\theta\mathbf{q} + \cos\theta\mathbf{a}_k$
22: Effect of Jacobi rotation
23: $\mathbf{q} := \mathbf{v}_j$
24: $\mathbf{v}_j := \cos\theta\mathbf{q} + \sin\theta\mathbf{v}_k$
25: $\mathbf{v}_k := -\sin\theta\mathbf{q} + \cos\theta\mathbf{v}_k$
26: **end if**
27: **end for**
28: **until** $maxt > tol$
29: **for** $j = 1$ to n **do**
30: $\sigma_j := ||\mathbf{a}_j||_2$
31: **end for**
32: $\Sigma := \text{diag}(\sigma_1, \sigma_2, \cdots, \sigma_n)$
33: $U := A\Sigma^{-1}$

7 Conclusion

In this thesis, we discussed about the implementation details and applications of the OQDS algorithm and the Jacobi algorithm.

The narrow band reduction approach was introduced as the first application of the OQDS algorithm in Section 3 to utilize the expandability of the OQDS algorithm for the lower triangular matrices. This approach intends to reduce the total computation time of singular value computation with the block Householder reduction that reduces an input matrix to a lower triangular matrix faster than the conventional Householder bidiagonalization. We formulated the OQDS algorithm for the lower tridiagonal matrix obtained as the result of the BLAS level 2.5 Householder reduction algorithm which approximately cuts in half of data transfer between the main memory and cache by computing two vectors-matrix multiplication simultaneously. A new shift strategy consists of the Gerschgoriin shift, the algebraic shit, the Laguerre shift and the Kato-Temple shift accelerates the convergence of the OQDS algorithm. In addition, we designed the convergence criteria to terminate the OQDS algorithm. If the criteria are satisfied, we can treat the elements as zero without any effect to the obtained singular values and perform deflation and splitting operations for the target matrix. By introducing these improvements, the OQDS algorithm got a practical performance for computing singular values of lower tridiagonal matrices. The numerical experiment shows that the OQDS algorithm for lower tridiagonal matrices performs a faster computation of singular values of real symmetric matrices than the DQDS algorithm does for bidiagonal matrices in terms of total computation time of preliminary reduction and diagonalization as in Section 3.6.

On the other hand, the narrow band reduction approach could not utilize the accuracy of the OQDS algorithm because of the increase of the number of iterations caused by the expansion for lower tridiagonal matrices. However, since the high accuracy of orthogonal transformation based formulation of the OQDS algorithm is a great strength so we proposed an application of the OQDS algorithm for bidiagonal matrix utilizing the accuracy of singular values and singular vectors. The column space computation method using the combination of the OQDS algorithm and the DQDS algorithm was introduced in Section 4. The main idea of the method is that we compute singular values by the DQDS algorithm which is extremely fast in exchange for the capability of computing singular vectors and then, compute the column vectors of the target matrix by the OQDS algorithm corresponding to the singular values given by the DQDS algorithm. For the DQDS algorithm, we explained the formulation and introduced shift strategy suitable for bidiagonal matrix. The shift strategy consists of the generalized Rutishauser bound, the Laguerre method and the Newton method based bound, the Collatz inequality based bound and the Johnson bound as described in Section 4.1.1. As the convergence criteria of the DQDS algorithm, we adopt the criteria which adds one convergence condition based on a recurrence relation of the relative values of diagonal and off-diagonal elements to the conventional convergence criteria. For the OQDS algorithm, we formulated the sequence of operations obtaining singular vectors for bidiagonal matrices and pro-

pose the shift strategy and convergence criteria. We added the generalized Rutishauser bound, the Collatz inequality based bound and the Johnson bound which are suitable for estimating the minimum singular value of bidiagonal matrix to the shift strategy designed for the OQDS algorithm in Section 3. The detail of the shift strategy is described in Section 4.2.2. The convergence criteria for the OQDS algorithm for bidiagonal matrix is based on Weyl's monotonicity theorem as same as that of the OQDS algorithm for lower tridiagonal matrices. Section 4.5 describes the numerical experiments of the column space computation method. Our numerical experiments show that the proposed method performs faster and more accurate computation than that of the conventional implementation of the OQDS algorithm[5]. The column space computation method can be used as the subroutine of the Sakurai-Sugiura method[17] which computes eigenvalues of polynomial eigenvalue problem by a parallel computation.

In the latter part of this thesis, we discussed about the Jacobi algorithm as another singular value decomposition algorithm based on orthogonal transformation. The two-sided Jacobi algorithm, an explicit form of the Jacobi algorithm, is explained in Section 5. We carefully implemented the Givens transformation which is the main component of the two-sided Jacobi algorithm. Five patterns of the implementations were proposed as the candidate, conventional arctangent calculation, faster and accurate variants of the arctangent calculation, Rutishauser's calculation method and the Givens rotation. And then, we correct the result of each implementation with the false position method and the secant method in order to achieve a higher accuracy. In addition, in the perspective of the expansion of OQDS algorithm, a singular value sorting feature similar to that of the DQDS and OQDS algorithm could be introduced to the two-sided Jacobi algorithm with an appropriate selection of the angle of Givens rotation. In the implementation of the two-sided Jacobi algorithm, the FMA instruction that performs a fast and accurate computation that combines multiply and add operation can be applied to the formulation. The FMA instruction significantly contributes the speed and accuracy of the two-sided algorithm therefore, the Givens rotation based implementation which contains the most number of the FMA instruction achieved the best performance among the five implementations. As a result of these improvements, the proposed implementation of the two-sided Jacobi algorithm achieved a higher and accurate performance than that of the conventional implementation as the numerical experiments in Section 5.10 show.

In Section 6, we presented the improvements to the one-sided Jacobi algorithm in order to develop a practical solver for singular value decomposition. We adopted the De Rijk method to construct the Jacobi pairs and carefully implemented the components of the algorithm to avoid numerical error. The FMA instruction is also introduced the formulation. As a result, the one-sided Jacobi algorithm improved its speed and accuracy compared with the conventional implementation. In general, the computational speed of the one-sided Jacobi algorithm is inferior to that of the QR algorithm for general dense matrix and our proposed one-sided Jacobi algorithm is still slower than the QR algorithm. On the other hand, if the number of non-zero elements of the input matrix is very small such as the case of a sparse matrix, or the non-zero elements are gathered near diagonal entries at the initial state of the input matrix, the one-sided Jacobi algorithm

is expected to performs faster computation than the QR algorithm. The accuracy of the one-sided Jacobi algorithm is notably high on the situation which we can tolerate the slowness. Especially, as the result of the numerical experiments presented in Section 6.4, the accuracy of original matrix restoration and the orthogonality of the column vectors are the highest among the tested four algorithms despite of the disadvantageous of input matrix. For the application of singular value decomposition which requires a high accuracy in a small system such as singular spectrum analysis, the proposed one-sided Jacobi algorithm might be the most suitable solver.

There are two widely used algorithm for computing singular value decomposition, the QR algorithm and the divide and conquer algorithm using the Newton method as the LAPACK subroutines DBDSQR and DBDSDC, respectively. Among the two algorithms, it is reported that the accuracy of the divide and conquer algorithm gets worse for small singular values of ill-conditioned matrices[40]. Since, in this thesis, we compared the performance of two orthogonal transformation based algorithms, the OQDS algorithm and the Jacobi algorithm, with those of the QR algorithm and the DQDS algorithm in the context of a robust singular value decomposition algorithm. We improved the performance of the OQDS algorithm and the Jacobi algorithm compared with each conventional implementation through the appropriate design. For the OQDS algorithm, we carefully treat the numerical error in the formulation and accelerate the convergence with well-designed shift strategy. For the Jacobi algorithm, the sorting feature of singular values increases the convenience and accelerates the convergence of the algorithm, and the formulation which utilizes the FMA instruction derives a strong advantage in the accuracy of the algorithm. Then, for some applications, our proposed algorithms are prior to the QR algorithm and the DQDS algorithm. Through the fast and accurate implementation of the OQDS algorithm and the Jacobi algorithm, it is shown that how the orthogonal transformation contributes the accuracy of the algorithms based on the transformation. In addition, we discovered an intrinsic relationship between the two orthogonal transformation based algorithms which have been considered as individually. As a future work, We would like to find more common features among the algorithms and make use of them to improve the performance of the algorithm each other.

Acknowledgments

The authors would like to express a sincere gratitude to Professor Yoshimasa Nakamura for many instructions and encouragements. His sincere gratitude also goes to Professor Kinji Kimura at University of Fukui for the continuous support of Ph.D study and research, for his patience, motivation, enthusiasm and immense knowledge. He would also like to thank Professor Masami Takata at Nara Women's University and all Nakamura laboratory members for valuable discussions and comments.

References

[1] Golub, G.H. and van Loan, C.F., *Matrix Computations*, Third edition, Johns Hopkins University Press, 1996.

[2] Linear Algebra PACKage, `http://www.netlib.org/lapack/`, (2019, 12, 19).

[3] Dongarra, J. J., Hammarling, S. J. and Sorenson, D. C., *Block reduction of matrices to condensed forms for eigenvalue computations*, Journal of Computational and Applied Mathematics, Vol. 27, pp. 215–227 (1989).

[4] Dongarra, J. J. and van de Geijn, R. A., *Reduction to condensed form for the eigenvalue problem on distributed architectures*, Parallel Computing, Vol.18, No.9, pp. 973–982 (1992).

[5] von Matt, U., *The orthogonal QD algorithm*, SIAM J. Sci. Comput., vol.18, Issue:4, pp.1163–1186 (1997).

[6] De Rijk, *A one-sided Jacobi algorithm for computing the singular value decomposition on a vector computer*, SIAM J. Sci. Stat. Comp., 10, pp.359–371 (1998).

[7] Rutishauser, H., *Der Quotienten-Differenzen-Algorithms*, ZAMP, vol. 5, pp. 233–251 (1954).

[8] Rutishauser, H., *Lectures on Numerical Mathematics*, Birkhäuser, Boston, 1990.

[9] Parlett, B. N., *The new qd algorithms*, Acta Numerica, vol. 4, pp. 459–491 (1995).

[10] Fernando, K. V. and Parlett, B. N., *Accurate singular values and differential qd algorithms*, Numerische Mathematik, vol. 67, pp. 191–229 (1994).

[11] Howell, G. W., Demmel, J. W., Fulton, C. T., Hammerling, S. and Marmol, K., *Cache Efficient Bidiagonalization Using BLAS 2.5 Operators*, LAWNs, lawn174, 2006.

[12] Chatelin, F., *Valeurs propres de matrices*, Masson, Paris, 1988.

[13] Gerschgorin, S., *Uber die Abgrenzung der Eigenwerte einer Matrix*, Izv. Akad. Nauk., USSR Otd. Fiz.-Mat. Nauk 7, pp.749–754, (1931).

[14] Parlett, B. N., *Laguerre's method applied to the matrix eigenvalue problem*, Math. Comp., 18, pp. 464–485, (1964).

[15] Parlett, B. N., *The Symmetric Eigenvalue Problem*, Prentice-Hall, Englewood Cliffs, 1980.

[16] Wilkinson, J. H., *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford, 1995.

[17] Sakurai, T., and Tadano, H.: *CIRR: A Rayleigh-Ritz type method with counter integral for generalized eigenvalue problems*, Hokkaido Math. J., Vol. 36, pp. 745–757 (2007).

[18] Aishima, K., Matsuo, T., and Murota, K.: *Rigorous proof of cubic convergence for the dqds algorithm for singular values*, Japan J. Indust. Appl. Math. , Vol.25, pp.63–81 (2008).

[19] Johnson, C. R.: *A Gersgorin-type lower bound for the smallest singular value*, Lin. Alg. Appl., Vol. 112, pp. 1–7 (1989).

[20] Collatz, L.: *Einschliessungssatz fuer die charakteristischen Zahlen von Matrizen*, Mathemathische Zeitschrift, Vol.48, pp.221-226 (1942).

[21] IEEE Standard for Floating-Point Arithmetic, *IEEE Std. 754-2008*, 2000.

[22] Conte, S. D., *Elementary Numerical Analysis / an algorithmic approach*, McGraw-Hill, 1965.

[23] Strom, C. and Avriel, M., *Nonlinear Programming: Analysis and Methods*, Prentice Hall, 1976.

[24] Kudo, S., Yasuda, K. and Yamamoto, Y., *Performance of the parallel block Jacobi method with dynamic ordering for the symmetric eigenvalue problem*, JSIAM Letters, 10(2015), pp. 41–44 (2015).

[25] Hari, V., and Zadelj-Martić V., *Parallelizing the Kogbetliantz Method: A First Attempt. Journal of Numerical Analysis*, Industrial and Applied Mathematics (JNAIAM), Vol. 2 No 1–2, pp.49–66 (2007).

[26] Imamura, T., Yamada, S. and Machida, M., *Narrow-band reduction approach of a DRSM eigensolver on a multicore-based cluster system*, Advances in Parallel Computing, vol.19, Parallel Computing: From Multicores and GPU's to Petascale, IOS Press, 2010, pp.91–98.

[27] Rutishauser, H., *Uber eine kubisch konvergente Variante der LR-Thransformation*, Zeitschrift fur Angewandte Mathematik und Mechanik, vol. 40, pp. 49–54 (1960).

[28] Wilkinson, J.H. and Reinsch, C. (eds.), *Linear Algebra*, Springer-Verlag, 1971.

[29] Bischof, C. H., Lang, B., and Sun, X., *The SBR toolbox - software for successive band reduction*, ACM TOMS 26(4), pp. 602–616, (2000).

[30] Wu, Y. J., Alpatov, P. A., Bischof, C. H., and Geijn, R. A., *A parallel implementation of symmetric band reduction using PLAPACK*, In Proc. Scalable Parallel Library Conference (1996).

[31] Bischof, C.H. and van Loan, C. F., *The WY representation for products of Householder matrices*, SIAM J. Sci. and Stat. Comput., Vol.8, No. 1, pp. s2-s13 (1987).

[32] Demmel, J., and Veselic, K.: *Jacobi's method is more accurate than QR*, SIAM J. Matrix Anal. Appl., Vol. 13, No. 4, pp.1204–1245 (1992).

[33] Hida, Y., Li, X. S., and Bailey, D. H.: *Library for Double-Double and Quad-Double Arithmetic*, Proc. 15th Symposium on Computer Algorithmetic, pp.155–162 (2007).

[34] Parlett, B. N., and Marques, O. A.: *An Implementation of the dqds Algorithm (Positive Case)*, Lin. Alg. Appl, Vol. 309, No. 1-3, pp. 217–259 (2000).

[35] Yamashita, T., Kimura, K., and Yamamoto, Y.: *A new subtraction-free formula for lower bounds of the minimal singular value of an upper bidiagonal matrix*, Numerical Algorithms, Vol. 69, Issue 4, pp. 893–912 (2015).

[36] Demmel, J.: *Applied Numerical Linear Algebra*, SIAM, Philadelphia (1997).

[37] Yamamoto, Y.: *On the optimality and sharpness of Laguerre's lower bound on the smallest eigenvalue of a symmetric positive definite matrix*, Applications of Mathematics, Vol. 62, Issue 4, pp. 319–331 (2017).

[38] Basic Linear Algebra Subprograms, Netlib. (online), `http://netlib.org/blas/index.html` (2018.05.29)

[39] Rutishauser, H.: *The Jacobi method for real symmetric matrices*, Numerische Mathematik, Vol. 9, No. 1, pp.1–10 (1966).

[40] Iwasaki, M. and Nakamura, Y., *Positivity of DLV and mDLVs Algorithms for Computing Singular Values*, Electronic Transactions on Numerical Analysis.Volume 38, pp. 184–201 (2011).

List of the author's papers related to this thesis

1. Sho Araki, Hiroki Tanaka, Kinji Kimura and Yoshimasa Nakamura, *Implementation of the orthogonal qd algorithm for lower tridiagonal matrices*, in: Proceedings of The 2013 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA2013), 2013, pp. 161–167. {Section 3}

2. Sho Araki, Kinji Kimura, Yusaku Yamamoto and Yoshimasa Nakamura, *Implementation details of an extended oqds algorithm for singular values*, JSIAM Letters, 7(2015), pp. 9–12. {Section 3}

3. Sho Araki, Hiroki Tanaka, Masami Takata, Kinji Kimura and Yoshimasa Nakamura, *Fast computation method of column space by using the DQDS method and the OQDS method*, in: Proceedings of 2018 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA2018), 2018, pp. 333–339. {Section 4}

4. Sho Araki, Masana Aoki, Masami Takata, Kinji Kimura and Yoshimasa Nakamura, *On an implementation of the two-sided Jacobi method*, IPSJ Trans. Math. Modeling Appl. Vol.14, No.1, pp. 12—20 (Jan. 2021). {Section 5}

5. Masami Takata, Sho Araki, Takahiro Miyamae, Kinji Kimura, Yoshimasa Nakamura, *On an Implementation of the One-Sided Jacobi Method with High Accuracy*, IPSJ Trans. Math. Modeling Appl. to appear. {Section 6}