Numerical Algorithms (2024) 95:979–1009
https://doi.org/10.1007/s11075-023-01596-9

ORIGINAL PAPER

Acceleration of iterative refinement for singular value decomposition

Yuki Uchino[1] · Takeshi Terao[2,3] · Katsuhisa Ozaki[4]

Received: 5 August 2022 / Accepted: 1 June 2023 / Published online: 19 July 2023

Abstract
We propose fast numerical algorithms to improve the accuracy of singular vectors for a real matrix. Recently, Ogita and Aishima proposed an iterative refinement algorithm for singular value decomposition that is constructed with highly accurate matrix multiplications carried out six times per iteration. The algorithm runs for the problem that has no multiple and clustered singular values. In this study, we show that the same algorithm can be run with highly accurate matrix multiplications carried out five times. Also, we proposed four algorithms constructed with highly accurate matrix multiplications, two algorithms with the multiplications carried out four times, and the other two with the multiplications carried out five times. These algorithms adopt the idea of a mixed-precision iterative refinement method for linear systems. Numerical experiments demonstrate speed-up and quadratic convergence of the proposed algorithms. As a result, the fastest algorithm is 1.7 and 1.4 times faster than the Ogita-Aishima algorithm per iteration on a CPU and GPU, respectively.

Keywords Singular value decomposition · Iterative refinement · Mixed-precision computation · Accurate numerical computation

Takeshi Terao and Katsuhisa Ozaki contributed equally to this work

✉ Yuki Uchino
nb22105@shibaura-it.ac.jp

Takeshi Terao
terao.takeshi.412@m.kyushu-u.ac.p

Katsuhisa Ozaki
ozaki@sic.shibaura-it.ac.jp

1 Graduate School of Engineering and Science, Shibaura Institute of Technology, 307 Fukasaku, Minuma-ku, Saitama-city 337-8570, Saitama, Japan

2 RIKEN Center for Computational Science, 7-1-26 Minatojima-minami-machi, Chuo-ku, Kobe 650-0047, Hyogo, Japan

3 Research Institute for Information Technology, Kyushu University, 744 Motooka, Nishi-ku 819-0395, Fukuoka, Japan

4 Department of Mathematical Sciences, Shibaura Institute of Technology, 307 Fukasaku, Minuma-ku, Saitama-city 337-8570, Saitama, Japan

Mathematics Subject Classification (2010) 65F15 · 65F30 · 65F10

1 Introduction

This study investigates the singular value decomposition for $A \in \mathbb{R}^{m\times n}$:

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

where $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ are orthogonal matrices whose ith columns are the left singular vectors $u_{(i)} \in \mathbb{R}^m$ for $i = 1, \dots, m$ and the right singular vectors $v_{(i)} \in \mathbb{R}^n$ for $i = 1, \dots, n$, respectively and $\Sigma \in \mathbb{R}^{m\times n}$ is a rectangular diagonal matrix whose ith diagonal elements are the singular values $\sigma_i \in \mathbb{R}$ for $i = 1, \dots, n$. Throughout the paper, we assume that $m \ge n$. The approximation $\hat{u}_{(i)} \approx u_{(i)}, \hat{\sigma}_i \approx \sigma_i$, and $\hat{v}_{(i)} \approx v_{(i)}$ for all i can be obtained by using various numerical solvers for singular value decomposition, e.g., `gesdd`, `gesvd`, `gesvdx` in LAPACK [1] and `svd`, `svds`, or `svdsketch` in MATLAB.

Now, for $k \le n$, let

$$\begin{aligned} U_{1:k} &:= (u_{(1)}, \dots, u_{(k)}), & \hat{U}'_{1:k} &:= (\hat{u}'_{(1)}, \dots, \hat{u}'_{(k)}) & &\in \mathbb{R}^{m\times k}, \\ V_{1:k} &:= (v_{(1)}, \dots, v_{(k)}), & \hat{V}'_{1:k} &:= (\hat{v}'_{(1)}, \dots, \hat{v}'_{(k)}) & &\in \mathbb{R}^{n\times k} \end{aligned}$$

with $\hat{u}'_{(i)} := \hat{u}_{(i)}/\|\hat{u}_{(i)}\|_2$ and $\hat{v}'_{(i)} := \hat{v}_{(i)}/\|\hat{v}_{(i)}\|_2$, and $\hat{\Sigma}'_k := \operatorname{diag}(\hat{\sigma}'_1, \dots, \hat{\sigma}'_k) \in \mathbb{R}^{k\times k}$ with $\hat{\sigma}'_i := (\hat{u}'_{(i)})^T A \hat{v}'_{(i)}$. The residuals are defined as

$$R_{1:k} := A\hat{V}'_{1:k} - \hat{U}'_{1:k}\hat{\Sigma}'_k, \quad S_{1:k} := A^T\hat{U}'_{1:k} - \hat{V}'_{1:k}\hat{\Sigma}'_k,$$

and the gap of the singular values is given by

$$\delta_k := \min_{\substack{1\le i\le k \\ k<j\le \max(n,k+1)}} |\hat{\sigma}'_i - \sigma_j|, \tag{2}$$

where $\sigma_{n+1} := 0$ for the sake of convenience. Wedin [2] proposed the $\sin\Theta$ theorem for singular value decomposition extending the $\sin\Theta$ theorems for Hermitian matrices proposed by Davis and Kahan in [3]. If $\delta_k > 0$, then from [2], it holds that

$$\sqrt{\|\sin\Theta(U_{1:k}, \hat{U}'_{1:k})\|_F^2 + \|\sin\Theta(V_{1:k}, \hat{V}'_{1:k})\|_F^2} \le \frac{\sqrt{\|R_{1:k}\|_F^2 + \|S_{1:k}\|_F^2}}{\delta_k}, \tag{3}$$

where $\Theta(U_{1:k}, \hat{U}'_{1:k})$ and $\Theta(V_{1:k}, \hat{V}'_{1:k})$ are matrices of canonical angles (see [4]) between $U_{1:k}$ and $\hat{U}'_{1:k}$ and between $V_{1:k}$ and $\hat{V}'_{1:k}$, respectively, and $\|\cdot\|_F$ denotes the Frobenius norm. Note that Dopico [5] also proposed a similar theorem. From (3), the smaller the value of δ_k in (2), the lower the accuracy of the computed singular vectors. Thus, iterative refinement methods are useful for obtaining sufficiently accurate results in singular value decomposition.

Let $\hat{U} := (\hat{u}_{(1)}, \dots, \hat{u}_{(m)})$ and $\hat{V} := (\hat{v}_{(1)}, \dots, \hat{v}_{(n)})$. Davies and Smith proposed an iterative refinement algorithm for singular value decomposition of $\hat{U}^T A \hat{V}$ in [6]. However, the singular values for $\hat{U}^T A \hat{V}$ are slightly perturbed compared to the exact singular values for the original matrix A. Convergence to results that include perturbations indicates a limitation of the achievable accuracy using the Davies-Smith algorithm. Recently, Ogita and Aishima proposed an iterative refinement algorithm for singular value decomposition of A constructed with highly accurate matrix multiplications carried out six times per iteration in [7]. That algorithm converges to exact singular vectors of A when $\hat{U}$ and $\hat{V}$ are moderately accurate.

The main contributions of this study are

- speeding up the Ogita-Aishima algorithm and
- performance evaluation of the proposed methods on a CPU and GPU when improving the accuracy of $\hat{U}$ and $\hat{V}$ from double-precision to quadruple-precision or from single-precision to double-precision.

We use the following environments for numerical experiments in this paper:

Env. 1) MATLAB R2021a on a personal computer with an Intel Core i9-10900X CPU (3.70 GHz) with 128 GB main memory, a GeForce RTX 3090 GPU with CUDA 11.6, and Windows 10 operating system

Env. 2) MATLAB R2021b on a personal computer with four AMD EPYC 7542 CPUs (2.90 GHz) with 512 GB shared main memory, a A100 Tensor Core GPU with CUDA 11.3, and Ubuntu 20.04.2 LTS operating system

Generally, double-precision arithmetic is faster than arbitrary precision arithmetic implemented in software or arithmetic for multiple-component format values, such as double-double arithmetic [8, 9]. Moreover, on an enthusiast-class GPU, such as the GeForce RTX 3090, single-precision arithmetic is much faster than double-precision arithmetic. For example, using Env. 1, we obtain the results shown in Table 1. The table indicates double-precision matrix multiplication is 1353 times faster than quadruple-precision on CPU and single-precision matrix multiplication is 34 times faster than double-precision on GPU. Note that for quadruple-precision arithmetic in Table 1, we use the Advanpix Multiprecision Computing Toolbox for MATLAB [10]. Hence, lower-precision computations are much faster than higher-precision computations. Thus, we consider algorithms that reduce the cost of higher-precision computations at the expense of increasing the cost of lower-precision computations.

Table 1 Computing time in seconds for matrix multiplication of $A \in \mathbb{R}^{n \times n}$ and $B \in \mathbb{R}^{n \times n}$ for $n = 10000$ using Env. 1

CPU			GPU		
double	quadruple	ratio	single	double	ratio
2.45	3316	1353	0.11	3.73	34

The computing time ratio, (single-precision arithmetic):(double-precision arithmetic), for singular value decomposition is about $1 : 2$ to $2 : 3$ on a CPU and GPU in Env. 1 and 2, respectively. Moreover, matrix multiplication can be performed much faster than singular value decomposition. For example, from Table 2, using the single-precision results as the initial guess, the Ogita-Aishima algorithm with double-precision matrix multiplication can be run for $(22.7 - 15.4)/(0.11 \cdot 6) \approx 11$ iterations within the double-precision computing time. For a linear system, LAPACK provides a routine `dsgesv` that computes an initial guess for the solution using single-precision arithmetic and obtains double-precision equivalent results by using the mixed-precision iterative refinement method. Our study will enable the development of a similar routine for singular value decomposition.

In this study, we first show that the Ogita-Aishima algorithm can be executed with highly accurate matrix multiplications carried out four times per iteration, that is, with one fewer multiplication than in their original paper. Next, we propose four iterative refinement algorithms for singular value decomposition, combining the idea of a mixed-precision iterative refinement method for a linear system with the Ogita-Aishima algorithm. Those algorithms are constructed with highly accurate matrix multiplications carried out either four or five times per iteration. The Ogita-Aishima algorithm and proposed algorithms (Algorithms 5, 6, and 8 in the paper) are the same in exact arithmetic, their limiting accuracy in finite precision is different, and will be explored experimentally in this work. Numerical experiments are conducted to demonstrate speed-up and quadratic convergence of the proposed algorithms on a CPU and GPU.

The remainder of this paper is organized as follows: Section 2 introduces results obtained in previous studies; Section 3 presents the proposed iterative refinement algorithms for singular value decomposition; Section 4 presents numerical examples to illustrate the efficiency of the proposed algorithms; and Section 5 provides final remarks.

2 Notation and previous work

2.1 Notation

Let $(\cdot)_h$ and $(\cdot)_\ell$ denote the results of numerical arithmetic, where all operations inside the parentheses are executed at higher- and lower-precision, respectively. The combinations of precision [higher precision and lower precision] using in this study are [quadruple-precision and double-precision] and [double-precision and single-precision]. For simplicity, we will omit terms less than $\mathcal{O}(m^k n^\ell)$, $k + \ell = 3$ from the operations count. For example, we will write $s_1 m^3 + s_2 m^2 n + s_3 mn^2 + s_4 n^3 + s_5 m^2 + s_6 mn + s_7 n^2 + \cdots$ operations for $s_i \in \mathbb{N}$ as $s_1 m^3 + s_2 m^2 n + s_3 mn^2 + s_4 n^3$ operations.

Table 2 Computing time in seconds for matrix multiplication of A and B and the singular value decomposition of A for $A \in \mathbb{R}^{n\times n}$ and $B \in \mathbb{R}^{n\times n}$ for $n = 10000$ using GPU using Env. 2

`svd`			matrix multiplication		
single	double	ratio	single	double	ratio
15.44	22.74	1.5	0.11	0.11	1

2.2 Mixed-precision iterative refinement technique

We consider a linear system $Ax = b$ for $x, b \in \mathbb{R}^n$ and a non-singular matrix $A \in \mathbb{R}^{n\times n}$. Let $\hat{x} \approx x$ be a computed solution to $Ax = b$. There is an iterative refinement algorithm with mixed-precision arithmetic to improve the accuracy of $\hat{x}$. The process is shown in Algorithm 1.

Algorithm 1 Iterative refinement for the approximate solution $\hat{x}$ of a linear system $Ax = b$. The initial $\hat{x}$ is obtained using lower-precision arithmetic.

```
1: repeat
2:     Compute $r \leftarrow (b - A\hat{x})_h$.
3:     Convert $r$ into lower-precision.
4:     Compute $y \leftarrow (A^{-1}r)_\ell$.
5:     Update $\hat{x}$ as $\hat{x} \leftarrow (\hat{x} + y)_{h \text{ or } \ell}$.
6: until accuracy of $\hat{x}$ is sufficient.
```

Note that if the matrix decomposition of A is already obtained for the initial approximation $\hat{x}$, the cost of line 4 is almost negligible. For using a solver based on LU factorization, Wilkinson [11] gave the error analysis for fixed-point arithmetic, and Moler [12] extended that for floating-point arithmetic. Jankowski and Woźniakowski [13] showed that the algorithm with an arbitrary solver could be made stable in the usual sense by normwise error analysis. Skeel [14] also showed that with a solver based on LU factorization by elementwise error analysis. Higham [15–17] extended Skeel's analysis to an arbitrary solver. Langou et al. [18] showed the maximum number of iterations to convergence on the method with a solver based on LU factorization using single- and double-precision arithmetic. Carson and Higham [19] gave an error analysis for ill-conditioned A. In addition, Algorithm 1 has been accelerated using three precisions [20–22]. We focus on the lower-precision solution of the linear system in line 4 in Algorithm 1. Based on this principle, this study aims to reduce the number of higher-precision matrix multiplications of the iterative refinement algorithm for singular value decomposition proposed by Ogita and Aishima.

2.3 Iterative refinement for symmetric eigenvalue decomposition

In this subsection, we introduce the iterative refinement algorithms for symmetric eigenvalue decomposition proposed by Ogita and Aishima in [23] and Uchino, Ozaki, and Ogita in [24]. We will write I_n to denote the $n \times n$ identity matrix. Now, we assume $A = A^T \in \mathbb{R}^{n\times n}$. Let $X \in \mathbb{R}^{n\times n}$ be orthogonal, $D \in \mathbb{R}^{n\times n}$ be diagonal, and $A = XDX^T$. The ith columns of X are the eigenvectors $x_{(i)} \in \mathbb{R}^n$ and the ith diagonal elements of D are the eigenvalues $\lambda_i \in \mathbb{R}$ for $i = 1, \ldots, n$. Here, we assume that $\lambda_i \neq \lambda_j$ for $i \neq j$. For $\hat{X} \approx X$, we define the error matrix $E \in \mathbb{R}^{n\times n}$ such that $X = \hat{X}(I_n + E)$. Ogita and Aishima proposed an algorithm to compute $\tilde{E} \approx E$ and update $\hat{X}$ into $\tilde{X}$ as $\tilde{X} \leftarrow \hat{X}(I_n + \tilde{E})$. The algorithm converges quadratically, provided that the error matrix E satisfies the following conditions from [25, Theorem 3.4]:

$$\|E\|_2 < \min\left(\frac{\min\limits_{1\le i<j\le n} |\lambda_i - \lambda_j|}{10\sqrt{n}\|A\|_2}, \frac{1}{100}\right). \tag{4}$$

This process is shown in Algorithm 2. Note that Algorithm 2 is simplified from the original algorithm [23, Algorithm 1] because multiple eigenvalues are not considered.

Algorithm 2 Refinement of approximate eigenvectors $\hat{X} \in \mathbb{R}^{n\times n}$ for a real symmetric matrix $A \in \mathbb{R}^{n\times n}$ from [23]. Assume $\tilde{\lambda}_i \neq \tilde{\lambda}_j$ for $i \neq j$. The total cost is $6n^3$ to $8n^3$ operations.

```
Require: $A \in \mathbb{R}^{n\times n}$, $\hat{X} \in \mathbb{R}^{n\times n}$
Ensure: $\tilde{X} \in \mathbb{R}^{n\times n}$, $\tilde{D} \in \mathbb{R}^{n\times n}$
1: function $[\tilde{X}, \tilde{D}] \leftarrow$ RefSyEv$(A, \hat{X})$
2:     $R \leftarrow (I_n - \hat{X}^T\hat{X})_h$                                  ▷ $R = (r_{ij})$
3:     $S \leftarrow (\hat{X}^T A\hat{X})_h$                                     ▷ $S = (s_{ij})$
4:     $\tilde{\lambda}_i \leftarrow \left(\frac{s_{ii}}{1-r_{ii}}\right)_h$ (for $1 \le i \le n$)      ▷ Approximate eigenvalues
5:     $\tilde{D} \leftarrow \mathrm{diag}(\tilde{\lambda}_1, \dots, \tilde{\lambda}_n)$
6:     $F \leftarrow (S + R\tilde{D})_h$                                        ▷ $F = (f_{ij})$
7:     $\tilde{e}_{ij} \leftarrow \begin{cases} \left(\frac{f_{ij}}{\tilde{\lambda}_j - \tilde{\lambda}_i}\right)_h & (i \neq j) \\ \left(\frac{r_{ij}}{2}\right)_h & (\text{otherwise}) \end{cases}$ (for $1 \le i, j \le n$)
8:     $\tilde{X} \leftarrow (\hat{X} + \hat{X}\tilde{E})_h$                       ▷ $\tilde{E} = (\tilde{e}_{ij})$
9: end function
```

Since $\hat{X}^T\hat{X}$ and $\hat{X}^T(A\hat{X})$ are symmetric, only the upper or lower triangular part needs to be computed. Thus, the total cost of Algorithm 2 is $6n^3$ operations. However, if the matrix product is obtained by using highly accurate algorithms, e.g., Ozaki's scheme [26] or a subroutine in an arbitrary precision arithmetic library, such as MPLAPACK [27][1], XBLAS (extra precise BLAS) [28], or Advanpix Multiprecision Computing Toolbox for MATLAB, the total cost is up to $8n^3$ operations since the symmetry of the matrix product may not be considered. The cost is divided into

$2n^3$: $A\hat{X}$ and $\hat{X}\tilde{E}$,
n^3: $\hat{X}^T\hat{X}$ and $\hat{X}^T(A\hat{X})$ after computing $A\hat{X}$ (exploiting symmetry),
$2n^3$: $\hat{X}^T\hat{X}$ and $\hat{X}^T(A\hat{X})$ after computing $A\hat{X}$ (no exploiting symmetry).

Let $e := \lceil \log_{10}(\|\tilde{E}\|_2) \rceil$. It is reported in [24] that the required arithmetic precision of high-accuracy matrix multiplications is $2e$ decimal digits for $\hat{X}^T\hat{X}$ and $\hat{X}^T A\hat{X}$ and e decimal digits for $\hat{X}\tilde{E}$ in Algorithm 2, i.e., line 8 in Algorithm 2 can be replaced by $\tilde{X} \leftarrow (\hat{X} + (\hat{X}\tilde{E})_\ell)_h$. Thus, the total cost of $2e$ decimal digit computations is $4n^3$ to $6n^3$ operations.

Uchino, Ozaki, and Ogita [24] proposed an algorithm to reduce the required arithmetic precision of Algorithm 2. It is based on iterative refinement for the solution of linear systems using mixed-precision arithmetic. From $R = I_n - \hat{X}^T\hat{X}$ and $S = \hat{X}^T A\hat{X}$ in Algorithm 2, F at line 6 satisfies

$$F = S + R\tilde{D} = \hat{X}^T A\hat{X} + (I_n - \hat{X}^T\hat{X})\tilde{D} = \hat{X}^T(A\hat{X} - \hat{X}\tilde{D}) + \tilde{D}.$$

[1] MPLAPACK provides a subroutine `rsyrk` for $\hat{X}^T\hat{X}$ with n^3 operations.

Since the diagonal part of F is not referenced, it can be computed as

$$F := \hat{X}^T W, \tag{5}$$

where $W := A\hat{X} - \hat{X}\tilde{D}$. From line 4 in Algorithm 1, (5) can be computed with lower-precision. Moreover, the off-diagonal parts of R and S are not necessary. Finally, a variant equivalent of Algorithm 2 in exact arithmetic is obtained. The condition for convergence is the same as (4) for Algorithm 2.

Algorithm 3 Refinement of approximate eigenvectors $\hat{X} \in \mathbb{R}^{n\times n}$ for a real symmetric matrix $A \in \mathbb{R}^{n\times n}$ from [24]. Assume $\tilde{\lambda}_i \neq \tilde{\lambda}_j$ for $i \neq j$. The total cost is $6n^3$ operations.

```
Require: $A \in \mathbb{R}^{n\times n}$, $\hat{X} = (x_{(1)}, \dots, x_{(n)}) \in \mathbb{R}^{n\times n}$
Ensure: $\tilde{X} \in \mathbb{R}^{n\times n}$, $\tilde{D} \in \mathbb{R}^{n\times n}$
1: function $[\tilde{X}, \tilde{D}] \leftarrow$ RefSyEv2$(A, \hat{X})$
2:     $P \leftarrow (A\hat{X})_h$                                        ▷ $P = (p_{(1)}, \dots, p_{(n)})$
3:     $r_{ii} \leftarrow (1 - \hat{x}_{(i)}^T \hat{x}_{(i)})_h$ (for $1 \le i \le n$)
4:     $s_{ii} \leftarrow (\hat{x}_{(i)}^T p_{(i)})_h$ (for $1 \le i \le n$)
5:     $\tilde{\lambda}_i \leftarrow \left(\frac{s_{ii}}{1-r_{ii}}\right)_h$ (for $1 \le i \le n$)          ▷ Approximate eigenvalues
6:     $\tilde{D} \leftarrow \mathrm{diag}(\tilde{\lambda}_1, \dots, \tilde{\lambda}_n)$
7:     $W \leftarrow (P - \hat{X}\tilde{D})_h$
8:     $F \leftarrow (\hat{X}^T W)_\ell$                                  ▷ $F = (f_{ij})$
9:     $\tilde{e}_{ij} \leftarrow \begin{cases} \left(\frac{f_{ij}}{\tilde{\lambda}_j - \tilde{\lambda}_i}\right)_h & (i \neq j) \\ \left(\frac{r_{ij}}{2}\right)_h & (\text{otherwise}) \end{cases}$ (for $1 \le i, j \le n$)
10:    $\tilde{X} \leftarrow (\hat{X} + (\hat{X}\tilde{E})_\ell)_h$                ▷ $\tilde{E} = (\tilde{e}_{ij})$
11: end function
```

The total cost of Algorithm 3 is $6n^3$ operations divided into

$2n^3$: $A\hat{X}$, $\hat{X}^T W$, and $\hat{X}\tilde{E}$,

among which there are no matrix multiplications whose result is symmetric. It is reported in [24] that the required arithmetic precision of high-accuracy matrix multiplications is $2e$ decimal digits for $A\hat{X}$ and e decimal digits for $\hat{X}^T W$ and $\hat{X}\tilde{E}$ in Algorithm 3. Thus, the total cost of $2e$ decimal digit computations is $2n^3$ operations.

2.4 Iterative refinement for singular value decomposition

In this subsection, we introduce an iterative refinement for full singular value decomposition proposed by Ogita and Aishima in [7]. Hereafter, we assume that

$$\sigma_1 > \sigma_2 > \cdots > \sigma_n$$

and the approximation $\tilde{\sigma}_i$ of σ_i satisfies $\tilde{\sigma}_i \neq \tilde{\sigma}_j$ for $i \neq j$. Let $\hat{U} \approx U$ and $\hat{V} \approx V$ for U and V in (1). Define the error matrices $F \in \mathbb{R}^{m\times m}$ and $G \in \mathbb{R}^{n\times n}$ such that

$$U = \hat{U}(I_m + F) \quad \text{and} \quad V = \hat{V}(I_n + G), \tag{6}$$

respectively. Ogita and Aishima proposed an algorithm which is based on the same idea as Algorithm 2. It computes $\tilde{F} \approx F$ and $\tilde{G} \approx G$, which hold from [7, Lemma 2], and updates $\hat{U}$ and $\hat{V}$ to $\tilde{U}$ and $\tilde{V}$ as $\tilde{U} \leftarrow \hat{U}(I_m + \tilde{F})$ and $\tilde{V} \leftarrow \hat{V}(I_n + \tilde{G})$, respectively. This process is shown in Algorithm 4. Note that for $T, \hat{U}, \tilde{U}, R \in \mathbb{R}^{m\times m}$, we have

$$T = (T_1\ T_2), \quad \hat{U} = (\hat{U}_1\ \hat{U}_2), \quad \tilde{U} = (\tilde{U}_1\ \tilde{U}_2), \quad \text{and} \quad R = \begin{pmatrix} R_{11} & R_{12} \\ R_{21} & R_{22} \end{pmatrix}$$

with $T_1, \hat{U}_1, \tilde{U}_1 \in \mathbb{R}^{m\times n}$, $T_2, \hat{U}_2, \tilde{U}_2 \in \mathbb{R}^{m\times(m-n)}$, $R_{11} \in \mathbb{R}^{n\times n}$, $R_{12}, R_{21}^T \in \mathbb{R}^{n\times(m-n)}$, and $R_{22} \in \mathbb{R}^{(m-n)\times(m-n)}$.

Let $\tilde{U}$ and $\tilde{V}$ be obtained from $\hat{U}$ and $\hat{V}$ using Algorithm 4. Define $F' \in \mathbb{R}^{m\times m}$ and $G' \in \mathbb{R}^{n\times n}$ as $U = \tilde{U}(I_m + F')$ and $V = \tilde{V}(I_n + G')$, respectively, and $\varepsilon := \max(\|F\|_2, \|G\|_2)$, $\varepsilon' := \max(\|F'\|_2, \|G'\|_2)$. If ε satisfies

$$\varepsilon < \frac{\min\limits_{1\le i\le n-1}(\sigma_i - \sigma_{i+1})}{30m\|A\|_2}, \tag{7}$$

then

$$\varepsilon' < \frac{7}{10}\varepsilon \quad \text{and} \quad \limsup_{\varepsilon\to 0} \frac{\varepsilon'}{\varepsilon^2} \le \frac{18m\|A\|_2}{\min\limits_{1\le i\le n-1}(\sigma_i - \sigma_{i+1})} \tag{8}$$

hold from [7, Theorem 1], and (8) implies that Algorithm 4 converges quadratically. For Algorithm 4, we assume that these are clustered singular values if (7) is not satisfied.

The total cost of Algorithm 4 is $3m^3 + 2m^2n + 2mn^2 + 3n^3$ operations; however, the cost can be up to $4m^3 + 2m^2n + 2mn^2 + 4n^3$ operations due to the computational problem of symmetry of the result of $\hat{U}^T\hat{U}$ and $\hat{V}^T\hat{V}$. The cost is divided into

m^3:	$\hat{U}^T\hat{U}$ (exploiting symmetry),
$2m^3$:	$\hat{U}^T\hat{U}$ (no exploiting symmetry),
n^3:	$\hat{V}^T\hat{V}$ (exploiting symmetry),
$2n^3$:	$\hat{V}^T\hat{V}$ (no exploiting symmetry),
$2mn^2$:	$A\hat{V}$,
$2m^2n$:	$\hat{U}^T(A\hat{V})$ after computing $A\hat{V}$,
$2m^3$:	$\hat{U}\tilde{F}$,
$2n^3$:	$\hat{V}\tilde{G}$.

Algorithm 4 Refinement of approximate singular vectors $\hat{U} \in \mathbb{R}^{m\times m}$ and $\hat{V} \in \mathbb{R}^{n\times n}$ for a real matrix $A \in \mathbb{R}^{m\times n}$ from [7]. Assume $\tilde{\sigma}_i \neq \tilde{\sigma}_j$ for $i \neq j$. The total cost is $3m^3 + 2m^2n + 2mn^2 + 3n^3$ to $4m^3 + 2m^2n + 2mn^2 + 4n^3$ operations.

```
Require: $A \in \mathbb{R}^{m\times n}, \hat{U} \in \mathbb{R}^{m\times m}, \hat{V} \in \mathbb{R}^{n\times n}$
Ensure: $\tilde{U} \in \mathbb{R}^{m\times m}, \tilde{\Sigma} \in \mathbb{R}^{m\times n}, \tilde{V} \in \mathbb{R}^{n\times n}$
 1: function $[\tilde{U}, \tilde{\Sigma}, \tilde{V}] \leftarrow$ RefSVD$(A, \hat{U}, \hat{V})$
 2:     $R \leftarrow (I_m - \hat{U}^T\hat{U})_h$                                  ▷ $R = (r_{ij})$
 3:     $S \leftarrow (I_n - \hat{V}^T\hat{V})_h$                                  ▷ $S = (s_{ij})$
 4:     $T \leftarrow (\hat{U}^T A\hat{V})_h$                                      ▷ $T = (t_{ij})$
 5:     $\tilde{\sigma}_i \leftarrow \left(\frac{t_{ii}}{1-(r_{ii}+s_{ii})/2}\right)_h$ (for $1 \le i \le n$)   ▷ Approximate singular values
 6:     $\tilde{\Sigma}_n \leftarrow \mathrm{diag}(\tilde{\sigma}_1, \dots, \tilde{\sigma}_n)$; $\tilde{\Sigma} \leftarrow (\tilde{\Sigma}_n, O_{n,m-n})^T$
 7:     $C_\alpha \leftarrow (T_1 + R_{11}\tilde{\Sigma}_n)_h$; $C_\beta \leftarrow (T_1^T + S\tilde{\Sigma}_n)_h$
 8:     $D \leftarrow (\tilde{\Sigma}_n C_\alpha + C_\beta\tilde{\Sigma}_n)_h$                  ▷ $D = (d_{ij})$
 9:     $E \leftarrow (C_\alpha\tilde{\Sigma}_n + \tilde{\Sigma}_n C_\beta)_h$                  ▷ $E = (e_{ij})$
10:     $\tilde{g}_{ij} \leftarrow \begin{cases} \left(\frac{d_{ij}}{\tilde{\sigma}_j^2-\tilde{\sigma}_i^2}\right)_h & (i \neq j) \\ \left(\frac{s_{ii}}{2}\right)_h & (\text{otherwise}) \end{cases}$ (for $1 \le i, j \le n$)
11:     $\tilde{f}_{ij} \leftarrow \begin{cases} \left(\frac{e_{ij}}{\tilde{\sigma}_j^2-\tilde{\sigma}_i^2}\right)_h & (i \neq j,\ i, j \le n) \\ \left(-\frac{t_{ji}}{\tilde{\sigma}_i}\right)_h & (i \le n < j) \\ \left(r_{ij} - \tilde{f}_{ji}\right)_h & (j \le n < i) \\ \left(\frac{r_{ij}}{2}\right)_h & (\text{otherwise}) \end{cases}$ (for $1 \le i, j \le m$)
12:     $\tilde{U} \leftarrow (\hat{U} + \hat{U}\tilde{F})_h$; $\tilde{V} \leftarrow (\hat{V} + \hat{V}\tilde{G})_h$           ▷ $\tilde{F} = (\tilde{f}_{ij}), \tilde{G} = (\tilde{g}_{ij})$
13: end function
```

Here, we define $\tilde{\varepsilon} := \max(\|\tilde{F}\|_2, \|\tilde{G}\|_2)$ and

$$d := \lceil -\log_{10}\tilde{\varepsilon}\rceil. \tag{9}$$

Then, using (6) and $\varepsilon \approx \tilde{\varepsilon}$ from $F \approx \tilde{F}$ and $G \approx \tilde{G}$ yields

$$\max\left(\frac{\|U-\hat{U}\|_2}{\|\hat{U}\|_2}, \frac{\|V-\hat{V}\|_2}{\|\hat{V}\|_2}\right) \le \varepsilon \approx \tilde{\varepsilon} \approx \mathcal{O}(10^{-d}). \tag{10}$$

Since (8) holds,

$$\begin{aligned}\max\left(\frac{\|U-\tilde{U}\|_2}{\|\tilde{U}\|_2}, \frac{\|V-\tilde{V}\|_2}{\|\tilde{V}\|_2}\right) &\le \varepsilon' \\ &\le \frac{18m\|A\|_2}{\min\limits_{1\le i\le n-1}(\sigma_i - \sigma_{i+1})}\varepsilon^2 \\ &\approx \frac{18m\|A\|_2}{\min\limits_{1\le i\le n-1}(\sigma_i - \sigma_{i+1})}\tilde{\varepsilon}^2 \approx \mathcal{O}(10^{-2d})\end{aligned}$$

as $\varepsilon \to 0$. Thus, the required arithmetic precision of Algorithm 4 is almost $2d$ decimal digits. However, that for the computations of $\hat{U}\tilde{F}$ and $\hat{V}\tilde{G}$ at line 12 in Algorithm 4 is about d decimal digits. The reason for this is the same as for the required arithmetic precision for $\hat{X}\tilde{E}$ in Algorithm 2 and 3—since the first d decimal digits or so of $\hat{U}$ and $\hat{V}$ are correct from (10), only the first d decimal digits or so of $\hat{U}\tilde{F}$ and $\hat{V}\tilde{G}$ can affect the results. This situation is depicted in Fig. 1. From the above considerations, line 12 in Algorithm 4 is replaced by $\tilde{U} \leftarrow (\hat{U} + (\hat{U}\tilde{F})_\ell)_h;\ \ \tilde{V} \leftarrow (\hat{V} + (\hat{V}\tilde{G})_\ell)_h$. Therefore, the total cost of $2d$ and d decimal digit computations is $m^3 + 2m^2n + 2mn^2 + n^3$ to $2m^3 + 2m^2n + 2mn^2 + 2n^3$ operations and $2m^3 + 2n^3$ operations, respectively.

2.5 Highly accurate matrix multiplication

We assume floating-point operations in rounding to nearest value according to IEEE Std. 754 [29] and no overflow or underflow. Let $\mathbb{F}$ be a set of floating-point numbers and $\mathrm{fl}(\cdot)$ denote the result of a floating-point operation, where all operations inside the parentheses are executed in ordinary precision. Ozaki, Ogita, Oishi, and Rump proposed an algorithm for error-free transformation of matrix multiplication called Ozaki's scheme [26]. Their algorithm transforms $A \in \mathbb{R}^{m\times k}$ and $B \in \mathbb{R}^{k\times n}$ into

$$A = \sum_{p=1}^{n_A-1} A^{(p)} + \underline{A}^{(n_A)}, \quad A^{(p)}, \underline{A}^{(n_A)} \in \mathbb{F}^{m\times k}$$
$$B = \sum_{q=1}^{n_B-1} B^{(q)} + \underline{B}^{(n_B)}, \quad B^{(q)}, \underline{B}^{(n_B)} \in \mathbb{F}^{k\times n},$$

where

$$\underline{A}^{(s)} := A - \sum_{p=1}^{s-1} A^{(p)}, \quad \underline{B}^{(s)} := B - \sum_{p=1}^{s-1} B^{(p)},$$

and

$$|a_{ij}^{(s)}| \ge |a_{ij}^{(t)}| \quad \text{if } a_{ij}^{(s)} \ne 0 \quad \text{for } s < t,$$
$$|b_{ij}^{(s)}| \ge |b_{ij}^{(t)}| \quad \text{if } b_{ij}^{(s)} \ne 0 \quad \text{for } s < t$$

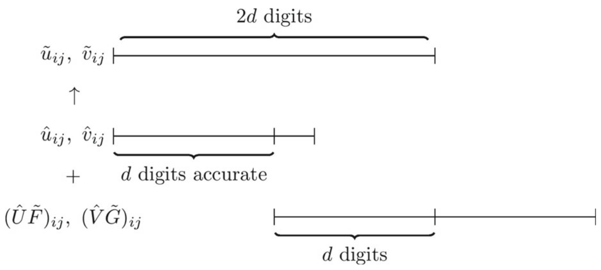
$2d$ digits
$\tilde{u}_{ij},\ \tilde{v}_{ij}$
$\uparrow$
$\hat{u}_{ij},\ \hat{v}_{ij}$
$+$
d digits accurate
$(\hat{U}\tilde{F})_{ij},\ (\hat{V}\tilde{G})_{ij}$
d digits

Fig. 1 Diagram representing $\tilde{U} \leftarrow \hat{U} + \hat{U}\tilde{F}$ and $\tilde{V} \leftarrow \hat{V} + \hat{V}\tilde{G}$ from Algorithm 4

as described in [26, Algorithm 3]. In this case, there is no rounding error in $\mathrm{fl}(A^{(p)}B^{(q)})$ for all (p, q) pairs. In this study, we compute $(AB)_h$ as

$$\left(\sum_{p+q \le \max(nA, nB)} (A^{(p)}B^{(q)})_\ell + \sum_{p=1}^{n_A-1} (A^{(p)}\underline{B}^{(n_B-p+1)})_\ell + (\underline{A}^{(n_A)}B)_\ell \right)_h . \tag{11}$$

We set $n_A = n_B = 4$ in this paper.

3 Proposed algorithms

In this section, we propose four faster algorithms for the iterative refinement of singular vectors.

3.1 Proposed algorithms based on iterative refinement for singular value decomposition

Here, we propose two algorithms based on Algorithm 4. From $R = I_m - \hat{U}^T\hat{U}$, $S = I_n - \hat{V}^T\hat{V}$, and $T = \hat{U}^T A\hat{V}$ in Algorithm 4, C_α and C_β at line 7 satisfy

$$\begin{aligned} C_\alpha &= T_1 + R_{11}\tilde{\Sigma}_n \\ &= \hat{U}_1^T A\hat{V} + (I_n - \hat{U}_1^T\hat{U}_1)\tilde{\Sigma}_n \\ &= \hat{U}_1^T(A\hat{V} - \hat{U}_1\tilde{\Sigma}_n) + \tilde{\Sigma}_n, \\ C_\beta &= T_1^T + S\tilde{\Sigma}_n \\ &= \hat{V}^T A^T\hat{U}_1 + (I_n - \hat{V}^T\hat{V})\tilde{\Sigma}_n \\ &= \hat{V}^T(A^T\hat{U}_1 - \hat{V}\tilde{\Sigma}_n) + \tilde{\Sigma}_n. \end{aligned}$$

Since the diagonal parts of C_α and C_β are not necessary because the diagonal parts of D and E are not used, they can be computed as

$$\begin{aligned} C_\alpha &= \hat{U}_1^T(A\hat{V} - \hat{U}_1\tilde{\Sigma}_n), \\ C_\beta &= \hat{V}^T(A^T\hat{U}_1 - \hat{V}\tilde{\Sigma}_n). \end{aligned}$$

Let $\tilde{F}_{11} \in \mathbb{R}^{n\times n}$, $\tilde{F}_{12} \in \mathbb{R}^{n\times(m-n)}$, $\tilde{F}_{21} \in \mathbb{R}^{(m-n)\times n}$, $\tilde{F}_{22} \in \mathbb{R}^{(m-n)\times(m-n)}$ such that

$$\tilde{F} = \begin{pmatrix} \tilde{F}_{11} & \tilde{F}_{12} \\ \tilde{F}_{21} & \tilde{F}_{22} \end{pmatrix}.$$

Then, from line 11 in Algorithm 4, the following holds:

$$(\tilde{F}_{11})_{ij} = \begin{cases} \frac{e_{ij}}{\tilde{\sigma}_j - \tilde{\sigma}_i} & (i \neq j) \\ \frac{r_{ij}}{2} & (\text{otherwise}) \end{cases} \quad (\text{for } 1 \leq i, j \leq n),$$

$$\tilde{F}_{12} = -\tilde{\Sigma}_n^{-1} T_2^T = -\tilde{\Sigma}_n^{-1} \hat{V}^T A^T \hat{U}_2,$$

$$\tilde{F}_{21} = R_{21} - \tilde{F}_{12}^T = -\hat{U}_2^T \hat{U}_1 + \hat{U}_2^T A \hat{V} \tilde{\Sigma}_n^{-1} = \hat{U}_2^T (A\hat{V} - \hat{U}_1 \tilde{\Sigma}_n) \tilde{\Sigma}_n^{-1},$$

$$\tilde{F}_{22} = \frac{1}{2} R_{22} = \frac{1}{2}(I_{m-n} - \hat{U}_2^T \hat{U}_2).$$

Thus, only the computations for diag(R_{11}), R_{22}, and T_2 are necessary. Finally, we obtain the following algorithm, which is a variant equivalent to Algorithm 4 in exact arithmetic.

Algorithm 5 Refinement of approximate singular vectors $\hat{U} \in \mathbb{R}^{m\times m}$ and $\hat{V} \in \mathbb{R}^{n\times n}$ for a real matrix $A \in \mathbb{R}^{m\times n}$. Assume $\tilde{\sigma}_i \neq \tilde{\sigma}_j$ for $i \neq j$. The total cost is $3m^3 + 2m^2n + 3mn^2 + 4n^3$ to $4m^3 + 4mn^2 + 4n^3$ operations. Bold font indicates differences from Algorithm 4.

```
Require: $A \in \mathbb{R}^{m\times n}$, $\hat{U} \in \mathbb{R}^{m\times m}$, $\hat{V} \in \mathbb{R}^{n\times n}$
Ensure: $\tilde{U} \in \mathbb{R}^{m\times m}$, $\tilde{\Sigma} \in \mathbb{R}^{m\times n}$, $\tilde{V} \in \mathbb{R}^{n\times n}$
1: function $[\tilde{U}, \tilde{\Sigma}, \tilde{V}] \leftarrow$ RefSVD2$(A, \hat{U}, \hat{V})$
2:     $\boldsymbol{P \leftarrow (A\hat{V})_h}$    ▷ $P = (p_{(1)}, \dots, p_{(n)})$
3:     $\boldsymbol{Q \leftarrow (A^T \hat{U}_1)_h}$    ▷ $Q = (q_{(1)}, \dots, q_{(n)})$
4:     $\boldsymbol{r_{ii} \leftarrow (1 - \hat{u}_{(i)}^T \hat{u}_{(i)})_h}$ (for $\boldsymbol{1 \leq i \leq n}$)
5:     $\boldsymbol{s_{ii} \leftarrow (1 - \hat{v}_{(i)}^T \hat{v}_{(i)})_h}$ (for $\boldsymbol{1 \leq i \leq n}$)
6:     $\boldsymbol{t_{ii} \leftarrow (\hat{u}_{(i)}^T \hat{p}_{(i)})_h}$ (for $\boldsymbol{1 \leq i \leq n}$)
7:     $\tilde{\sigma}_i \leftarrow \left(\frac{t_{ii}}{1-(r_{ii}+s_{ii})/2}\right)_h$ (for $1 \leq i \leq n$)    ▷ Approximate singular values
8:     $\tilde{\Sigma}_n \leftarrow \mathrm{diag}(\tilde{\sigma}_1, \dots, \tilde{\sigma}_n)$; $\tilde{\Sigma} \leftarrow (\tilde{\Sigma}_n, O_{n,m-n})^T$
9:     $\boldsymbol{C_\gamma \leftarrow (P - \hat{U}_1 \tilde{\Sigma}_n)_h;\ C_\delta \leftarrow (Q - \hat{V}\tilde{\Sigma}_n)_h;}$
10:    $\boldsymbol{C_\alpha \leftarrow (\hat{U}_1^T C_\gamma)_\ell;\ C_\beta \leftarrow (\hat{V}^T C_\delta)_\ell}$
11:    $D \leftarrow (\tilde{\Sigma}_n C_\alpha + C_\beta \tilde{\Sigma}_n)_h$    ▷ $D = (d_{ij})$
12:    $E \leftarrow (C_\alpha \tilde{\Sigma}_n + \tilde{\Sigma}_n C_\beta)_h$    ▷ $E = (e_{ij})$
13:    $\tilde{g}_{ij} \leftarrow \begin{cases} \left(\frac{d_{ij}}{\tilde{\sigma}_j^2 - \tilde{\sigma}_i^2}\right)_h & (i \neq j) \\ \left(\frac{s_{ii}}{2}\right)_h & (\text{otherwise}) \end{cases}$ (for $1 \leq i, j \leq n$)
14:    $\tilde{f}_{ij} \leftarrow \begin{cases} \left(\frac{e_{ij}}{\tilde{\sigma}_j^2 - \tilde{\sigma}_i^2}\right)_h & (i \neq j) \\ \left(\frac{r_{ij}}{2}\right)_h & (\text{otherwise}) \end{cases}$ (for $1 \leq i, j \leq n$)    ▷ $\tilde{F}_{11} = (\tilde{f}_{ij})$
15:    $\boldsymbol{\tilde{F}_{12} \leftarrow (\tilde{\Sigma}_n^{-1} P^T \hat{U}_2)_h}$
16:    $\boldsymbol{\tilde{F}_{21} \leftarrow ((\hat{U}_2^T C_\gamma)_\ell \tilde{\Sigma}_n^{-1})_h}$
17:    $\boldsymbol{\tilde{F}_{22} \leftarrow \left(\frac{1}{2}(I_{m-n} - \hat{U}_2^T \hat{U}_2)\right)_h}$
18:    $\tilde{U} \leftarrow (\hat{U} + (\hat{U}\tilde{F})_\ell)_h$; $\tilde{V} \leftarrow (\hat{V} + (\hat{V}\tilde{G})_\ell)_h$    ▷ $\tilde{G} = (\tilde{g}_{ij})$
19: end function
```

The total cost of Algorithm 5 is $3m^3 + 2m^2n + 3mn^2 + 4n^3$ to $4m^3 + 4mn^2 + 4n^3$ operations divided into

$2n^3$:	$\hat{V}^T C_\delta$,
$2mn^2$:	$A\hat{V}$, $A^T\hat{U}_1$, $\hat{U}_1^T C_\gamma$,
$2mn(m-n)$:	$P^T\hat{U}_2$, $\hat{U}_2^T C_\gamma$,
$m(m-n)^2$:	$\hat{U}_2^T\hat{U}_2$ (exploiting symmetry),
$2m(m-n)^2$:	$\hat{U}_2^T\hat{U}_2$ (no exploiting symmetry),
$2m^3$:	$\hat{U}\tilde{F}$,
$2n^3$:	$\hat{V}\tilde{G}$

and is more expensive than that of Algorithm 4. The difference between the cost of Algorithms 4 and 5 is $mn^2 + n^3$ to $2m^2n - 2mn^2$ operations. For d in (9), the required arithmetic precision for the computations of $\hat{U}_1^T C_\gamma$, $\hat{V}^T C_\delta$, $\hat{U}_2^T C_\gamma$, $\hat{U}\tilde{F}$, and $\hat{V}\tilde{G}$ is d decimal digits, while that for the other computations is $2d$ decimal digits. Therefore, the total cost of $2d$ and d decimal digits higher-precision computations is $m^3 + 3mn^2$ to $2m^3 - 2m^2n + 4mn^2$ operations and $2m^3 + 2m^2n + 4n^3$ operations, respectively. The cost of $2d$ decimal digit computations in Algorithm 5 is $2m^2n - mn^2 + n^3$ to $4m^2n - 2mn^2 + 2n^3$ operations less than that of Algorithm 4.

We propose another algorithm with lower total cost than Algorithm 5. From C_α and C_β at line 7 in Algorithm 4 and $R = R^T$, the following holds:

$$\begin{aligned} C_\beta &= T_1^T + S\tilde{\Sigma}_n \\ &= \hat{V}^T A^T \hat{U} + S\tilde{\Sigma}_n \\ &= \hat{V}^T A^T \hat{U} + \tilde{\Sigma}_n R - \tilde{\Sigma}_n R + S\tilde{\Sigma}_n \\ &= (\hat{U}^T A \hat{V} + R\tilde{\Sigma}_n)^T - \tilde{\Sigma}_n R + S\tilde{\Sigma}_n \\ &= C_\alpha^T - \tilde{\Sigma}_n R + S\tilde{\Sigma}_n. \end{aligned} \tag{12}$$

Thus, we obtain the following algorithm combining Algorithms 4, 5, and (12).

The total cost of Algorithm 6 is $3m^3 + 2m^2n + 2mn^2 + 3n^3$ to $4m^3 + 4mn^2 + 4n^3$ operations divided into

n^3:	$\hat{V}^T\hat{V}$ (exploiting symmetry),
$2n^3$:	$\hat{V}^T\hat{V}$ (no exploiting symmetry),
$2mn^2$:	$A\hat{V}$, $\hat{U}_1^T\hat{U}_1$, $\hat{U}_1^T C_\gamma$,
$2mn(m-n)$:	$P^T\hat{U}_2$, $\hat{U}_2^T C_\gamma$,
$m(m-n)^2$:	$\hat{U}_2^T\hat{U}_2$ (exploiting symmetry),
$2m(m-n)^2$:	$\hat{U}_2^T\hat{U}_2$ (no exploiting symmetry),
$2m^3$:	$\hat{U}\tilde{F}$,
$2n^3$:	$\hat{V}\tilde{G}$,

Algorithm 6 Refinement of approximate singular vectors $\hat{U} \in \mathbb{R}^{m\times m}$ and $\hat{V} \in \mathbb{R}^{n\times n}$ for a real matrix $A \in \mathbb{R}^{m\times n}$. Assume $\tilde{\sigma}_i \neq \tilde{\sigma}_j$ for $i \neq j$. The total cost is $3m^3 + 2m^2n + 2mn^2 + 3n^3$ to $4m^3 + 4mn^2 + 4n^3$ operations. Bold font indicates differences from Algorithm 5.

```
Require: $A \in \mathbb{R}^{m\times n}$, $\hat{U} \in \mathbb{R}^{m\times m}$, $\hat{V} \in \mathbb{R}^{n\times n}$
Ensure: $\tilde{U} \in \mathbb{R}^{m\times m}$, $\tilde{\Sigma} \in \mathbb{R}^{m\times n}$, $\tilde{V} \in \mathbb{R}^{n\times n}$
1: function $[\tilde{U}, \tilde{\Sigma}, \tilde{V}] \leftarrow$ RefSVD3$(A, \hat{U}, \hat{V})$
2:     $P \leftarrow (A\hat{V})_h$                                    ▷ $P = (p_{(1)}, \ldots, p_{(n)})$
3:     $\boldsymbol{R \leftarrow (I_n - \hat{U}_1^T\hat{U}_1)_h}$                                    ▷ $R = (r_{ij})$
4:     $\boldsymbol{S \leftarrow (I_n - \hat{V}^T\hat{V})_h}$                                    ▷ $S = (s_{ij})$
5:     $t_{ii} \leftarrow (\hat{u}_{(i)}^T p_{(i)})_h$ (for $1 \le i \le n$)
6:     $\tilde{\sigma}_i \leftarrow \left(\frac{t_{ii}}{1-(r_{ii}+s_{ii})/2}\right)_h$ (for $1 \le i \le n$)                ▷ Approximate singular values
7:     $\tilde{\Sigma}_n \leftarrow \mathrm{diag}(\tilde{\sigma}_1, \ldots, \tilde{\sigma}_n)$; $\tilde{\Sigma} \leftarrow (\tilde{\Sigma}_n, O_{n,m-n})^T$
8:     $C_\gamma \leftarrow (P - \hat{U}_1\tilde{\Sigma}_n)_h$
9:     $C_\alpha \leftarrow (\hat{U}_1^T C_\gamma)_\ell$; $\boldsymbol{C_\beta \leftarrow (C_\alpha^T - \tilde{\Sigma}_n R + S\tilde{\Sigma}_n)_h}$
10:    $D \leftarrow (\tilde{\Sigma}_n C_\alpha + C_\beta\tilde{\Sigma}_n)_h$                                    ▷ $D = (d_{ij})$
11:    $E \leftarrow (C_\alpha\tilde{\Sigma}_n + \tilde{\Sigma}_n C_\beta)_h$                                    ▷ $E = (e_{ij})$
12:    $\tilde{g}_{ij} \leftarrow \begin{cases} \left(\frac{d_{ij}}{\tilde{\sigma}_j^2 - \tilde{\sigma}_i^2}\right)_h & (i \neq j) \\ \left(\frac{s_{ii}}{2}\right)_h & \text{(otherwise)} \end{cases}$ (for $1 \le i, j \le n$)
13:    $\tilde{f}_{ij} \leftarrow \begin{cases} \left(\frac{e_{ij}}{\tilde{\sigma}_j^2 - \tilde{\sigma}_i^2}\right)_h & (i \neq j) \\ \left(\frac{r_{ij}}{2}\right)_h & \text{(otherwise)} \end{cases}$ (for $1 \le i, j \le n$)                ▷ $\tilde{F}_{11} = (\tilde{f}_{ij})$
14:    $\tilde{F}_{12} \leftarrow (\tilde{\Sigma}_n^{-1} P^T \hat{U}_2)_h$
15:    $\tilde{F}_{21} \leftarrow ((\hat{U}_2^T C_\gamma)_\ell \tilde{\Sigma}_n^{-1})_h$
16:    $\tilde{F}_{22} \leftarrow \left(\frac{1}{2}(I_{m-n} - \hat{U}_2^T\hat{U}_2)\right)_h$
17:    $\tilde{U} \leftarrow (\hat{U} + (\hat{U}\tilde{F})_\ell)_h$; $\tilde{V} \leftarrow (\hat{V} + (\hat{V}\tilde{G})_\ell)_h$                ▷ $\tilde{G} = (\tilde{g}_{ij})$
18: end function
```

which is a little more expensive than Algorithm 4. The difference between the costs of Algorithms 4 and 6 is 0 to $2m^2n - 2mn^2$ operations, i.e., the minimum costs of Algorithm 4 and 6 are the same. For d in (9), the required arithmetic precision for the computations of $\hat{U}_1^T C_\gamma$, $\hat{U}_2^T C_\gamma$, $\hat{U}\tilde{F}$, and $\hat{V}\tilde{G}$ is d decimal digits, while that for the other computations is $2d$ decimal digits. Therefore, the total cost of $2d$ and d decimal digit computations is $m^3 + 2mn^2 + n^3$ to $2m^3 - 2m^2n + 4mn^2 + 2n^3$ operations and $2m^3 + 2m^2n + 2n^3$ operations, respectively. The cost of $2d$ decimal digit computations in Algorithm 6 is $2m^2n$ to $4m^2n - 2mn^2$ operations less than that of Algorithm 4.

3.2 Proposed algorithms based on iterative refinement for symmetric eigenvalue decomposition

Here, we propose two algorithms based on Algorithm 3. The singular value decomposition is easily extended to eigenvalue decomposition. From (1),

$$\begin{aligned} A^TA &= V\Sigma^T\Sigma V^T \quad \text{and} \\ AA^T &= U\Sigma\Sigma^T U^T \end{aligned} \tag{13}$$

are satisfied, and these represent the eigenvalue decomposition of A^TA and AA^T, respectively. From Algorithms 3, 5, and (13), we immediately obtain the following algorithm. Note that $\tilde{V}$ obtained using Algorithm 7 may not converge as well as Algorithm 4 because Algorithm 7 does not use $\hat{V}$.

Algorithm 7 Refinement of approximate singular vectors $\hat{U} \in \mathbb{R}^{m\times m}$ and $\hat{V} \in \mathbb{R}^{n\times n}$ for a real matrix $A \in \mathbb{R}^{m\times n}$. Assume $\tilde{\sigma}_i \neq \tilde{\sigma}_j$ for $i \neq j$. The total cost is $3m^3+4m^2n-mn^2$ to $4m^3+2m^2n$ operations. Bold font indicates differences from Algorithm 5.

```
Require: $A \in \mathbb{R}^{m\times n}$, $B \leftarrow (AA^T)_h \in \mathbb{R}^{m\times m}$, $\hat{U} \in \mathbb{R}^{m\times m}$
Ensure: $\tilde{U} \in \mathbb{R}^{m\times m}$, $\tilde{\Sigma} \in \mathbb{R}^{m\times n}$
1: function $[\tilde{U}, \tilde{\Sigma}] \leftarrow$ RefSVD4$(B, \hat{U})$
2:     $\boldsymbol{P \leftarrow (B\hat{U}_1)_h}$                                   ▷ $P = (p_{(1)}, \dots, p_{(n)})$
3:     $r_{ii} \leftarrow (1 - \hat{u}_{(i)}^T\hat{u}_{(i)})_h$ (for $1 \le i \le n$)
4:     $\boldsymbol{s_{ii} \leftarrow (\hat{u}_{(i)}^T p_{(i)})_h}$ (for $\boldsymbol{1 \le i \le n}$)
5:     $\boldsymbol{\tilde{\lambda}_i \leftarrow \left(\frac{s_{ii}}{1-r_{ii}}\right)_h}$ (for $\boldsymbol{1 \le i \le n}$)
6:     $\boldsymbol{D \leftarrow \mathrm{diag}(\tilde{\lambda}_1, \dots, \tilde{\lambda}_n)}$
7:     $\boldsymbol{\tilde{\sigma}_i \leftarrow (\sqrt{\tilde{\lambda}_i})_h}$ (for $\boldsymbol{1 \le i \le n}$)                ▷ Approximate singular values
8:     $\tilde{\Sigma}_n \leftarrow \mathrm{diag}(\tilde{\sigma}_1, \dots, \tilde{\sigma}_n)$; $\tilde{\Sigma} \leftarrow (\tilde{\Sigma}_n, O_{n,m-n})^T$
9:     $\boldsymbol{C_\gamma \leftarrow (P - \hat{U}_1 D)_h}$
10:    $E \leftarrow (\hat{U}_1^T C_\gamma)_\ell$                                   ▷ $E = (e_{ij})$
11:    $\tilde{f}_{ij} \leftarrow \begin{cases} \left(\frac{e_{ij}}{\tilde{\lambda}_j - \tilde{\lambda}_i}\right)_h & (i \neq j) \\ \left(\frac{r_{ij}}{2}\right)_h & \text{(otherwise)} \end{cases}$ (for $1 \le i, j \le n$)      ▷ $\tilde{F}_{11} = (\tilde{f}_{ij})$
12:    $\tilde{F}_{12} \leftarrow (\tilde{\Sigma}_n^{-1} P^T \hat{U}_2)_h$
13:    $\tilde{F}_{21} \leftarrow ((\hat{U}_2^T C_\gamma)_\ell \tilde{\Sigma}_n^{-1})_h$
14:    $\tilde{F}_{22} \leftarrow \left(\frac{1}{2}(I_{m-n} - \hat{U}_2^T\hat{U}_2)\right)_h$
15:    $\tilde{U} \leftarrow (\hat{U} + (\hat{U}\tilde{F})_\ell)_h$                           ▷ $\tilde{F} = (\tilde{f}_{ij})$
16: end function
```

We can obtain $\tilde{V}$ as $\tilde{V} \leftarrow A^T\tilde{U}_1\tilde{\Sigma}_n^{-1}$. The costs for $B \leftarrow AA^T$ and $\tilde{V} \leftarrow A^T\tilde{U}_1\tilde{\Sigma}_n^{-1}$ are m^2n to $2m^2n$ and $2mn^2$, respectively. Thus, the total cost of ν iterations of Algorithm 7 is $(3m^3+4m^2n-mn^2)\nu + m^2n + 2mn^2$ to $(4m^3+2m^2n)\nu + 2m^2n + 2mn^2$ operations divided into

m^2n:	AA^T (exploiting symmetry),
$2m^2n$:	AA^T (no exploiting symmetry),
$2mn^2$:	$A^T\tilde{U}_1$,
$2m^2n\nu$:	$B\hat{U}_1$,
$2mn^2\nu$:	$\hat{U}_1^T C_\gamma$,

$2mn(m-n)\nu$:	$P^T\hat{U}_2, \hat{U}_2^T C_\gamma$,
$m(m-n)^2$:	$\hat{U}_2^T\hat{U}_2$ (exploiting symmetry),
$2m(m-n)^2$:	$\hat{U}_2^T\hat{U}_2$ (no exploiting symmetry),
$2m^3\nu$:	$\hat{U}\tilde{F}$,

which is less expensive than Algorithm 4. The difference between the costs of Algorithms 4 and 7 is $(2m^2n-3mn^2-3n^3)\nu+m^2n+2mn^2$ to $(-2mn^2-4n^3)\nu+2m^2n+2mn^2$ operations. For d in (9), the required arithmetic precision for the computations of $\hat{U}_1^T C_\gamma$, $\hat{U}_2^T C_\gamma$, and $\hat{U}\tilde{F}$ is d decimal digits, while that for the other computations is $2d$ decimal digits. Therefore, the total cost of $2d$ and d decimal digit computations is $(m^3+2m^2n-mn^2)\nu+m^2n+2mn^2$ to $2m^3\nu+2m^2n+2mn^2$ operations and $(2m^3+2m^2n)\nu$ operations, respectively. The cost of $2d$ decimal digit computations in Algorithm 7 is $(n^3+3mn^2)\nu-m^2n-2mn^2$ to $(2m^2n+2mn^2+2n^3)\nu-2m^2n-2mn^2$ operations less than that of Algorithm 4.

We now introduce another extension of the singular value decomposition to the eigenvalue decomposition. We will write $O_{m,n}$ to denote the $m\times n$ zero matrix. Let $\Sigma_n := \mathrm{diag}(\sigma_1,\ldots,\sigma_n)\in\mathbb{R}^{n\times n}$. The eigenvalues of

$$B := \begin{pmatrix} O_{n,n} & A^T \\ A & O_{m,m}\end{pmatrix} \in \mathbb{R}^{(m+n)\times(m+n)} \tag{14}$$

are $\sigma_1,\ldots,\sigma_n,-\sigma_1,\ldots,-\sigma_n,0,\ldots,0$ from [30], and for $X, D\in\mathbb{R}^{(m+n)\times(m+n)}$ such that

$$X := \frac{1}{\sqrt{2}}\begin{pmatrix} V & V & O_{n,m-n} \\ U_1 & -U_1 & \sqrt{2}U_2\end{pmatrix},\quad D := \begin{pmatrix} \Sigma_n & O_{n,n} & O_{n,m-n} \\ O_{m,n} & -\Sigma & O_{m,m-n}\end{pmatrix},$$

it holds that

$$B = XDX^T \tag{15}$$

from [31].

We consider transforming the singular value decomposition into the symmetric eigenvalue decomposition as (15) and improving the approximate results using Algorithm 3. Hereafter, we discuss omitting unnecessary computations to improve efficiency because the matrix size is increased. Assume that $\tilde{\Sigma}_n$ is obtained by the same computation as in Algorithm 5. Let $\hat{X}, \tilde{D} = (\tilde{d}_{ij})\in\mathbb{R}^{(m+n)\times(m+n)}$ be

$$\hat{X} := \frac{1}{\sqrt{2}}\begin{pmatrix} \hat{V} & \hat{V} & O_{n,m-n} \\ \hat{U}_1 & -\hat{U}_1 & \sqrt{2}\hat{U}_2\end{pmatrix},\quad \tilde{D} := \begin{pmatrix} \tilde{\Sigma}_n & O_{n,n} & O_{n,m-n} \\ O_{m,n} & -\tilde{\Sigma} & O_{m,m-n}\end{pmatrix}.$$

Then, for $B\in\mathbb{R}^{(m+n)\times(m+n)}$ in (14),

$$B\hat{X} = \frac{1}{\sqrt{2}}\begin{pmatrix} A^T\hat{U}_1 & -A^T\hat{U}_1 & \sqrt{2}A^T\hat{U}_2 \\ A\hat{V} & A\hat{V} & O_{m,m-n}\end{pmatrix},$$

$$\hat{X}\tilde{D} = \frac{1}{\sqrt{2}}\begin{pmatrix} \hat{V}\tilde{\Sigma}_n & -\hat{V}\tilde{\Sigma}_n & O_{n,m-n} \\ \hat{U}_1\tilde{\Sigma}_n & \hat{U}_1\tilde{\Sigma}_n & O_{m,m-n}\end{pmatrix}$$

and

$$B\hat{X} - \hat{X}\tilde{D} = \frac{1}{\sqrt{2}}\begin{pmatrix} P_1 & -P_1 & \sqrt{2}A^T\hat{U}_2 \\ P_2 & P_2 & O_{m,m-n} \end{pmatrix},$$

where

$$\begin{cases} P_1 := A^T\hat{U}_1 - \hat{V}\tilde{\Sigma}_n, \\ P_2 := A\hat{V} - \hat{U}_1\tilde{\Sigma}_n \end{cases}$$

are satisfied. Therefore,

$$H = (h_{ij}) := \hat{X}^T(B\hat{X} - \hat{X}\tilde{D}) = \frac{1}{2}\begin{pmatrix} Q_1 & -Q_2 & \sqrt{2}\hat{V}^T A^T\hat{U}_2 \\ Q_2 & -Q_1 & \sqrt{2}\hat{V}^T A^T\hat{U}_2 \\ \sqrt{2}\hat{U}_2^T P_2 & \sqrt{2}\hat{U}_2^T P_2 & O_{m-n,m-n} \end{pmatrix},$$

where

$$\begin{cases} Q_1 := \hat{V}^T P_1 + \hat{U}_1^T P_2, \\ Q_2 := \hat{V}^T P_1 - \hat{U}_1^T P_2 \end{cases}$$

holds. From the 8th line of Algorithm 3, we can write the approximate error matrix $\tilde{E} = (\tilde{e}_{ij})$ for $\hat{X}$ as

$$\tilde{e}_{ij} := \begin{cases} \frac{r_{ij}}{2} & (i = j \text{ or } i, j > 2n) \\ \frac{h_{ij}}{\tilde{d}_{jj} - \tilde{d}_{ii}} & (\text{otherwise}) \end{cases}$$

for $R = (r_{ij}) = I_{m+n} - \hat{X}^T\hat{X}$. Now, let $\tilde{E}_1, \tilde{E}_2 \in \mathbb{R}^{n\times n}$, $\tilde{E}_3 \in \mathbb{R}^{n\times(m-n)}$, $\tilde{E}_4 \in \mathbb{R}^{(m-n)\times n}$, and $\tilde{E}_5 \in \mathbb{R}^{(m-n)\times(m-n)}$ such that

$$\tilde{E} = \begin{pmatrix} \tilde{E}_1 & \tilde{E}_2 & -\tilde{E}_3 \\ \tilde{E}_2 & \tilde{E}_1 & \tilde{E}_3 \\ \tilde{E}_4 & -\tilde{E}_4 & \tilde{E}_5 \end{pmatrix}.$$

Then, it holds that

$$\hat{X}\tilde{E} = \frac{1}{\sqrt{2}}\begin{pmatrix} \tilde{G} & \tilde{G} & O_{n,m-n} \\ \tilde{F}_1 & -\tilde{F}_1 & \sqrt{2}\tilde{F}_2 \end{pmatrix},$$

where

$$\begin{cases} \tilde{G} := \hat{V}(\tilde{E}_1 + \tilde{E}_2), \\ \tilde{F}_1 := \hat{U}_1(\tilde{E}_1 - \tilde{E}_2) + \sqrt{2}\hat{U}_2\tilde{E}_4, \\ \tilde{F}_2 := \hat{U}_2\tilde{E}_5 - \sqrt{2}\hat{U}_1\tilde{E}_3. \end{cases}$$

Thus, we can update $\hat{V}$, $\hat{U}_1$, and $\hat{U}_2$ as

$$\tilde{V} \leftarrow \hat{V} + \tilde{G}, \quad \tilde{U}_1 \leftarrow \tilde{U}_1 + \tilde{F}_1, \quad \tilde{U}_2 \leftarrow \tilde{U}_2 + \tilde{F}_2.$$

Finally, we obtain the following algorithm to improve the accuracy of the approximation of singular vectors of A.

The total cost of Algorithm 8 is $3m^3 + 2m^2n + 3mn^2 + 4n^3$ to $4m^3 + 4mn^2 + 4n^3$ operations divided into

$2n^3$:	$\hat{V}^T P_1$,
$2mn^2$:	$A\hat{V}$, $A^T\hat{U}_1$, $\hat{U}_1^T P_2$, $\hat{U}_1(\tilde{E}_1 - \tilde{E}_2)$
$2mn(m-n)$:	$P^T\hat{U}_2$, $\hat{U}_2^T P_2$, $\hat{U}_2\tilde{E}_4$, $\hat{U}_1\tilde{E}_3$
$m(m-n)^2$:	$\hat{U}_2^T\hat{U}_2$ (exploiting symmetry),
$2m(m-n)^2$:	$\hat{U}_2^T\hat{U}_2$ (no exploiting symmetry),
$2m(m-n)^2$:	$\hat{U}_2\tilde{E}_5$,
$2n^3$:	$\hat{V}(\tilde{E}_1 + \tilde{E}_2)$,

which is more expensive than Algorithm 4. The difference between the costs of Algorithms 4 and 8 is $-mn^2 - n^3$ to $2m^2n - 2mn^2$ operations. For d in (9), the required arithmetic precision for the computations of multiplications $\hat{V}^T P_1$, $\hat{U}_1^T P_2$, $\hat{U}_2^T P_2$, $\hat{U}_1(\tilde{E}_1 - \tilde{E}_2)$, $\hat{U}_2\tilde{E}_4$, $\hat{U}_2\tilde{E}_5$, $\hat{U}_1\tilde{E}_3$, and $\hat{V}(\tilde{E}_1 + \tilde{E}_2)$ is d decimal digits, while that for the other computations is $2d$ decimal digits. Therefore, the total costs of $2d$ and d decimal digit computations is $m^3 + 3mn^2$ to $2m^3 - 2m^2n + 4mn^2$ operations and $2m^3 + 2m^2n + 4n^3$ operations, respectively. The cost of $2d$ decimal digit computations in Algorithm 8 is $2m^2n - mn^2 + n^3$ to $4m^2n - 2mn^2 + 2n^3$ operations less than that of Algorithm 4. Moreover, the costs of Algorithms 5 and 8 are the same.

3.3 Comparison of the algorithms costs

Here, we compare the costs of Algorithms 4, 5, 6, 7, and 8. Let ν denote the number of iterations. Define Case 1 and 2 as the cases when the symmetry of the matrix product is considered and not, respectively. We first focus on Case 1. Table 3 shows the total cost of higher-precision computations for all algorithms, and Figs. 2 and 3 indicate their ratios to the cost of Algorithm 4. From Fig. 3, the efficiency of Algorithm 7 increases with ν. For $m/n \gtrsim 1.5$ or $\nu \leq 2$, the costs of Algorithms 6 and 8 are the lowest from Fig. 2, while for other cases, the cost of Algorithm 7 is the lowest, as shown in Fig. 3.

Next, we focus on Case 2. Table 4 shows the total costs of higher-precision computations for all algorithms, and Figs. 4 and 5 indicate the ratio to the cost of Algorithm 4. From Fig. 5, the efficiency of Algorithm 7 increases with ν. For small m/n or $\nu = 1$,

Table 3 Total cost (number of operations) of higher-precision computations for ν iterations for Case 1

algorithms	$m > n$	$m = n$
Alg. 4	$2\nu m^3 + 2\nu m^2n + 2\nu mn^2 + 2\nu n^3$	$8\nu m^3$
Alg. 5	$2\nu m^3 - 2\nu m^2n + 4\nu mn^2$	$4\nu m^3$
Alg. 6	$2\nu m^3 - 2\nu m^2n + 4\nu mn^2 + 2\nu n^3$	$6\nu m^3$
Alg. 7	$2\nu m^3 + 2m^2n + 2mn^2$	$(2\nu + 4)m^3$
Alg. 8	$2\nu m^3 - 2\nu m^2n + 4\nu mn^2$	$4\nu m^3$

Algorithm 8 Refinement of approximate singular vectors $\hat{U} \in \mathbb{R}^{m\times m}$ and $\hat{V} \in \mathbb{R}^{n\times n}$ for a real matrix $A \in \mathbb{R}^{m\times n}$. Assume $\tilde{\sigma}_i \neq \tilde{\sigma}_j$ for $i \neq j$. The total cost is $3m^3 + 2m^2n + 3mn^2 + 4n^3$ to $4m^3 + 4mn^2 + 4n^3$ operations. Bold font indicates differences from Algorithm 5.

b!

```
Require: $A \in \mathbb{R}^{m\times n}$, $\hat{U} \in \mathbb{R}^{m\times m}$
Ensure: $\tilde{U} \in \mathbb{R}^{m\times m}$, $\tilde{\Sigma} \in \mathbb{R}^{m\times n}$, $\tilde{V} \in \mathbb{R}^{n\times n}$
1: function $[\tilde{U}, \tilde{\Sigma}, \tilde{V}] \leftarrow$ RefSVD5$(A, \hat{U}, \hat{V})$
2:     $P \leftarrow (A\hat{V})_h$; $Q \leftarrow (A^T\hat{U}_1)_h$
3:     $\boldsymbol{r_{ii} \leftarrow \left(1 - \frac{\hat{u}_{(i)}^T\hat{u}_{(i)} + \hat{v}_{(i)}^T\hat{v}_{(i)}}{2}\right)_h}$ (for $1 \le i \le n$)
4:     $t_{ii} \leftarrow (\hat{u}_{(i)}^T p_{(i)})_h$ (for $1 \le i \le n$)
5:     $\boldsymbol{\tilde{\sigma}_i \leftarrow \left(\frac{t_{ii}}{1-r_{ii}}\right)_h}$ (for $1 \le i \le n$)    ▷ Approximate singular values
6:     $\tilde{\Sigma}_n \leftarrow \mathrm{diag}(\tilde{\sigma}_1, \ldots, \tilde{\sigma}_n)$; $\tilde{\Sigma} \leftarrow (\tilde{\Sigma}_n, O_{n,m-n})^T$
7:     $P_1 \leftarrow (Q - \hat{V}\tilde{\Sigma}_n)_h$; $P_2 \leftarrow (P - \hat{U}_1\tilde{\Sigma}_n)_h$
8:     $P_3 \leftarrow (\hat{V}^T P_1)_\ell$; $P_4 \leftarrow (\hat{U}_1^T P_2)_\ell$
9:     $\boldsymbol{Q_1 \leftarrow \left(\frac{1}{2}(P_3 + P_4)\right)_h}$; $\boldsymbol{Q_2 \leftarrow \left(\frac{1}{2}(P_3 - P_4)\right)_h}$    ▷ $Q_i = (q_{ij}^{(i)})$
10:    $\boldsymbol{Q_3 \leftarrow \left(\frac{1}{\sqrt{2}}P^T\hat{U}_2\right)_h}$; $\boldsymbol{Q_4 \leftarrow \left(\frac{1}{\sqrt{2}}(\hat{U}_2^T P_2)_\ell\right)_h}$
11:    $\boldsymbol{\tilde{e}_{ij}^{(1)} \leftarrow \begin{cases} \left(\frac{q_{ij}^{(1)}}{\tilde{\sigma}_j - \tilde{\sigma}_i}\right)_h & (i \neq j) \\ \left(\frac{r_{ii}}{2}\right)_h & (\text{otherwise}) \end{cases}}$ (for $1 \le i, j \le n$)    ▷ $\tilde{E}_1 = (\tilde{e}_{ij}^{(1)})$
12:    $\boldsymbol{\tilde{e}_{ij}^{(2)} \leftarrow \left(\frac{q_{ij}^{(2)}}{\tilde{\sigma}_i + \tilde{\sigma}_j}\right)_h}$ (for $1 \le i, j \le n$)    ▷ $\tilde{E}_2 = (\tilde{e}_{ij}^{(2)})$
13:    $\boldsymbol{\tilde{E}_3 \leftarrow (\tilde{\Sigma}_n^{-1}Q_3)_h}$; $\boldsymbol{\tilde{E}_4 \leftarrow (Q_4\tilde{\Sigma}_n^{-1})_h}$; $\tilde{E}_5 \leftarrow \left(\frac{1}{2}(I_{m-n} - \hat{U}_2^T\hat{U}_2)\right)_h$
14:    $\boldsymbol{\tilde{U}_1 \leftarrow (\hat{U}_1 + (\hat{U}_1(\tilde{E}_1 - \tilde{E}_2)_h)_\ell + (\sqrt{2}(\hat{U}_2\tilde{E}_4))_\ell)_h}$
15:    $\boldsymbol{\tilde{U}_2 \leftarrow (\hat{U}_2 + (\hat{U}_2\tilde{E}_5)_\ell - (\sqrt{2}(\hat{U}_1\tilde{E}_3))_\ell)_h}$
16:    $\boldsymbol{\tilde{V} \leftarrow (\hat{V} + (\hat{V}(\tilde{E}_1 + \tilde{E}_2)_h)_\ell)_h}$
17: end function
```

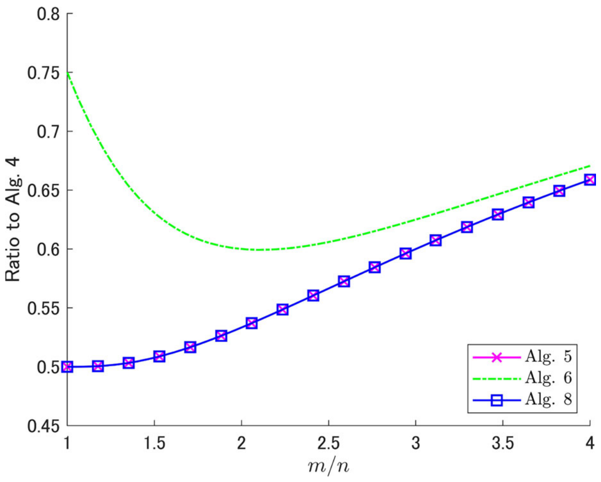

Fig. 2 Ratio of total cost of Algorithms 5, 6, and 8 to that of Algorithm 4 for any ν for Case 1

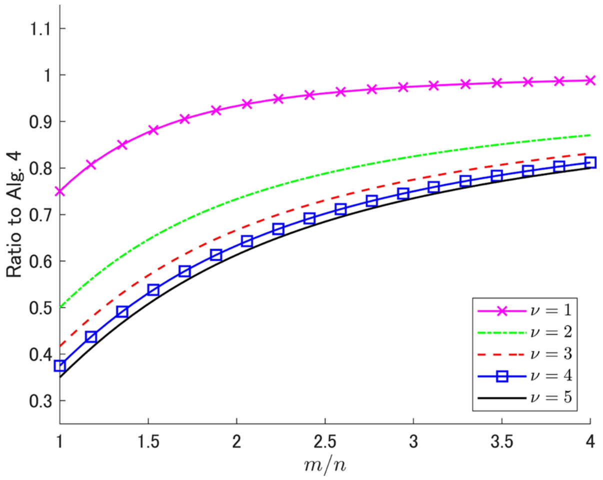

Fig. 3 Ratio of total cost of Algorithm 7 to that of Algorithm 4 for $\nu \in \{2, 4, 6, 8, 10\}$ for Case 1

Table 4 Total cost (number of operations) of higher-precision computations for ν iterations for Case 2

algorithms	$m > n$	$m = n$
Alg. 4	$\nu m^3 + 2\nu m^2 n + 2\nu mn^2 + \nu n^3$	$6\nu m^3$
Alg. 5	$\nu m^3 + 3\nu mn^2$	$4\nu m^3$
Alg. 6	$\nu m^3 + 2\nu mn^2 + \nu n^3$	$4\nu m^3$
Alg. 7	$\nu m^3 + (2\nu + 1)m^2 n + (2 - \nu)mn^2$	$(2\nu + 3)m^3$
Alg. 8	$\nu m^3 + 3\nu mn^2$	$4\nu m^3$

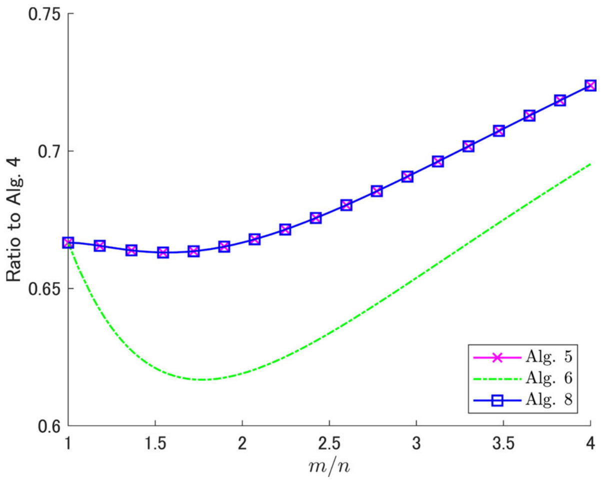

Fig. 4 Ratio of total costs of Algorithms 5, 6, and 8 to that of Algorithm 4 for any ν for Case 2

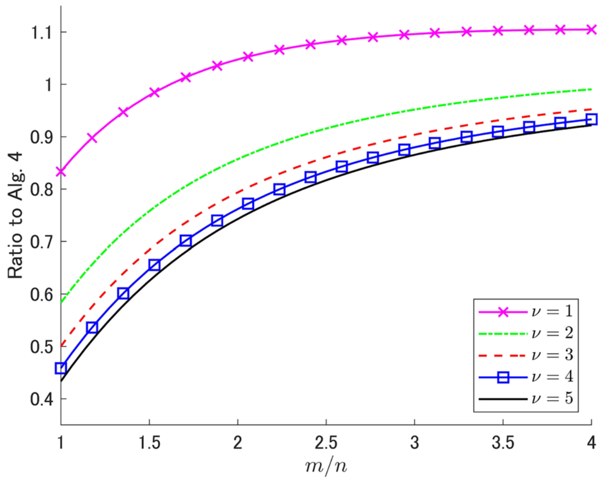

Fig. 5 Ratio of total cost of Algorithm 7 to that of Algorithm 4 for $\nu \in \{2, 4, 6, 8, 10\}$ for Case 2

the costs of Algorithms 6 and 8 are the lowest, as shown in Fig. 4, while for other cases, the cost of Algorithm 7 is the lowest, as shown in Fig. 5.

Next, we show that the performance ratio thresholds for higher- and lower-precision computations for each algorithm are lower than or comparable to Algorithm 4. We assume the performance of lower-precision arithmetic is r times faster than that of higher-precision arithmetic. Note that r means the ratio of hardware's actual measured computation speed between higher- and lower-precision computations, not the hardware peak performance ratio. The performance ratio threshold t is defined as follows:

$$t := \frac{\text{Alg. * } (r \cdot \text{(higher-precision costs)} + \text{(lower-precision costs)})}{\text{Alg. 4 } (r \cdot \text{(higher-precision costs)} + \text{(lower-precision costs)})}. \tag{16}$$

Table 5 indicates r when $t = 1$ in (16) for each algorithm for ν iterations. If the performance ratio is greater than one, the algorithm is faster than Algorithm 4. The values for Algorithms 5, 6, and 8 in the table are less than or equal to 2. For example, in Env. 1, Algorithms 5, 6, and 8 are expected to be faster than Algorithm 4 because the performance ratio of double- and quadruple-precision arithmetic is 1353 on CPU and that of single- and double-precision arithmetic is 34 on GPU from Table 1. In

Table 5 Necessary performance ratio of higher- and lower-precision computations for Algorithms 5, 6, 7, and 8 to be comparable to the speed of Algorithm 4 for ν iterations

	Alg. 5	Alg. 6	Alg. 7	Alg. 8
Case 1	$\frac{2m^2+2n^2}{2m^2-mn+n^2}$	1	$\frac{(2m^2-2n^2)\nu}{3mn\nu+n^2\nu-2mn-m^2}$	$\frac{2m^2+2n^2}{2m^2-mn+n^2}$
Case 2	$\frac{m^2+n^2}{2m^2-mn+n^2}$	$\frac{m}{2m-n}$	$\frac{(m^2-n^2)\nu}{m^2\nu+mn\nu+n^2\nu-mn-m^2}$	$\frac{m^2+n^2}{2m^2-mn+n^2}$

Env. 2, Algorithms 5, 6, and 8 are expected to be slower than or comparable to the speed of Algorithm 4 because the performance ratio of single- and double-precision arithmetic is 1. The performance ratio thresholds for Algorithm 7 depend on ν, and Algorithm 7 to be faster than Algorithm 4 for larger ν.

4 Numerical experiments

In this section, we present the results of numerical experiments showing the performance of Algorithms 4, 5, 6, 7, and 8. We generate $A \in \mathbb{F}^{m\times n}$ using the MATLAB built-in function `gallery` as

```
  A = gallery('randsvd',[m n],cnd,mode,m,n,1),
```

where `cnd` denotes the approximate condition number of A, and `mode` is a variable that specifies

1. one large and $n-1$ small singular values: $\sigma_1 \approx 1, \sigma_i \approx \mathtt{cnd}^{-1}, i = 2, \ldots, n$,
2. one small and $n-1$ large singular values: $\sigma_n \approx \mathtt{cnd}^{-1}, \sigma_i \approx 1, i = 1, \ldots, n-1$,
3. geometrically distributed singular values: $\sigma_i \approx \mathtt{cnd}^{-(i-1)/(n-1)}, i = 1, \ldots, n$,
4. arithmetically distributed singular values: $\sigma_i \approx 1 - (1 - \mathtt{cnd}^{-1})(i-1)/(n-1)$, $i = 1, \ldots, n$, and
5. random singular values with uniformly distributed logarithm: $\sigma_i \approx \mathtt{cnd}^{-r(i)}$, $r(i) \in (0, 1), i = 1, \ldots, n$.

Figure 6 shows the singular value distribution for $n = 100$ and $\mathtt{cnd} = 10^5$.

We fix `mode` $= 4$ to satisfy (7). Note that for `mode` $= 3$, (7) is satisfied if A is very well-conditioned; otherwise, clustered singular values appear and all algorithms do not work well. We regard the approximate singular values of A obtained by using the built-in function `svd` in the Advanpix Multiprecision Computing Toolbox for MATLAB with 68 decimal digits as the exact singular values.

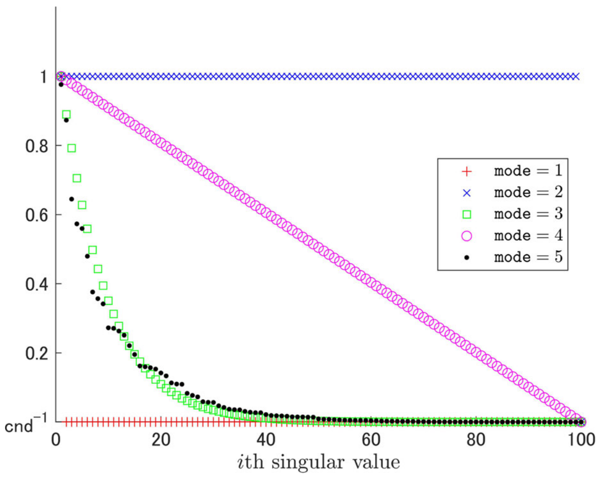

Fig. 6 Distribution of the singular values for $n = 100$ and $\mathtt{cnd} = 10^5$

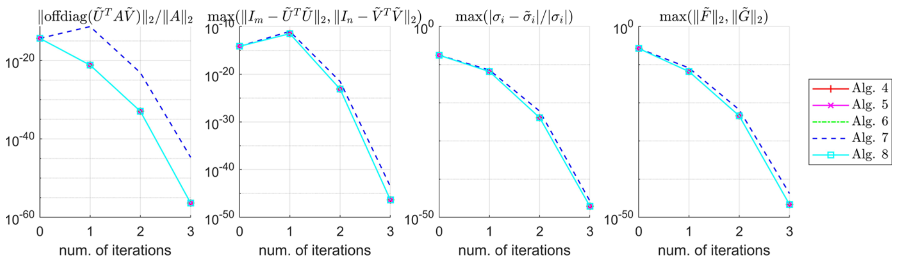

Fig. 7 Convergence of all algorithms for $m = 2n = 1000$ and $\mathtt{cnd} = 10^{10}$

4.1 Convergence of algorithms

We show the results of convergence of Algorithms 4, 5, 6, 7, and 8 computed in 340 decimal digits arithmetic in order to simulate exact arithmetic. Here, we regard the approximate singular values of A obtained by using the built-in function svd in 340 decimal digits as the exact singular values. Figures 7 and 8 show the convergence of all algorithms for $\mathtt{cnd} = 10^{10}$ and $m = 1000$. Among the results, the convergence for Algorithms 4, 5, 6, and 8 are the same, while that for Algorithm 7 is worse. The reason for this is that the condition number of AA^T is squared compared to that of A and Algorithm 7 does not consider the improvement of $\hat{V}$.

4.2 Numerical experiments on CPU

Here, we show the numerical results obtained using a CPU. The numerical experiments are run using Env. 1. Assume that the results of the approximate singular value decomposition of A obtained using the MATLAB built-in function svd with double-precision arithmetic are the initial values $\hat{U}$ and $\hat{V}$ of U and V, respectively. Additionally, all operations inside the parentheses of $(\cdot)_h$ and $(\cdot)_\ell$ are executed in double-double- and double-precision, respectively.

Tables 6 and 7 show the computing time in seconds for all algorithms for $\mathtt{cnd} = 10^2$. Note that svd in the following tables is executed as `[U,S,V]=svd(A)`. Also, the results of Algorithm 7 include the computing time for AA^T and $A^T\tilde{U}_1\tilde{\Sigma}_n^{-1}$. Figures 9 and 10 show the convergence of all algorithms for $\mathtt{cnd} = 10^2$ and $m = 5000$.

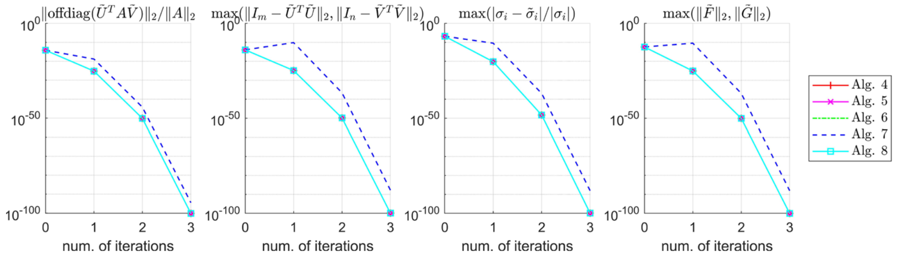

Fig. 8 Convergence of all algorithms for $m = n = 1000$ and $\mathtt{cnd} = 10^{10}$

Table 6 Cumulative computing time in seconds for all algorithms for $m = 2n$ on CPU

m		`svd(A)`	Alg. 4	Alg. 5	Alg. 6	Alg. 7	Alg. 8
1000	1st	0.05	0.79	0.66	0.71	0.79	0.69
	2nd		1.57	1.29	1.46	1.31	1.37
	3rd		2.35	1.92	2.21	1.84	2.05
5000	1st	2.80	20.0	15.2	16.7	20.2	15.6
	2nd		37.4	27.7	30.6	31.1	28.4
	3rd		54.6	40.3	44.4	41.9	41.1
10000	1st	18.4	110	79.6	86.1	107.5	80.7
	2nd		201	140	153	162	143
	3rd		293	202	221	217	205

Note that offdiag$(\tilde{U}^T A \tilde{V})$ denotes the off-diagonal part of $\tilde{U}^T A \tilde{V}$, that is, the diagonal elements are set to zero. The results indicate that Algorithms 5 and 8 are comparable. More specifically, they are 1.5 and 1.7 times faster than Algorithm 4 for $m = 2n$ and $m = n$ per iteration, respectively. Moreover, Algorithm 7 is the fastest; however, the convergence is a little worse than the others. Figures 11 and 12 show the convergence for all algorithms for `cnd` $= 10^6$ and $m = 5000$. Among the results, the convergence for Algorithms 4, 5, 6, and 8 are similar for the case of `cnd` $= 10^2$, while that for Algorithm 7 is worse. The reason for this is that the condition number of AA^T is squared compared to that of A. Figures 13 and 14 show the convergence for all algorithms for `cnd` $= 10^{10}$ and $m = 5000$. From Figs. 9, 10, 11, 12, 13, and 14, we can see the tendency of bounds on the limiting accuracy for each `cnd`.

4.3 Numerical experiments on GPU

Next, we show the numerical results obtained using a GPU. The numerical experiments are run using the environments Env. 1 and 2. Let $B \in \mathbb{F}^{m\times n}$ be the conversion of A

Table 7 Cumulative computing time in seconds for all algorithms for $m = n$ on CPU

m		`svd(A)`	Alg. 4	Alg. 5	Alg. 6	Alg. 7	Alg. 8
1000	1st	0.14	1.22	0.84	1.01	0.95	0.86
	2nd		2.46	1.59	2.02	1.32	1.62
	3rd		3.70	2.34	3.03	1.68	2.38
5000	1st	10.7	43.2	30.2	36.8	33.2	30.2
	2nd		75.8	49.5	63.0	42.8	49.8
	3rd		108	68.9	89.3	52.2	69.4
10000	1st	72.6	252	178	216	201	179
	2nd		430	283	359	253	285
	3rd		609	388	502	304	392

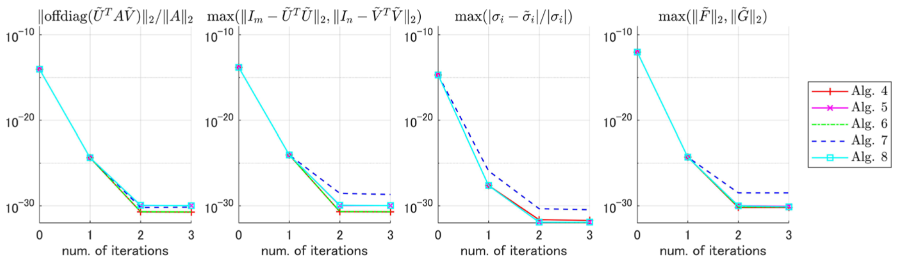

Fig. 9 Convergence of all algorithms for $m = 2n = 5000$ and $\texttt{cnd} = 10^2$ on CPU

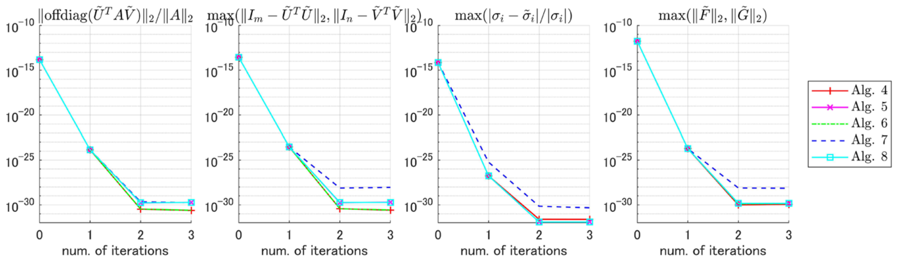

Fig. 10 Convergence of all algorithms for $m = n = 5000$ and $\texttt{cnd} = 10^2$ on CPU

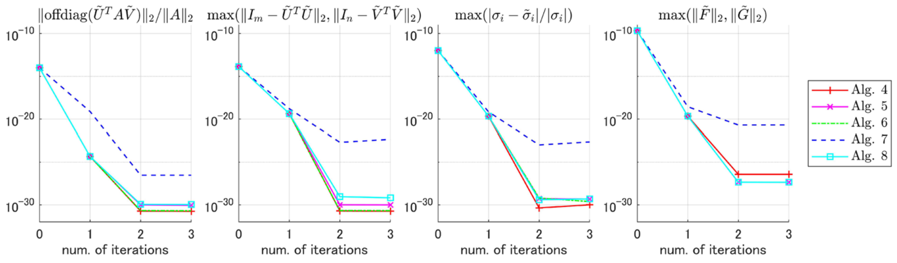

Fig. 11 Convergence of all algorithms for $m = 2n = 5000$ and $\texttt{cnd} = 10^6$ on CPU

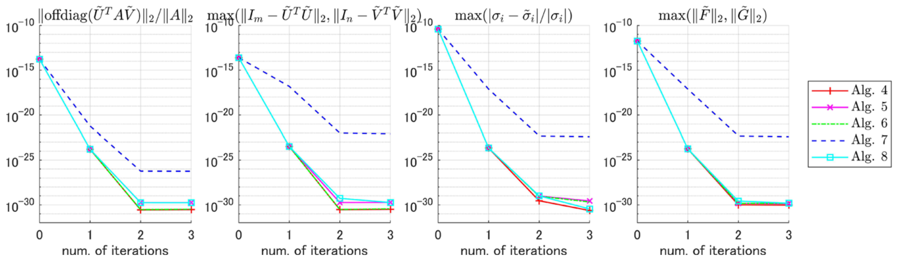

Fig. 12 Convergence of all algorithms for $m = n = 5000$ and $\texttt{cnd} = 10^6$ on CPU

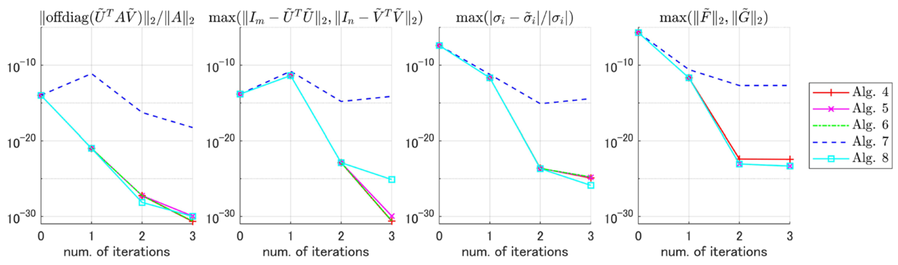

Fig. 13 Convergence of all algorithms for $m = 2n = 5000$ and $\mathtt{cnd} = 10^{10}$ on CPU

to a single-precision floating-point matrix as `B=single(A)`, where `single` is the built-in function in MATLAB. Assume that the results of the approximate singular value decomposition of B obtained by using the MATLAB built-in function `svd` with single-precision arithmetic are the initial values $\hat{U}$ and $\hat{V}$ of U and V, respectively. Here, all operations inside the parentheses of $(\cdot)_h$ and $(\cdot)_\ell$ are executed in double- and single-precision, respectively.

Tables 8 and 9 show the computing time in seconds for all algorithms for $\mathtt{cnd} = 10^2$ using Env. 1. Figures 15 and 16 show the convergence for all algorithms for $\mathtt{cnd} = 10^2$ and $m = 10000$. The results show that Algorithm 5 and 8 are comparable. More specifically, these algorithms are 1.2 times faster than Algorithm 4 for $m = 2n$ and $m = n$ per iteration. Moreover, Algorithm 7 is the fastest; however, the convergence is worse than the others. Figures 17 and 18 show the convergence of all algorithms for $\mathtt{cnd} = 10^4$ and $m = 10000$. Among these, the convergence of Algorithms 4, 5, 6, and 8 are similar for the case of $\mathtt{cnd} = 10^2$, while that of Algorithm 7 is worse. This result is due to the condition number of AA^T. For $\mathtt{cnd} = 10^2$, the approximate singular values obtained by using two iterations of Algorithms 5 and 8 are as accurate as or more accurate than those obtained by using `svd(A)`. Also, the computing time for two iterations using Algorithms 5 or 8 is faster than that of `svd(A)`. Thus, for a well-conditioned matrix, iterative refinement with Algorithms 5 and 8 for the results of `svd(B)` is superior to `svd(A)` in terms of computation speed and accuracy of the approximate singular values.

Tables 10 and 11 show the computing time in seconds for all algorithms for $\mathtt{cnd} = 10^2$ using Env. 2. Using Env. 1 and 2, the convergence for all algorithms is almost the same. In the results, using Env. 2, all algorithms are much faster than the functions for

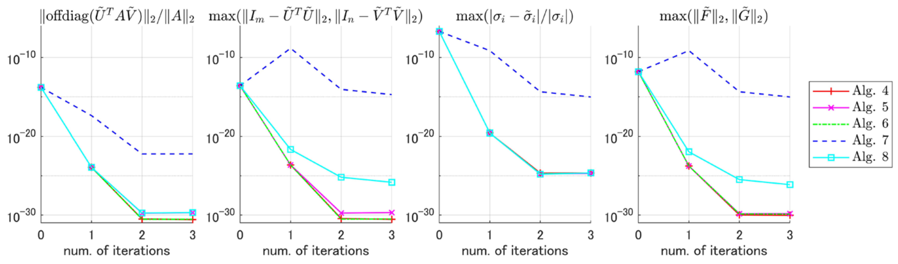

Fig. 14 Convergence of all algorithms for $m = n = 5000$ and $\mathtt{cnd} = 10^{10}$ on CPU

Table 8 Cumulative computing time in seconds for all algorithms for $m = 2n$ using Env. 1

m		`svd(A)`	`svd(B)`	Alg. 4	Alg. 5	Alg. 6	Alg. 7	Alg. 8
5000	1st	2.87	2.01	2.68	2.48	2.47	2.72	2.51
	2nd			3.35	2.95	2.93	3.18	3.01
	3rd			4.02	3.42	3.39	3.64	3.51
10000	1st	14.4	7.78	12.9	11.4	11.2	13.3	11.5
	2nd			18.1	15.1	14.6	16.9	15.3
	3rd			23.2	18.7	18.0	20.5	19.1

Table 9 Cumulative computing time in seconds for all algorithms for $m = n$ using Env. 1

m		`svd(A)`	`svd(B)`	Alg. 4	Alg. 5	Alg. 6	Alg. 7	Alg. 8
5000	1st	8.99	6.61	8.10	7.65	7.66	7.85	7.65
	2nd			9.60	8.69	8.70	8.37	8.69
	3rd			11.1	9.73	9.75	8.89	9.73
10000	1st	47.0	29.3	40.9	37.9	37.6	39.2	37.9
	2nd			52.5	46.4	45.9	43.5	46.5
	3rd			64.2	55.0	54.1	47.8	55.1

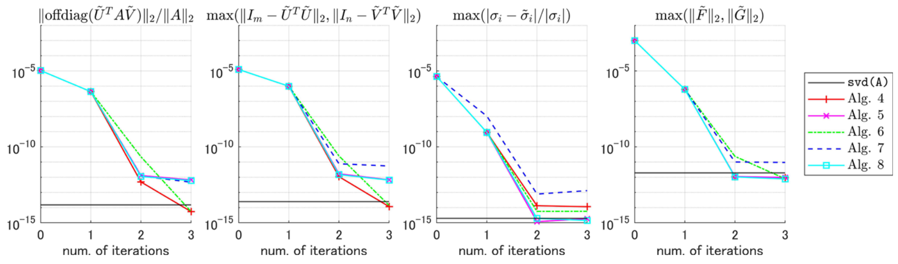

Fig. 15 Convergence of all algorithms for $m = 2n = 10000$ and $\texttt{cnd} = 10^2$ using Env. 1

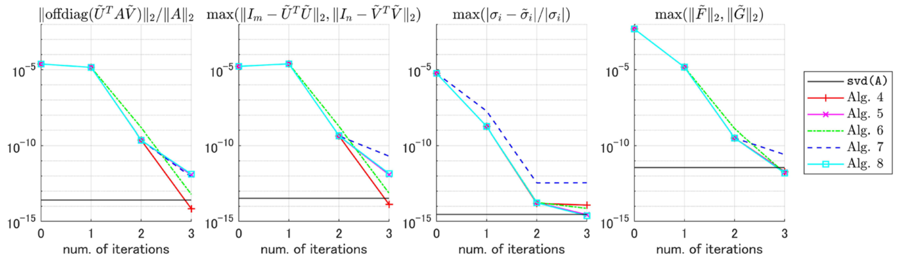

Fig. 16 Convergence of all algorithms for $m = n = 10000$ and $\texttt{cnd} = 10^2$ using Env. 1

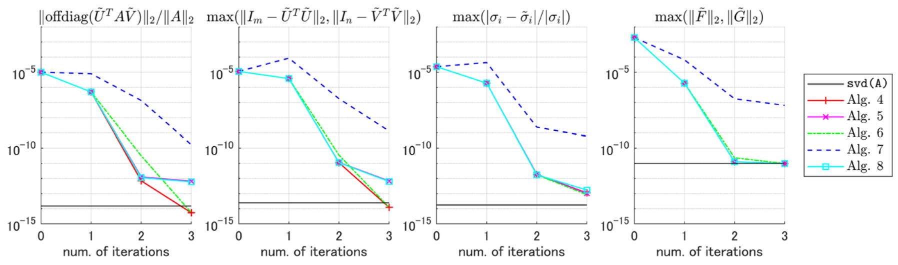

Fig. 17 Convergence of all algorithms for $m = 2n = 10000$ and $\texttt{cnd} = 10^4$ using Env. 1

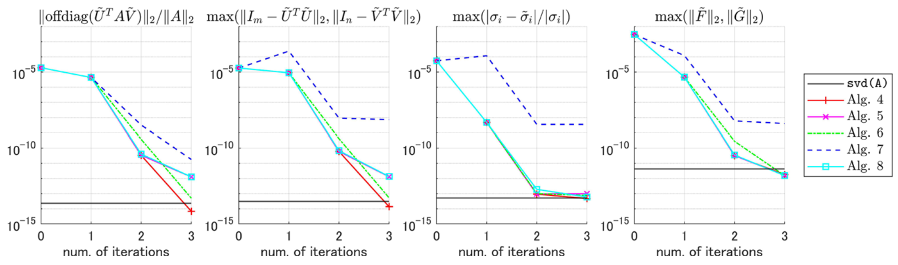

Fig. 18 Convergence of all algorithms for $m = n = 10000$ and $\texttt{cnd} = 10^4$ using Env. 1

Table 10 Cumulative computing time in seconds and ratio for all algorithms for $m = 2n$ using Env. 2

m		`svd(A)`	`svd(B)`	Alg. 4	Alg. 5	Alg. 6	Alg. 7	Alg. 8
5000	1st	1.56	1.10	1.15	1.15	1.15	1.15	1.16
	2nd			1.19	1.19	1.19	1.19	1.21
	3rd			1.23	1.23	1.23	1.23	1.26
10000	1st	6.94	4.56	4.85	4.86	4.85	4.89	4.90
	2nd			5.13	5.16	5.13	5.16	5.23
	3rd			5.41	5.46	5.42	5.43	5.56

Table 11 Cumulative computing time in seconds and ratio for all algorithms for $m = n$ using Env. 2

m		`svd(A)`	`svd(B)`	Alg. 4	Alg. 5	Alg. 6	Alg. 7	Alg. 8
5000	1st	4.81	3.30	3.38	3.39	3.38	3.37	3.39
	2nd			3.46	3.47	3.46	3.41	3.48
	3rd			3.54	3.56	3.54	3.46	3.57
10000	1st	23.9	15.3	15.9	16.1	15.9	15.9	16.1
	2nd			16.5	16.8	16.5	16.2	16.8
	3rd			17.0	17.5	17.1	16.6	17.6

singular value decomposition. In particular, Algorithm 7 is the fastest but has the worst convergence. For a well-conditioned matrix, iterative refinement with Algorithms 4, 5, 6 and 8 of the results of `svd(B)` is superior to the same method applied to those of `svd(A)` in terms of computation speed and accuracy of the approximate singular values.

5 Conclusion

In this paper, we showed that the Ogita-Aishima algorithm can be executed with highly accurate matrix multiplications carried out five times per iteration. Moreover, we proposed four iterative refinement algorithms for singular value decomposition constructed with highly accurate matrix multiplications carried out either four or five times per iteration. In an environment where lower-precision arithmetic is much faster than higher-precision arithmetic, the proposed algorithms are faster than the Ogita-Aishima algorithm. However, in an environment where the performance of lower-precision arithmetic is comparable to that of higher-precision arithmetic, the computing time for all algorithms is comparable.

All iterative refinement algorithms introduced in this paper do not work when multiple or clustered singular values are present. In the future, we will consider methods to overcome this problem. We also need to analyze the bounds on the limiting accuracy based on the precisions used and the convergence conditions.

Acknowledgements We appreciate the reviewer for your insightful comments on our paper.

Author contribution All authors contributed equally to this work.

Funding This work was supported by a Grant-in-Aid for JSPS Fellows No. 22J20869 and a Grant-in-Aid for Scientific Research (B) No. 20H04195 from the Japan Society for the Promotion of Science.

Data Availability Not applicable

Declarations

Ethical approval and consent to participate Not applicable

Consent for publication Not applicable

Human and animal ethics Not applicable

Conflict of interest The authors declare no competing interests.

References

1. LAPACK (Linear Algebra PACKage) (2022). http://www.netlib.org/lapack/
2. Wedin, P.Å.: Perturbation bounds in connection with singular value decomposition. BIT Numerical Mathematics **12**(1), 99–111 (1972)
3. Davis, C., Kahan, W.M.: The rotation of eigenvectors by a perturbation. III. SIAM J. Numer. Anal. **7**(1), 1–46 (1970)
4. Wedin, P.Å.: On angles between subspaces of a finite dimensional inner product space. In: Matrix Pencils, pp. 263–285. Springer, Berlin, Heidelberg (1983)
5. Dopico, F.M.: A note on sin Θ theorems for singular subspace variations. BIT Numerical Mathematics **40**(2), 395–403 (2000)
6. Davies, P.I., Smith, M.I.: Updating the singular value decomposition. J. Comput. Appl. Math. **170**(1), 145–167 (2004)
7. Ogita, T., Aishima, K.: Iterative refinement for singular value decomposition based on matrix multiplication. J. Comput. Appl. Math **369**, 112512 (2020)
8. Hida, Y., Li, X.S., Bailey, D.H.: Quad-double arithmetic: Algorithms, implementation, and application. In: 15th IEEE Symposium on Computer Arithmetic, pp. 155–162 (2000). Citeseer
9. Hida, Y., Li, X.S., Bailey, D.H.: Algorithms for quad-double precision floating point arithmetic. In: Proceedings 15th IEEE Symposium on Computer Arithmetic. ARITH-15 2001, pp. 155–162 (2001). IEEE
10. Advanpix LLC.: Multiprecision Computing Toolbox for MATLAB. (2023). http://www.advanpix.com/
11. Wilkinson, J.H.: Rounding errors in algebraic processes. Soc. Ind. Appl. Math. (2023)
12. Moler, C.B.: Iterative refinement in floating point. J. ACM (JACM) **14**(2), 316–321 (1967)
13. Jankowski, M., Woźniakowski, H.: Iterative refinement implies numerical stability. BIT Numer. Math. **17**, 303–311 (1977)
14. Skeel, R.D.: Iterative refinement implies numerical stability for Gaussian elimination. Math. Comput. **35**(151), 817–832 (1980)
15. Higham, N.J.: Iterative refinement enhances the stability of QR factorization methods for solving linear equations. BIT Numer. Math. **31**, 447–468 (1991)
16. Higham, N.J.: Iterative refinement for linear systems and LAPACK. IMA J. Numer. Anal. **17**(4), 495–509 (1997)
17. Higham, N.J.: Accuracy and stability of numerical algorithms. Soc. Ind. Appl. Math. (2002)
18. Langou, J., Langou, J., Luszczek, P., Kurzak, J. Buttari, A., Dongarra, J.: Exploiting the performance of 32 bit floating point arithmetic in obtaining 64 bit accuracy (revisiting iterative refinement for linear systems). In: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing (2006)
19. Carson, E., Higham, N.J.: A new analysis of iterative refinement and its application to accurate solution of ill-conditioned sparse linear systems. SIAM J. Sci. Comput. **39**(6), A2834–A2856 (2017)
20. Carson, E., Higham, N.J.: Accelerating the solution of linear systems by iterative refinement in three precisions. SIAM J. Sci. Comput. **40**(2), 817–847 (2018)
21. Higham, N.J.: Error analysis for standard and GMRES-based iterative refinement in two and three-precisions (2019)
22. Haidar, A., Bayraktar, H., Tomov, S., Dongarra, J., Higham, N.J.: Mixed-precision iterative refinement using tensor cores on GPUs to accelerate solution of linear systems. Proceedings of the Royal Society A. **476**(2243), 20200110 (2020)
23. Ogita, T., Aishima, K.: Iterative refinement for symmetric eigenvalue decomposition. Japan Journal of Industrial and Applied Mathematics. **35**(3), 1007–1035 (2018)
24. Uchino, Y., Ozaki, K., Ogita, T.: Acceleration of iterative refinement for symmetric eigenvalue decomposition (in Japanese). IPSJ Transactions on Advanced Computing System, in press **15**(1), 1–12 (2022)
25. Shiroma, K., Kudo, S., Yamamoto, Y.: An eigendecomposition tracking method for real symmetric matrices based on Ogita-Aishima's eigenvector refinement algorithm (in Japanese). Trans. JSIAM **29**(1), 78–120 (2019)
26. Ozaki, K., Ogita, T., Oishi, S., Rump, S.M.: Error-free transformations of matrix multiplication by using fast routines of matrix multiplication and its applications. Numerical Algorithms **59**(1), 95–118 (2012)
27. Nakata, M.: MPLAPACK version 1.0.0 user manual (2021)

28. Li, X.S., Demmel, J.W., Bailey, D.H., Henry, G., Hida, Y., Iskandar, J., Kahan, W., Kang, S.Y., Kapur, A., Martin, M.C., et al.: Design, implementation and testing of extended and mixed precision BLAS. ACM Transactions on Mathematical Software **28**(2), 152–205 (2002)
29. IEEE standard for floating-point arithmetic: IEEE Std **754–2008**, 1–70 (2008)
30. Golub, G.H., Kahan, W.M.: Calculating the singular values and pseudo-inverse of a matrix. Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis **2**(2), 205–224 (1965)
31. Golub, G.H., Van Loan, C.F.: Matrix Computations, 4th edn. JHU press, Baltimore (2013)