JSIAM Letters Vol.2 (2010) pp.25–28 ©2010 Japan Society for Industrial and Applied Mathematics

# Mixed double-multiple precision version of hyperplane constrained method for singular value decomposition

Kenichi Yadani[1], Koichi Kondo[2] and Masashi Iwasaki[3]

[1] Graduate School of Informatics, Kyoto University, Yoshida-Hommachi, Sakyo-ku, Kyoto 606-8501, Japan
[2] Faculty of Science and Engineering, Doshisha University, 1-3, Tatara Miyakodani, Kyotanabe City, Kyoto 610-0394, Japan
[3] Department of Informatics and Environmental Science, Kyoto Prefectural University, 1-5, Nagaragi-cho, Shimogamo, Sakyo-ku, Kyoto 606-8522, Japan

E-mail *yadani@amp.i.kyoto-u.ac.jp*

Received December 19, 2009, Accepted January 24, 2010

**Abstract**
In this paper, we design a mixed double-multiple precision version of the hyperplane constrained method for singular value decomposition (SVD), which is based on solving nonlinear systems with the solutions constrained on hyperplanes. We also propose its hybrid method in order to shorten the running time. Through some numerical examples for matrices with small singular values, it is shown that, by new versions, the SVD is computable with high relative accuracy.

**Keywords** mixed precision, singular value decomposition, hyperplane constrained method

**Research Activity Group** Algorithms for Matrix / Eigenvalue Problems and their Applications

## 1. Introduction

The singular value decomposition (SVD) of a rectangular matrix $A \in \boldsymbol{R}^{m\times n}$ with $m \le n$ is defined as

$$A = U\Sigma V^T, \quad \Sigma = (\mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_m)\, O_{m,n-m}),$$
$$U = (\boldsymbol{u}_1\, \boldsymbol{u}_2 \cdots \boldsymbol{u}_m), \quad V = (\boldsymbol{v}_1\, \boldsymbol{v}_2 \cdots \boldsymbol{v}_n),$$

where $\sigma_k$ are positive, and $U, V$ are orthogonal square matrices. We call $\sigma_k, \boldsymbol{u}_k$ and $\boldsymbol{v}_k$ the singular value, the left singular vector and the right singular vector, respectively. The pairs $(\sigma_k, \boldsymbol{u}_k, \boldsymbol{v}_k)$ for $k = 1, 2, \ldots, m$ are called singular pairs for simplicity.

In [1], a numerical SVD method named the hyperplane constrained method is proposed. This method generates the SVD of given matrix by solving the nonlinear systems whose solutions are constrained on hyperplanes. In [2,3], we show the convergence of the hyperplane constrained method. In [4], we demonstrate that the computed SVD is highly accurate in terms of residual error. We also propose the hybrid method of the hyperplane constrained method and other fast SVD method in order to shorten the running time. The purpose of this paper is to propose the mixed double-multiple precision versions of the hyperplane constrained method and its hybrid method for getting more accurate SVD of matrix, which is given with double precision format.

In Section 2, we first explain the original version of the hyperplane constrained method. By numerical examples, it is shown that the computed singular values by the original version are highly accurate in terms of absolute error. In double precision arithmetic, however, it is expected that the computed singular values are not always with high relative accuracy. In Section 3, we design a mixed precision version of the hyperplane constrained method. Some numerical examples are also shown. In Section 4, we propose a mixed precision version of the hybrid method, which is combining the hyperplane constrained method with other fast SVD method. Through numerical examples, it is observed that, in mixed precision arithmetic, the hybrid method runs faster than the hyperplane constrained method with keeping the accuracy. In Section 5, we conclude this paper.

## 2. Hyperplane constrained method

We start from introducing the original hyperplane constrained method for SVD proposed in [1, 2, 4].

The original version employs the Newton type iteration for solving the nonlinear system with arbitrary parameters $C \in \boldsymbol{R}\backslash\{0\}$ and $\boldsymbol{z} \in \boldsymbol{R}^m$,

$$\boldsymbol{H}(\boldsymbol{u}, \boldsymbol{v}) = \boldsymbol{0}, \tag{1}$$

where

$$\boldsymbol{H}(\boldsymbol{u}, \boldsymbol{v}) = \begin{pmatrix} A\boldsymbol{v} - \sigma(\boldsymbol{v})\boldsymbol{u} \\ A^T\boldsymbol{u} - \sigma(\boldsymbol{v})\boldsymbol{v} \end{pmatrix},$$

$$\sigma(\boldsymbol{v}) = \frac{\boldsymbol{w}^T\boldsymbol{v}}{C}, \quad \boldsymbol{w} = A^T\boldsymbol{z}. \tag{2}$$

Here, the solution of (1) becomes $(\boldsymbol{u}, \boldsymbol{v}) = (\alpha_k\boldsymbol{u}_k, \alpha_k\boldsymbol{v}_k)$ with $\alpha_k = C/(\boldsymbol{z}, \boldsymbol{u}_k)$. And then $\sigma(\alpha_k\boldsymbol{v}_k)$ becomes the singular value of $A$. Note that $\alpha_k\boldsymbol{u}_k$ lies on the hyperplane $\Gamma = \{\boldsymbol{u} \,|\, (\boldsymbol{z}, \boldsymbol{u}) = C\}$. One of the singular pairs is then given by the following Newton type iteration.

```
1. function $(\sigma, \boldsymbol{u}, \boldsymbol{v}, \ell) = \mathtt{hppair}(A, \boldsymbol{z}, \tilde{\boldsymbol{u}}, \tilde{\boldsymbol{v}}, C, \ell_{\max})$
2. $\boldsymbol{w} := A^T\boldsymbol{z}$
3. $\alpha := C/(\boldsymbol{z}^T\tilde{\boldsymbol{u}}); \boldsymbol{u} := \alpha\tilde{\boldsymbol{u}}; \boldsymbol{v} := \alpha\tilde{\boldsymbol{v}}$
```

```
 4. $\sigma := \boldsymbol{w}^T\boldsymbol{v}/C$
 5. $\ell := 0$
 6. while ($\ell \le \ell_{\max}$) do
 7.    $\boldsymbol{H} := \begin{pmatrix} A\boldsymbol{v}-\sigma\boldsymbol{u} \\ A^T\boldsymbol{u}-\sigma\boldsymbol{v} \end{pmatrix}$
 8.    $J := \begin{pmatrix} -\sigma I_m & A-\boldsymbol{u}\boldsymbol{w}^TC^{-1} \\ A^T & -\sigma I_n-\boldsymbol{v}\boldsymbol{w}^TC^{-1} \end{pmatrix}$
 9.    $\boldsymbol{e} := J^{-1}\boldsymbol{H}$
10.    $\begin{pmatrix} \boldsymbol{u} \\ \boldsymbol{v} \end{pmatrix} := \begin{pmatrix} \boldsymbol{u} \\ \boldsymbol{v} \end{pmatrix} - \boldsymbol{e}$
11.    $\alpha := C/(\boldsymbol{z}^T\boldsymbol{u}); \boldsymbol{u} := \alpha\boldsymbol{u}; \boldsymbol{v} := \alpha\boldsymbol{v}$
12.    $\sigma := \boldsymbol{w}^T\boldsymbol{v}/C$
13.    if ($\|\boldsymbol{e}\|_\infty < 2^{-46}\|\boldsymbol{u}\|_\infty$) then break; end if
14.    $\ell := \ell + 1$
15. end while
16. $\boldsymbol{u} := \boldsymbol{u}/\|\boldsymbol{u}\|_2; \boldsymbol{v} := \boldsymbol{v}/\|\boldsymbol{v}\|_2; \sigma := \boldsymbol{u}^T(A\boldsymbol{v})$
17. if ($\sigma < 0$) then $\sigma := -\sigma; \boldsymbol{v} := -\boldsymbol{v}$; end if
```

Moreover, using hppair with suitable $\boldsymbol{z}$, all singular pairs are computed as follows.

```
 1. function $(\Sigma, U, V) =$ hpsvd($A$)
 2. $C := 1$; $\ell_{\max} := 30$
 3. for $k = 1, 2, \ldots, m$ do
 4.    $\boldsymbol{z} :=$ select($\mathcal{U}_{k-1}^\perp$)
 5.    $\tilde{\boldsymbol{u}} :=$ select($\mathcal{U}_{k-1}^\perp$); $\tilde{\boldsymbol{v}} :=$ select($\mathcal{V}_{k-1}^\perp$)
 6.    $(\sigma_k, \boldsymbol{u}_k, \boldsymbol{v}_k, \ell) :=$ hppair($A, \boldsymbol{z}, \tilde{\boldsymbol{u}}, \tilde{\boldsymbol{v}}, C, \ell_{\max}$)
 7.    if ($\ell > \ell_{\max}$) then
 8.       $\boldsymbol{u}_k :=$gs($\boldsymbol{u}_k, \mathcal{U}_{k-1}$); $\boldsymbol{v}_k :=$gs($\boldsymbol{v}_k, \mathcal{V}_{k-1}$)
 9.       $\sigma_k := \boldsymbol{u}_k^T(A\boldsymbol{v}_k)$
10.    end if
11. end for
12. $(\boldsymbol{v}_{m+1}, \boldsymbol{v}_{m+2}, \ldots, \boldsymbol{v}_n) :=$ kernel($\mathcal{V}_m$)
13. $\Sigma := (\mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_m)\, O_{m,n-m})$
14. $U := (\boldsymbol{u}_1\, \boldsymbol{u}_2 \cdots \boldsymbol{u}_m); V := (\boldsymbol{v}_1\, \boldsymbol{v}_2 \cdots \boldsymbol{v}_n)$
```

Let $\mathcal{U}_k = \mathrm{span}(\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_k)$, $\mathcal{V}_k = \mathrm{span}(\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k)$. The functions select($\mathcal{U}_{k-1}^\perp$) and select($\mathcal{V}_{k-1}^\perp$) generate the vectors, which are randomly selected from the orthogonal complements of $\mathcal{U}_{k-1}$ and $\mathcal{V}_{k-1}$ with the help of the Gram-Schmidt process, respectively. The functions gs($\boldsymbol{u}_k, \mathcal{U}_{k-1}$) and gs($\boldsymbol{v}_k, \mathcal{V}_{k-1}$) orthonormalize $\boldsymbol{u}_k$ and $\boldsymbol{v}_k$ to $\{\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_{k-1}\}$ and $\{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_{k-1}\}$ by the Gram-Schmidt process, respectively. The function kernel($\mathcal{V}_m$) returns the orthonormal basis of the orthogonal complement of $\mathcal{V}_m$. It is proved in [2] that the iteration from the 6th line to the 15th line in hppair has quadratic convergence and the SVD of $A$ is computable by hpsvd.

We next show some numerical examples for hpsvd. We here prepare the test matrices $A_1$ and $A_2$ with double precision as $A_i = U(S_i\, O_{m,n-m})V^T$, where

$$S_1 = \mathrm{diag}\left(1, \varepsilon + \frac{(m-2)(1-\varepsilon)}{m-1}, \ldots, \varepsilon + \frac{1-\varepsilon}{m-1}, \varepsilon\right),$$

$$S_2 = \mathrm{diag}(1, \varepsilon^{1/(m-1)}, \ldots, \varepsilon^{(m-2)/(m-1)}, \varepsilon),$$

and the orthogonal matrices $U \in \boldsymbol{R}^{m\times m}, V \in \boldsymbol{R}^{n\times n}$ are randomly given. Let us set $m = 50, n = 100$ and $\varepsilon = 10^{-13}$. Then the condition number, namely, the ratio of the maximal singular value to the minimal one, for $A_i$ becomes $10^{13}$. With respect to the numerical error, we compare our hpsvd and the routine dgesvd in LAPACK [5]. Let $E_{\boldsymbol{H}}$ be the residual error defined as

$$E_{\boldsymbol{H}} = \left\| \begin{pmatrix} A\boldsymbol{v} - \sigma\boldsymbol{u} \\ A^T\boldsymbol{u} - \sigma\boldsymbol{v} \end{pmatrix} \right\|_\infty .$$

Moreover, let $E_\sigma$ and $E_\sigma^{\mathrm{r}}$ be the absolute error and the relative error of computed singular value $\sigma$, namely, $E_\sigma = |\sigma - \sigma^*|, E_\sigma^{\mathrm{r}} = |\sigma - \sigma^*|/\sigma^*$, where $\sigma^*$ denotes the singular value with double precision format given by casting the result from performing the hyperplane constrained method in 224-bit precision arithmetic. We use the multiple precision arithmetic library GMP 4.2.2 [6]. Numerical experiments are carried out on our computer with CPU: Intel Core i7 3.20GHz, memory: 3GB, OS: Linux kernel 2.6.26, compiler: gcc 4.3.2 and LAPACK 3.2.1.

Figs. 1-(a) and 2-(a) show the residual errors $E_{\boldsymbol{H}}$. In dgesvd, the largest $E_{\boldsymbol{H}}$ for $A_1$ and $A_2$ become $2^{-49}$. While, in hpsvd, all $E_{\boldsymbol{H}}$ for both matrices are smaller than $2^{-51}$. Figs. 1-(b) and 2-(b) give the graphs of $E_\sigma$. Though in dgesvd $\max E_\sigma = 2^{-49}$ for $A_1$ and $\max E_\sigma = 2^{-51}$ for $A_2$, in hpsvd $\max E_\sigma = 2^{-53}$ for both matrices. It is remarkable that all singular values computed by hpsvd have 52-bit accuracy in terms of absolute error.

Figs. 1-(c) and 2-(c) show the graphs of $E_\sigma^{\mathrm{r}}$. We observe that, in hpsvd, the relative error of the computed small singular value is no small in spite of the fact that $E_{\boldsymbol{H}}$ and $E_\sigma$ are small. Let us recall that both matrices have no small condition number. In double precision arithmetic, it seems to be not easy to compute the singular values of such matrices with 53-bit relative accuracy.

## 3. Mixed precision version

In this section, we propose a mixed precision version of the hyperplane constrained method for SVD. The key point is not to fix the precision of arithmetic, but to switch between double and multiple ones skillfully. The mixed precision version of hppair is as follows.

```
 1. function $(\sigma, \boldsymbol{u}, \boldsymbol{v}) =$ hppair_mix_*($A, \boldsymbol{z}, \tilde{\boldsymbol{u}}, \tilde{\boldsymbol{v}}, C, \ell_{\max}$)
 2. $\bar{A} :=$ mp($A$); $\bar{\boldsymbol{z}} :=$ mp($\boldsymbol{z}$); $\bar{C} :=$ mp($C$)
 3. $\bar{\boldsymbol{w}} := \bar{A}^T\bar{\boldsymbol{z}}$; $\boldsymbol{w} :=$ double($\bar{\boldsymbol{w}}$)
 4. $\bar{\boldsymbol{u}} :=$ mp($\tilde{\boldsymbol{u}}$); $\bar{\boldsymbol{v}} :=$ mp($\tilde{\boldsymbol{v}}$)
 5. $\bar{\alpha} := \bar{C}/(\bar{\boldsymbol{z}}^T\bar{\boldsymbol{u}}); \bar{\boldsymbol{u}} := \bar{\alpha}\bar{\boldsymbol{u}}; \bar{\boldsymbol{v}} := \bar{\alpha}\bar{\boldsymbol{v}}$
 6. $\boldsymbol{u} :=$ double($\bar{\boldsymbol{u}}$); $\boldsymbol{v} :=$ double($\bar{\boldsymbol{v}}$)
 7. $\bar{\sigma} := \bar{\boldsymbol{w}}^T\bar{\boldsymbol{v}}/\bar{C}$; $\sigma :=$ double($\bar{\sigma}$)
 8. $\ell := 0$
 9. while ($\ell \le \ell_{\max}$) do
10.    $\bar{\boldsymbol{H}} := \begin{pmatrix} \bar{A}\bar{\boldsymbol{v}}-\bar{\sigma}\bar{\boldsymbol{u}} \\ \bar{A}^T\bar{\boldsymbol{u}}-\bar{\sigma}\bar{\boldsymbol{v}} \end{pmatrix}$; $\boldsymbol{H} :=$ double($\bar{\boldsymbol{H}}$)
11.    $J := \begin{pmatrix} -\sigma I_m & A-\boldsymbol{u}\boldsymbol{w}^TC^{-1} \\ A^T & -\sigma I_n-\boldsymbol{v}\boldsymbol{w}^TC^{-1} \end{pmatrix}$
12.    $\boldsymbol{e} := J^{-1}\boldsymbol{H}$; $\bar{\boldsymbol{e}} :=$ mp($\boldsymbol{e}$)
13.    $\begin{pmatrix} \bar{\boldsymbol{u}} \\ \bar{\boldsymbol{v}} \end{pmatrix} := \begin{pmatrix} \bar{\boldsymbol{u}} \\ \bar{\boldsymbol{v}} \end{pmatrix} - \bar{\boldsymbol{e}}$
14.    $\bar{\alpha} := \bar{C}/(\bar{\boldsymbol{z}}^T\bar{\boldsymbol{u}}); \bar{\boldsymbol{u}} := \bar{\alpha}\bar{\boldsymbol{u}}; \bar{\boldsymbol{v}} := \bar{\alpha}\bar{\boldsymbol{v}}$
15.    $\boldsymbol{u} :=$ double($\bar{\boldsymbol{u}}$); $\boldsymbol{v} :=$ double($\bar{\boldsymbol{v}}$)
16.    $\bar{\sigma} := \bar{\boldsymbol{w}}^T\bar{\boldsymbol{v}}/\bar{C}$; $\sigma :=$ double($\bar{\sigma}$)
17.    $E_{\boldsymbol{H}} := \|\boldsymbol{H}\|_\infty/\|\boldsymbol{u}\|_\infty$
18.    if ($\|\boldsymbol{e}\|_\infty < 2^{-53}\|\boldsymbol{u}\|_\infty$) then break; end if
19.    $\ell := \ell + 1$
20. end while
```

```
21. $\boldsymbol{u} := \boldsymbol{u}/\|\boldsymbol{u}\|_2$; $\boldsymbol{v} := \boldsymbol{v}/\|\boldsymbol{v}\|_2$
22. if $(\sigma < 0)$ then $\sigma := -\sigma; \boldsymbol{v} := -\boldsymbol{v}$; end if
```

Here, the variables with bar and without bar are stored as multiple and double precision floating point number, respectively. By the function `mp`, double precision number is cast into multiple one. While, by the function `double`, multiple precision number is reduced to double one. Of course, by `double`, some bits of multiple precision number are eliminated. The input and output of `hppair_mix_*` are with double precision. In this paper, we discuss two cases where the significand of multiple precision number are 64 bits and 96 bits. As the multiple precision arithmetic, `hppair_mix_64` and `hppair_mix_96` employ 64-bit precision arithmetic and 96-bit one, respectively.

The vector $\boldsymbol{H}$ is computed with multiple precision, and then reduced to the double precision format. It takes $O((m+n)^3)$ operations for computing $\boldsymbol{e}$ such that $\boldsymbol{e} = J^{-1}\boldsymbol{H}$. This process, which is the dominant part of `hppair`, is carried out with double precision. The computed $\boldsymbol{e}$ is cast to the multiple precision format, and then $\bar{\boldsymbol{u}}$ and $\bar{\boldsymbol{v}}$ are updated as multiple precision vectors.

By replacing `hppair` with `hppair_mix_*` in `hpsvd`, we have the following routine.

```
 1. function $(\Sigma, U, V)$=hpsvd_mix_*$(A)$
 2. $C := 1$; $\ell_{\max} := 30$
 3. for $k = 1, 2, \ldots, m$ do
 4.     $\boldsymbol{z} :=$ select$(\mathcal{U}_{k-1}^{\perp})$
 5.     $\tilde{\boldsymbol{u}} :=$ select$(\mathcal{U}_{k-1}^{\perp})$; $\tilde{\boldsymbol{v}} :=$ select$(\mathcal{V}_{k-1}^{\perp})$
 6.     $(\sigma_k, \boldsymbol{u}_k, \boldsymbol{v}_k) :=$ hppair_mix_*$(A, \boldsymbol{z}, \tilde{\boldsymbol{u}}, \tilde{\boldsymbol{v}}, C, \ell_{\max})$
 7. end for
 8. $(\boldsymbol{v}_{m+1}, \boldsymbol{v}_{m+2}, \ldots, \boldsymbol{v}_n) :=$ kernel$(\mathcal{V}_m)$
 9. $\Sigma := (\mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_m)\, O_{m,n-m})$
10. $U := (\boldsymbol{u}_1\, \boldsymbol{u}_2\, \cdots\, \boldsymbol{u}_m)$; $V := (\boldsymbol{v}_1\, \boldsymbol{v}_2\, \cdots\, \boldsymbol{v}_n)$
```

It is emphasized here that `hpsvd_mix_*` requires multiple precision arithmetic in only `hppair_mix_*`. Let us recall that `hpsvd` equips with the additional process for orthogonalizing computed singular vectors in order to improve the accuracy of them. The routine `hpsvd_mix_*` differs from the original `hpsvd` in which it does not employ the orthogonalization process.

Let $E_{\boldsymbol{u}} = \|\boldsymbol{u} - \boldsymbol{u}^*\|_\infty / \|\boldsymbol{u}^*\|_2$ and $E_{\boldsymbol{v}} = \|\boldsymbol{v} - \boldsymbol{v}^*\|_\infty / \|\boldsymbol{v}^*\|_2$ be the relative errors of computed singular vectors $\boldsymbol{u}$ and $\boldsymbol{v}$, respectively. Similar to $\sigma^*$ in Section 2, we compute $\boldsymbol{u}^*$ and $\boldsymbol{v}^*$ with the help of 224-bit precision arithmetic. With respect to $E_\sigma^{\mathrm{r}}, E_{\boldsymbol{u}}$ and $E_{\boldsymbol{v}}$, we compare the mixed precision `hpsvd_mix_*` with the original `hpsvd`, LAPACK routine `dgesvd` and the routine `iisvd` in [4] based on inverse iteration method.

Figs. 1-(c) and 2-(c) illustrate the graphs of the relative error $E_\sigma^{\mathrm{r}}$. In `hpsvd_mix_64`, some of computed $E_\sigma^{\mathrm{r}}$ for $A_2$ are larger than $2^{-53}$. In `hpsvd_mix_96`, all $E_\sigma^{\mathrm{r}}$ for both $A_1$ and $A_2$ become less than $2^{-53}$. Figs. 1-(d), 2-(d) and 1-(e), 2-(e) show the errors $E_{\boldsymbol{u}}$ and $E_{\boldsymbol{v}}$, respectively. In `hpsvd_mix_64`, some $E_{\boldsymbol{u}}$ for $A_2$ and some $E_{\boldsymbol{v}}$ for $A_1, A_2$ are larger than $2^{-53}$. In `hpsvd_mix_96`, all $E_{\boldsymbol{u}}$ and $E_{\boldsymbol{v}}$ for $A_1$ and $A_2$ are smaller than $2^{-53}$. It is concluded that all singular pairs computed by `hpsvd_mix_96` have 53-bit accuracy. In `hpsvd`, `dgesvd` and `iisvd`, the most of $E_{\boldsymbol{u}}$ and $E_{\boldsymbol{v}}$ for $A_1$ and $A_2$ are larger than $2^{-53}$.

## 4. Mixed precision hybrid version

In [4], the hybrid method is designed by combining the hyperplane constrained method with other fast SVD method. Based on the routine of the hybrid method, we give the following routine as its mixed precision version.

```
 1. function $(\Sigma, U, V)$=hybridsvd_mix_*$(A)$
 2. $C := 1$; $\ell_{\max} := 20$
 3. $(\sigma_1, \ldots, \sigma_m, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_m, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_m) :=$ fast_svd$(A)$
 4. for $k = 1, 2, \ldots, m$ do
 5.     $\boldsymbol{z} :=$ select$(\mathcal{U}_{k-1}^{\perp})$
 6.     $(\sigma_k, \boldsymbol{u}_k, \boldsymbol{v}_k) :=$ hppair_mix_*$(A, \boldsymbol{z}, \boldsymbol{u}_k, \boldsymbol{v}_k, C, \ell_{\max})$
 7. end for
 8. $(\boldsymbol{v}_{m+1}, \boldsymbol{v}_{m+2}, \ldots, \boldsymbol{v}_n) :=$ kernel$(\mathcal{V}_m)$
 9. $\Sigma := (\mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_m)\, O_{m,n-m})$
10. $U := (\boldsymbol{u}_1\, \boldsymbol{u}_2\, \cdots\, \boldsymbol{u}_m)$; $V := (\boldsymbol{v}_1\, \boldsymbol{v}_2\, \cdots\, \boldsymbol{v}_n)$
```

Here the original hybrid routine `hybridsvd` shown in [4] is given by replacing `hppair_mix_*` with `hppair` in `hybridsvd_mix_*`. The routine `hybridsvd_mix_*` begins to solve the SVD roughly by a fast SVD routine. In this paper, we adopt the LAPACK routine `dgesvd` as `fast_svd`. The computed singular pairs by `fast_svd` are used as the initial guesses of `hppair_mix_*`. The function `hppair_mix_*` allows us to improve the accuracy of the computed singular pairs by `fast_svd`. The approximate singular vectors by `fast_svd` also play a key role for reducing the number of iterations from the 9th line to the 20th line in `hppair_mix_*`. Similar to `hpsvd_mix_*`, the mixed precision hybrid version `hybridsvd_mix_*` does not require multiple precision arithmetic except for `hppair_mix_*`.

From Figs. 1-(c), 1-(f), 2-(c) and 2-(f), it turns out that the graphs of $E_\sigma^{\mathrm{r}}$ in `hybridsvd_mix_64` and `hybridsvd_mix_96` are almost the same as those of `hpsvd_mix_64` and `hpsvd_mix_96`, respectively. Also, the graphs of $E_{\boldsymbol{u}}$ and $E_{\boldsymbol{v}}$ in `hybridsvd_mix_*` become similar to those of `hpsvd_mix_*`. Table 1 shows the average number of iterations, for computing 50 singular pairs, from the 9th line to the 20th line in `hppair_mix_*` and the running time. The iteration number and the running time of `hybridsvd_mix_96` for both matrices are less than those of `hpsvd_mix_96`. It is concluded that `hybridsvd_mix_96` is a speed-up version derived from `hpsvd_mix_96` without changing the accuracy virtually.

## 5. Conclusion

In this paper, the mixed double-multiple precision version of the hyperplane constrained method and its hybrid method are proposed. Numerically, it is observed that the SVD computed by new versions are relatively exact in double precision.

## References

[1] K. Kondo, S. Sugimoto and M. Iwasaki, An SVD algorithm based on solving nonlinear systems (in Japanese), Trans. JSIAM, **19** (2009), 81–103.

[2] K. Yadani, K. Kondo and M. Iwasaki, A singular value decomposition algorithm based on solving hyperplane constrained nonlinear systems, Appl. Math. Comput., **216** (2010), 779–790.

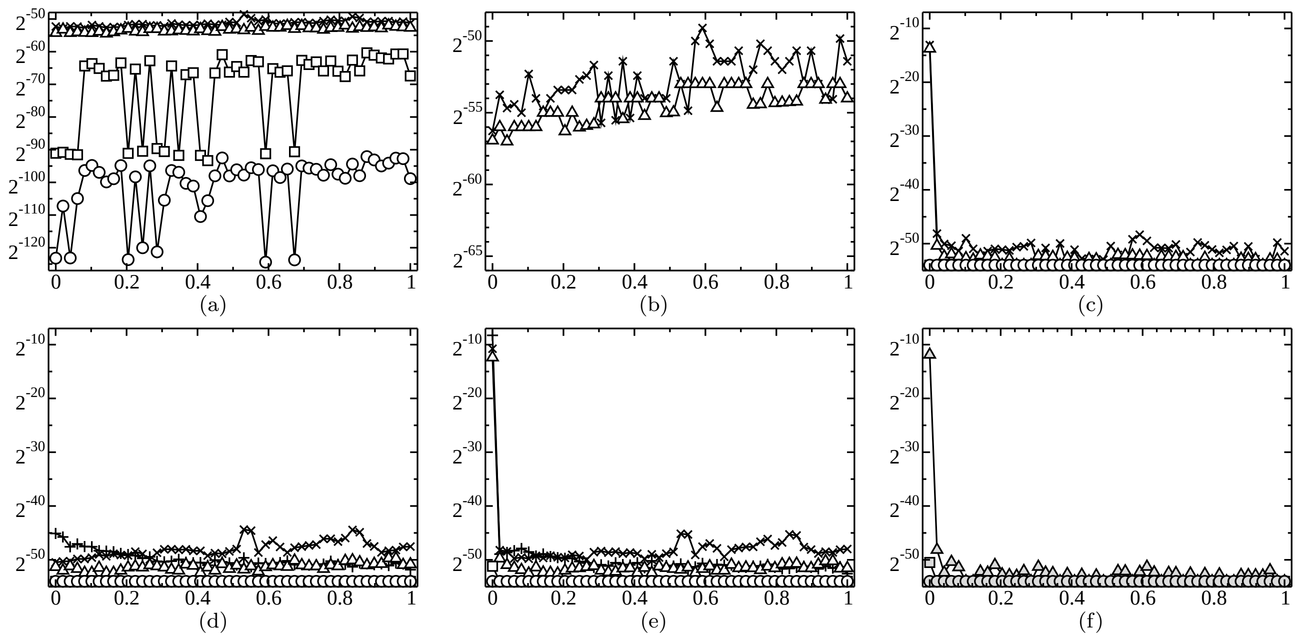

Fig. 1. Graphs of singular values (horizontal axis) and the values of $E_{\boldsymbol{H}}$ in (a), $E_\sigma$ in (b), $E_\sigma^{\mathrm{r}}$ in (c) and (f), $E_{\boldsymbol{u}}$ in (d) and $E_{\boldsymbol{v}}$ in (e) (vertical axis with logarithmic scale) for the SVD of $A_1$. ×: `dgesvd`, +: `iisvd`, △: `hpsvd`, □: `hpsvd_mix_64`, ◯: `hpsvd_mix_96`, ▲: `hybridsvd`, ■ : `hybridsvd_mix_64`, ● : `hybridsvd_mix_96`.

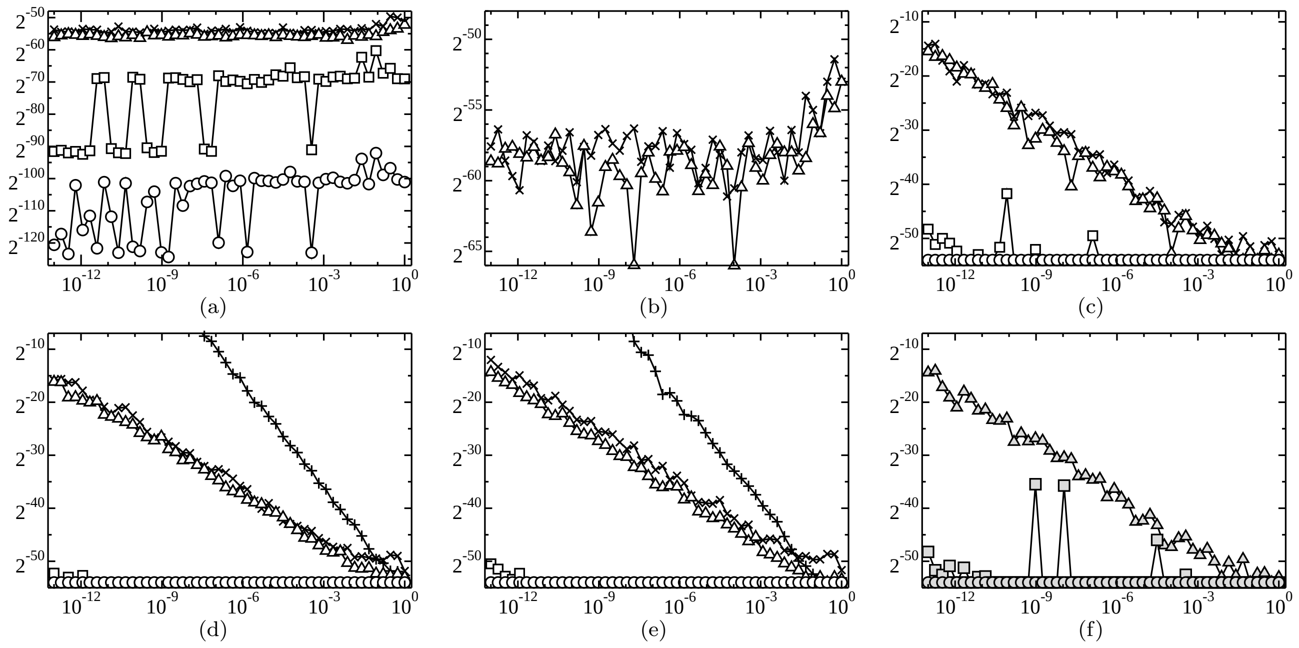

Fig. 2. Graphs of singular values (horizontal axis with logarithmic scale) and the values of $E_{\boldsymbol{H}}$ in (a), $E_\sigma$ in (b), $E_\sigma^{\mathrm{r}}$ in (c) and (f), $E_{\boldsymbol{u}}$ in (d) and $E_{\boldsymbol{v}}$ in (e) (vertical axis with logarithmic scale) for the SVD of $A_2$. ×: `dgesvd`, +: `iisvd`, △: `hpsvd`, □: `hpsvd_mix_64`, ◯: `hpsvd_mix_96`, ▲: `hybridsvd`, ■ : `hybridsvd_mix_64`, ● : `hybridsvd_mix_96`.

[3] K. Yadani, K. Kondo and M. Iwasaki, On the convergence of the V-type hyperplane constrained method for singular value decomposition, JSIAM Letters, **2** (2010), 21–24.

[4] K. Yadani, K. Kondo and M. Iwasaki, Numerical performance of hyperplane constrained method and its hybrid method for singular value decomposition, submitted.

[5] LAPACK, http://www.netlib.org/lapack/.

[6] GMP, http://gmplib.org/.

Table 1. The average of iteration number $\ell$ and running time $t$ (in seconds).

| | $A_1$ | | $A_2$ | |
|---|---|---|---|---|
| | $\ell$ | $t$ | $\ell$ | $t$ |
| `dgesvd` | – | 0.02 | – | 0.01 |
| `hpsvd` | 10.54 | 4.12 | 26.28 | 11.35 |
| `hpsvd_mix_64` | 10.36 | 5.19 | 12.76 | 7.03 |
| `hpsvd_mix_96` | 10.14 | 5.14 | 11.68 | 6.26 |
| `hybridsvd` | 2.30 | 1.08 | 0.10 | 0.09 |
| `hybridsvd_mix_64` | 2.38 | 1.48 | 5.52 | 3.07 |
| `hybridsvd_mix_96` | 2.06 | 1.29 | 2.54 | 1.61 |