

# GSVD in Julia

Prepared by Ji Wang (jiiwang@ucdavis.edu)

May 24, 2021

## Contents

0.1	GSVD in LAPACK and JuliaX.X . . . . .	2
0.2	GSVD in Edelman (2019) . . . . .	4
0.3	GSVD in MATLAB . . . . .	5
0.4	GSVD in Golub and Van Loan . . . . .	7
0.5	Examples . . . . .	8
0.6	GSVD algorithm . . . . .	24
0.7	GSVD of $Q_1$ and $Q_2$ . . . . .	28
0.8	Remarks . . . . .	32
<b>1</b>	<b>Software</b>	<b>33</b>
1.1	Why Julia . . . . .	33
1.2	Interfaces of GSVD in Julia . . . . .	38
1.3	GSVD module and implementation details . . . . .	39
1.4	Lessons and caveats . . . . .	41
1.5	GSVD in other languages . . . . .	42
1.6	Accuracy (backward stability) . . . . .	44
1.6.1	Numerical examples of small matrices . . . . .	44
1.6.2	Random dense matrices . . . . .	44
1.6.3	Special types of matrices . . . . .	46
1.7	Timing . . . . .	47
1.8	Prominent applications of the GSVD . . . . .	51
1.9	GSVD in genome analysis . . . . .	51
1.9.1	Mathematical framework: the GSVD . . . . .	52
1.9.2	Experiment . . . . .	53

## 0.1 GSVD in LAPACK and JuliaX.X

**Definition.** According to LAPACK [2, pp. 23–24], the generalized singular value decomposition (GSVD) of an  $m$ -by- $n$  matrix  $A$  and a  $p$ -by- $n$  matrix  $B$  is given by the pair of factorizations:

$$A = UC \begin{bmatrix} 0 & R \end{bmatrix} Q^T, \quad B = VS \begin{bmatrix} 0 & R \end{bmatrix} Q^T \quad (0.1)$$

where

- $U$  is  $m$ -by- $m$ ,  $V$  is  $p$ -by- $p$ ,  $Q$  is  $n$ -by- $n$  and all three matrices are orthogonal.
- $R$  is a  $(k + \ell)$ -by- $(k + \ell)$ , upper triangular and nonsingular,  $\begin{bmatrix} 0 & R \end{bmatrix}$  is  $k + \ell$ -by- $n$ .
- $C$  is  $m$ -by- $(k + \ell)$  and  $S$  is  $p$ -by- $(k + \ell)$ , both are real non-negative diagonal (returned in the arrays  $\alpha$  and  $\beta$ ), and  $C^T C + S^T S = I_{k+\ell}$ .  $C$  and  $S$  have the following detailed structures:

(1) Case  $m \geq k + \ell$ :

$$C = \begin{matrix} & k & \ell \\ & k & \ell \\ \begin{matrix} k \\ \ell \\ m - k - \ell \end{matrix} & \begin{pmatrix} I & 0 \\ 0 & \Sigma_1 \\ 0 & 0 \end{pmatrix} \end{matrix}, \quad S = \begin{matrix} & k & \ell \\ & \ell & k \\ \begin{matrix} \ell \\ p - \ell \end{matrix} & \begin{pmatrix} 0 & \Sigma_2 \\ 0 & 0 \end{pmatrix} \end{matrix},$$

where  $\Sigma_1$  and  $\Sigma_2$  are diagonal matrices. and  $\Sigma_1^2 + \Sigma_2^2 = I_\ell$  and  $\Sigma_2$  is nonsingular. In this case,

$$\alpha_1 = \cdots = \alpha_k = 1, \quad (\Sigma_1)_{ii} = \alpha_{k+i} \text{ for } i = 1, \dots, \ell, \\ \beta_1 = \cdots = \beta_k = 0, \quad (\Sigma_2)_{ii} = \beta_{k+i} \text{ for } i = 1, \dots, \ell.$$

(2) Case  $m < k + \ell$ :

$$C = \begin{matrix} & k & m - k & k + \ell - m \\ & k & m - k & k + \ell - m \\ \begin{matrix} k \\ m - k \end{matrix} & \begin{pmatrix} I & 0 & 0 \\ 0 & \Sigma_1 & 0 \end{pmatrix} \end{matrix}, \quad S = \begin{matrix} & k & m - k & k + \ell - m \\ & m - k & k + \ell - m & m - k \\ \begin{matrix} k + \ell - m \\ p - \ell \end{matrix} & \begin{pmatrix} 0 & \Sigma_2 & 0 \\ 0 & 0 & I \\ 0 & 0 & 0 \end{pmatrix} \end{matrix},$$

where  $\Sigma_1$  and  $\Sigma_2$  are diagonal matrices and  $\Sigma_1^2 + \Sigma_2^2 = I$ , and  $\Sigma_2$  is nonsingular. In this case,

$$\alpha_1 = \cdots = \alpha_k = 1, \quad (\Sigma_1)_{ii} = \alpha_{k+i} \text{ for } i = 1, \dots, m - k, \quad \alpha_{m+1} = \cdots = \alpha_{k+\ell} = 0. \\ \beta_1 = \cdots = \beta_k = 0, \quad (\Sigma_2)_{ii} = \beta_{k+i} \text{ for } i = 1, \dots, m - k, \quad \beta_{m+1} = \cdots = \beta_{k+\ell} = 1.$$

**Q:** can two cases be consolidated into one as the definition by Edelman (0.7)?

### Essential properties.

*Property 1.*  $k + \ell = \text{rank}([A; B])$  and  $\ell = \text{rank}(B)$ .

*Property 2.*  $\alpha_i, \beta_i \in [0, 1]$  for  $i = 1, \dots, k + \ell$ . The ratios

$$\sigma_i \equiv \alpha_i / \beta_i \quad (0.2)$$

are called the **generalized singular values** of the pair  $(A, B)$ , and are in non-increasing order. The first  $k$  values are infinite, the remaining  $\ell$  values are finite.

*Property 3.* If we rewrite the GSVD (0.1) as

$$A \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} = UC \begin{bmatrix} 0 & R_0 \end{bmatrix}, \quad B \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} = VS \begin{bmatrix} 0 & R_0 \end{bmatrix} \quad (0.3)$$

where  $Q_1$  is  $n$ -by- $(n - k - \ell)$ ,  $Q_2$  is  $n$ -by- $(k + \ell)$  and  $R_0$  is  $(k + \ell)$ -by- $(k + \ell)$ . Then,

$$\text{null}(A) \cap \text{null}(B) = \text{span}(Q_1),$$

i.e.,  $Q_1$  is an orthonormal basis of the common nullspace of  $A$  and  $B$ .

*Property 4.* Let

$$X = Q \begin{matrix} n - k - \ell & k + \ell \\ n - k - \ell & \\ k + \ell & \end{matrix} \begin{pmatrix} I & 0 \\ 0 & R^{-1} \end{pmatrix}, \quad (0.4)$$

then  $A^T A$  and  $B^T B$  are simultaenously diagonalized:

$$X^T A^T A X = \begin{matrix} n - k - \ell & k + \ell \\ n - k - \ell & \\ k + \ell & \end{matrix} \begin{pmatrix} 0 & 0 \\ 0 & C^T C \end{pmatrix}, \quad (0.5a)$$

$$X^T B^T B X = \begin{matrix} n - k - \ell & k + \ell \\ n - k - \ell & \\ k + \ell & \end{matrix} \begin{pmatrix} 0 & 0 \\ 0 & S^T S \end{pmatrix}. \quad (0.5b)$$

Thus, we know the “non-trivial” eigenpairs of the generalized eigenvalue problem:

$$A^T A X_{i+n-k-\ell} = \lambda_i B^T B X_{i+n-k-\ell}$$

for  $i = 1, \dots, k + \ell$ , where  $\lambda_i = (\alpha_i / \beta_i)^2$  are “non-trivial” eigenvalues of  $(A^T A, B^T B)$ .  $X_{i+n-k-\ell}$  denotes the  $(i + n - k - \ell)$ th column of  $X$  and are the corresponding eigenvectors.

*Property 5.* Two special cases of the GSVD:

- (a) When  $B$  is square and nonsingular, the GSVD of  $A$  and  $B$  is equivalent to the SVD of  $AB^{-1}$ :

$$AB^{-1} = U(CS^{-1})V^T$$

- (b) If the columns of  $\begin{bmatrix} A^T & B^T \end{bmatrix}^T$  are orthonormal, then the GSVD of  $A$  and  $B$  is equivalent to the Cosine-Sine decomposition (CSD) of  $(A^T, B^T)^T$ :

$$A = UCQ^T, \quad B = VSQ^T \quad (0.6)$$

where  $U$  is  $m$ -by- $m$ ,  $V$  is  $p$ -by- $p$  and  $Q$  is  $n$ -by- $n$  and all of them are orthogonal matrices.

## 0.2 GSVD in Edelman (2019)

In [15], the GSVD of an  $m$ -by- $n$  matrix  $A$  and a  $p$ -by- $n$  matrix  $B$  is defined as follows:

$$A = UCH, \quad B = VSH \quad (0.7)$$

where

- $U$  is  $m$ -by- $m$ , and  $V$  is a  $p$ -by- $p$ , and both are orthogonal matrices.
- $C$  is an  $m$ -by- $(k + \ell)$  matrix and  $S$  is an  $p$ -by- $(k + \ell)$  matrix, and  $C^T C + S^T S = I$ .  $C$  and  $S$  are of the following detailed structures:

$$C = \begin{matrix} & k & s & \ell - s \\ \begin{matrix} k \\ s \\ m - k - s \end{matrix} & \begin{pmatrix} I & 0 & 0 \\ 0 & \Sigma_1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \end{matrix}, \quad S = \begin{matrix} & k & s & \ell - s \\ \begin{matrix} p - \ell \\ s \\ \ell - s \end{matrix} & \begin{pmatrix} 0 & 0 & 0 \\ 0 & \Sigma_2 & 0 \\ 0 & 0 & I \end{pmatrix} \end{matrix},$$

where  $k + \ell = \text{rank}([A; B])$ ,  $\ell = \text{rank}(B)$ ,  $s = \text{rank}(A) + \text{rank}(B) - \text{rank}([A; B])$ . Furthermore,  $C$  and  $S$  are stored in the arrays  $\alpha$  and  $\beta$  of length  $k + \ell$  such that

$$\begin{aligned} \alpha_1 = \dots = \alpha_k = 1, \quad \Sigma_1 = \text{diag}(\alpha_{k+1}, \dots, \alpha_{k+s}), \quad \alpha_{k+s+1} = \dots = \alpha_{k+\ell} = 0, \\ \beta_1 = \dots = \beta_k = 0, \quad \Sigma_2 = \text{diag}(\beta_{k+1}, \dots, \beta_{k+s}), \quad \beta_{k+s+1} = \dots = \beta_{k+\ell} = 1. \end{aligned}$$

- $H$  is an  $(k + \ell)$ -by- $n$  matrix and has full row rank.

A few remarks are on order:

1. All properties in Section 0.1 hold true by the definition (0.7). In particular, by the RQ factorization of  $H$ :  $H = \begin{bmatrix} 0 & R_0 \end{bmatrix} Q^T$ , where  $R_0$  is an  $(k + \ell)$ -by- $(k + \ell)$  upper triangular matrix and  $Q$  is an  $n$ -by- $n$  orthogonal matrix, then

$$\text{null}(A) \cap \text{null}(B) = \text{span}\{Q(:, 1 : n - k - \ell)\}.$$

In addition, let  $X = Q \begin{pmatrix} I & 0 \\ 0 & R_0^{-1} \end{pmatrix}$ , then the “non-trivial” eigenvalues of the generalized eigenvalue problem  $A^T A x = \lambda B^T B x$  are the square of the generalized singular values of  $A$  and  $B$ , and the last  $(k + \ell)$  columns of  $X$  are the corresponding eigenvectors.

2. From the LAPACK GSVD (0.1) in Section 0.1, there is no value  $s$  to determine the  $s$ -by- $s$  blocks in (0.7). ... How to resolve this issue?

It seems that in the work by Paige and Saunders [26] and Bai and Demmel [4], there is a description of the  $s$ -blocks. ... need to double check.

Q: should we use the GSVD definitions (0.1) or (0.7) in “JuliaX.X”, or leave as options?

### 0.3 GSVD in MATLAB

In MATLAB 2019b [23], the GSVD of an  $m$ -by- $n$  matrix  $A$  and a  $p$ -by- $n$  matrix  $B$  is the following:

$$A = UCX^T, \quad B = VSX^T \quad (0.8)$$

where

- $U$  is  $m$ -by- $m$ ,  $V$  is  $p$ -by- $p$  and both matrices are orthogonal.
- $X$  is an  $n$ -by- $q$  matrix, where  $q = \min\{m + p, n\}$ .
- $C$  is  $m$ -by- $q$ ,  $S$  is  $p$ -by- $q$  and both matrices are nonnegative diagonal.

The nonzero elements of  $S$  are always on its main diagonal. The nonzero elements of  $C$  are on the diagonal  $\text{diag}(C, \max(0, q - m))$ . If  $m \geq q$ , this is the main diagonal of  $C$ .

Both  $C$  and  $S$  are nonnegative and  $C^T C + S^T S = I$ . If  $q > m$ , the rightmost  $m$ -by- $m$  block of  $C$  is diagonal. Otherwise, nonzero elements are on the main diagonal of  $C$ .

Furthermore,  $C^T C = \text{diag}(\alpha_1^2, \dots, \alpha_q^2)$ ,  $S^T S = \text{diag}(\beta_1^2, \dots, \beta_q^2)$ , where  $\alpha_i, \beta_i \in [0, 1]$  for  $i = 1, \dots, q$ . The ratios  $\alpha_i/\beta_i$  are called the generalized singular values of the pair  $(A, B)$  and are in non-decreasing order.

The following structures of  $C$  and  $S$  are not explicitly documented in MATLAB, but observed by the author.

1.  $m + p \geq n$ , thus  $q = n$ :

- (a)  $n > m, n \leq p$ :

$$C = \begin{matrix} & n-m & m \\ m & \begin{pmatrix} 0 & \Sigma_1 \end{pmatrix} \end{matrix}, \quad S = \begin{matrix} & n \\ p-n & \begin{pmatrix} \Sigma_2 \\ 0 \end{pmatrix} \end{matrix}$$

where  $\Sigma_1 = \text{diag}(\alpha_{n-m+1}, \dots, \alpha_n)$  and  $\Sigma_2 = \text{diag}(\beta_1, \dots, \beta_n)$ .

- (b)  $n \leq m, n > p$ :

$$C = \begin{matrix} & n \\ m-n & \begin{pmatrix} \Sigma_1 \\ 0 \end{pmatrix} \end{matrix}, \quad S = \begin{matrix} p & n-p \\ p & \begin{pmatrix} \Sigma_2 & 0 \end{pmatrix} \end{matrix}$$

where  $\Sigma_1 = \text{diag}(\alpha_1, \dots, \alpha_n)$  and  $\Sigma_2 = \text{diag}(\beta_1, \dots, \beta_p)$ .

- (c)  $n \leq m, n \leq p$ :

$$C = \begin{matrix} & n \\ m-n & \begin{pmatrix} \Sigma_1 \\ 0 \end{pmatrix} \end{matrix}, \quad S = \begin{matrix} & n \\ p-n & \begin{pmatrix} \Sigma_2 \\ 0 \end{pmatrix} \end{matrix}$$

where  $\Sigma_1 = \text{diag}(\alpha_1, \dots, \alpha_n)$  and  $\Sigma_2 = \text{diag}(\beta_1, \dots, \beta_n)$ .

- (d)  $n > m, n > p$ :

$$C = \begin{matrix} & n-m & m \\ m & \begin{pmatrix} 0 & \Sigma_1 \end{pmatrix} \end{matrix}, \quad S = \begin{matrix} p & n-p \\ p & \begin{pmatrix} \Sigma_2 & 0 \end{pmatrix} \end{matrix}$$

where  $\Sigma_1 = \text{diag}(\alpha_{n-m+1}, \dots, \alpha_n)$  and  $\Sigma_2 = \text{diag}(\beta_1, \dots, \beta_p)$ .

2.  $m + p < n$ , thus  $q = m + p$ :

$$C = \begin{matrix} & p & m \\ m & \begin{pmatrix} 0 & \Sigma_1 \end{pmatrix} \end{matrix}, \quad S = \begin{matrix} & p & m \\ p & \begin{pmatrix} \Sigma_2 & 0 \end{pmatrix} \end{matrix}$$

where  $\Sigma_1 = \text{diag}(\alpha_{p+1}, \dots, \alpha_{p+m})$  and  $\Sigma_2 = \text{diag}(\beta_1, \dots, \beta_p)$ .

A few remarks are in order:

1. The  $n \times q$  matrix  $X$  cannot be guaranteed to be of full rank  $q$ .
2. “The matrix  $X$  has full rank if and only if the matrix  $[A; B]$  has full rank. In fact, the SVD of  $X$  and the condition number of  $X$  are equal to the SVD of  $[A; B]$  and the condition number of  $[A; B]$ , respectively.”
3. The generalized singular values (gsvs) defined in (0.8) could be different from the ones defined in (0.2), see Examples 0.2 and 0.4.
4. By the definition (0.8), we have the factorizations of  $A^T A$  and  $B^T B$ :

$$A^T A = X C^T C X^T, \quad B^T B = X S^T S X^T. \quad (0.9)$$

However, since  $X$  is not guaranteed to be nonsingular, The factorization (0.9) is **not** the simultaneous diagonalization of  $(A^T A, B^T B)$  unless  $X$  is nonsingular. This implies that in general, there is **no connection** between MATLAB’s generalzied singular values (and singular vectors) and the “non-trivial” eigenpairs of  $(A^T A, B^T B)$ . **See Examples 0.2 and 0.4.**

Meanwhile, *Property 5* is true given this definition,.... (a) holds but (b) needs to be verified.

5. MATLAB’s GSVD (0.8) is also different from the one defined in Golub and Van Loan [16, pp. 309], see (0.10) below.

**MATLAB manual cites Golub and Van Loand, third edition, 1996... check the definition in the third edition**

6. **There are two examples on the MATLAB GSVD on MATLAB’s website. are we getting the same results? (looks like all full column rank.**

**Q: should we communicate with MATLAB about improperly defined GSVD (0.8)? how?**

## 0.4 GSVD in Golub and Van Loan

In Golub and Van Loan (4th edition) [16, pp. 309], given an  $m$ -by- $n$  matrix  $A$  and a  $p$ -by- $n$  matrix  $B$  with  $m \geq n$  and  $r = \text{rank}([A; B])$ , the GSVD of  $A$  and  $B$  is:

$$A = UCX^{-1}, \quad B = VSX^{-1} \quad (0.10)$$

where

- $U$  is an  $m$ -by- $m$  orthogonal matrix.
- $V$  is a  $p$ -by- $p$  orthogonal matrix.
- $C$  and  $S$  are  $m$ -by- $n$  and  $p$ -by- $n$ :

$$C = \begin{matrix} & \begin{matrix} q & r-q & n-r \end{matrix} \\ \begin{matrix} q \\ r-q \\ m-r \end{matrix} & \begin{pmatrix} I & 0 & 0 \\ 0 & \Sigma_1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \end{matrix}, \quad S = \begin{matrix} & \begin{matrix} q & r-q & n-r \end{matrix} \\ \begin{matrix} q \\ r-q \\ p-r \end{matrix} & \begin{pmatrix} 0 & 0 & 0 \\ 0 & \Sigma_2 & 0 \\ 0 & 0 & 0 \end{pmatrix} \end{matrix}$$

where  $q = \max\{r - p, 0\}$ . The diagonal elements of  $C$  and  $S$  are stored in the arrays  $\alpha$  and  $\beta$ :

$$\alpha_1 = \cdots = \alpha_q = 1, \quad \Sigma_1 = \text{diag}(\alpha_{q+1}, \dots, \alpha_r), \\ \beta_1 = \cdots = \beta_q = 0, \quad \Sigma_2 = \text{diag}(\beta_{q+1}, \dots, \beta_r)$$

and  $\Sigma_1^2 + \Sigma_2^2 = I$ .

- $X$  is an  $n$ -by- $n$  nonsingular matrix.

**Q: why there is no need to have two different cases for  $C$  and  $S$  as in LAPACK definition (0.1)?**

A few remarks are in order:

1. This definition is due to Van Loan [39]. It holds all properties in Section 0.1 Specifically, for *Property 3*, by  $A(X_1, X_2) = AX = UC = U(C_1, 0)$  and  $B(X_1, X_2) = BX = VS = V(S_1, 0)$ , then we have  $\text{null}(A) \cap \text{null}(B) = \text{span}(X_2)$ , although in this case,  $X_2$  is not an orthonormal basis.
2. The generalized singular value are elements of the set  $\mu(A, B) = \{\alpha_i/\beta_i \mid i = 1, \dots, r\}$ .
3.  $\text{rank}([A; B])$  is the number of “non-trivial???” diagonal entries of  $C$  and  $S$ .
4. By the definition (0.10),  $A^T A$  and  $B^T B$  are simultaneously diagonalized:

$$X^T A^T A X = C^T C, \quad X^T B^T B X = S^T S,$$

Therefore, the first  $r$  quotients of the diagonal entries of  $C^T C$  and  $S^T S$  are the “non-trivial” eigenvalues of the matrix pairs  $(A^T A, B^T B)$ , and the first  $r$  columns of  $X$  are the corresponding eigenvectors.

5. **Note: MATLAB GSVD (0.8) is also not in line with the GSVD (0.10)!**

## 0.5 Examples

We now illustrate our definition 0.1 and that of MATLAB's discussed in Section 0.3 with matrices of small size. Depending on the structures of  $C$  and  $S$  documented in Section 0.1, we devise four pairs: Examples 1 and 2 are contained in “case (1) ( $m \geq k + \ell$ )”, while Examples 3 and 4 fall into “case (2) ( $m < k + \ell$ )”.



**Example 0.1.** Consider a 5-by-4 matrix  $A$  and a 3-by-4 matrix  $B$ :

$$A = \begin{bmatrix} 1 & 2 & 3 & 0 \\ 5 & 4 & 2 & 1 \\ 0 & 3 & 5 & 2 \\ 2 & 1 & 3 & 3 \\ 2 & 0 & 5 & 3 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 0 & 3 & -1 \\ -2 & 5 & 0 & 1 \\ 4 & 2 & -1 & 2 \end{bmatrix}$$

where  $\text{rank}([A; B]) = 4$  and  $\text{rank}(B) = 3$ .

(1). The LAPACK GSVD (0.1) computed by “JuliaGSVD”:

$k = 1$  and  $\ell = 3$ . Since  $m = 5 \geq k + \ell = 1 + 3$ ,  $C$  and  $S$  are of the form in “case (1)”:

$$C = \begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.894685 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.600408 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.27751 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}, \quad S = \begin{bmatrix} 0.0 & 0.446698 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.799694 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.960723 \end{bmatrix}$$

The generalized singular values computed are

$$\text{Inf}, \quad 2.0028872436786482, \quad 0.7507971450334572, \quad 0.2888559753309598.$$

The computed orthogonal matrices  $U$ ,  $V$ ,  $Q$ , and the  $R$  matrix are:

$$U = \begin{bmatrix} -0.060976 & -0.446679 & -0.448921 & -0.482187 & -0.602266 \\ 0.0904806 & -0.867093 & 0.416172 & 0.115882 & 0.230944 \\ -0.481907 & -0.212508 & -0.636747 & 0.477322 & 0.298869 \\ -0.523214 & 0.0347528 & 0.410748 & 0.420777 & -0.615851 \\ -0.69434 & 0.0475385 & 0.226075 & -0.590913 & 0.339624 \end{bmatrix}$$

$$V = \begin{bmatrix} -0.804633 & -0.328486 & -0.494634 \\ -0.288044 & -0.512512 & 0.808927 \\ -0.519227 & 0.793365 & 0.317765 \end{bmatrix}$$

$$Q = \begin{bmatrix} 0.214542 & 0.484366 & 0.833941 & -0.15461 \\ 0.259709 & 0.413752 & -0.147691 & 0.85997 \\ -0.361334 & 0.767117 & -0.413972 & -0.331052 \\ -0.86946 & -0.0756949 & 0.333702 & 0.356304 \end{bmatrix}$$

$$R = \begin{bmatrix} 5.74065 & -7.07986 & 0.125979 & -0.316232 \\ 0.0 & -7.96103 & -2.11852 & -2.98601 \\ 0.0 & -4.44089e-16 & 5.72211 & -0.43623 \\ 0.0 & 1.33227e-15 & -8.88178e-16 & 5.66474 \end{bmatrix}$$

The residual norms are

$res_A = \frac{\ \tilde{U}^T A \tilde{Q} - \tilde{C} \tilde{R}\ _1}{\max(m,n) \ A\ _1 \varepsilon}$	0.35988438508439907
$res_B = \frac{\ \tilde{V}^T B \tilde{Q} - \tilde{S} \tilde{R}\ _1}{\max(p,n) \ B\ _1 \varepsilon}$	0.45714285714285713

- (2). By GSVD in Julia 1.3 (“`svd(A,B)`”), we have  $k = 1$  and  $\ell = 3$ .  $D1$  and  $D2$  (equivalent to  $C$  and  $S$  in LAPACK GSVD (0.1)) are:

$$D1 = \begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.894685 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.600408 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.27751 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}, \quad D2 = \begin{bmatrix} 0.0 & 0.446698 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.799694 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.960723 \end{bmatrix}$$

The computed orthogonal matrices  $U$ ,  $V$ ,  $Q$ , the  $R$  are

$$U = \begin{bmatrix} -0.060976 & -0.446679 & -0.448921 & 0.482187 & -0.602266 \\ 0.0904806 & -0.867093 & 0.416172 & -0.115882 & 0.230944 \\ -0.481907 & -0.212508 & -0.636747 & -0.477322 & 0.298869 \\ -0.523214 & 0.0347528 & 0.410748 & -0.420777 & -0.615851 \\ -0.69434 & 0.0475385 & 0.226075 & 0.590913 & 0.339624 \end{bmatrix}$$

$$V = \begin{bmatrix} -0.804633 & -0.328486 & 0.494634 \\ -0.288044 & -0.512512 & -0.808927 \\ -0.519227 & 0.793365 & -0.317765 \end{bmatrix}$$

$$Q = \begin{bmatrix} 0.214542 & 0.484366 & -0.833941 & 0.15461 \\ 0.259709 & 0.413752 & 0.147691 & -0.85997 \\ -0.361334 & 0.767117 & 0.413972 & 0.331052 \\ -0.86946 & -0.0756949 & -0.333702 & -0.356304 \end{bmatrix}$$

$$R0 = \begin{bmatrix} 5.74065 & -7.07986 & -0.125979 & 0.316232 \\ 0.0 & -7.96103 & 2.11852 & 2.98601 \\ 0.0 & 0.0 & -5.72211 & 0.43623 \\ 0.0 & 0.0 & 0.0 & 5.66474 \end{bmatrix}$$

The residual norms

$res_A = \frac{\ \tilde{U}^T A \tilde{Q} - \tilde{D}_1 \tilde{R}_0\ _1}{\max(m,n)\ A\ _1 \varepsilon}$	0.35988438508439907
$res_B = \frac{\ \tilde{V}^T B \tilde{Q} - \tilde{D}_2 \tilde{R}_0\ _1}{\max(p,n)\ B\ _1 \varepsilon}$	0.45714285714285713

- (3). The MATLAB GSVD (0.8) computed by “`gsvd(A,B)`”:

Since  $m + p = 5 + 3 > n = 4$ ,  $m = 5 > n = 4$  and  $p = 3 < n = 4$ , the structures of  $C$  and  $S$  are of is the “case 1.(b)” in Section 0.3:

$$C = \begin{bmatrix} 0.2775 & 0 & 0 & 0 \\ 0 & 0.6004 & 0 & 0 \\ 0 & 0 & 0.8947 & 0 \\ 0 & 0 & 0 & 1.0000 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad S = \begin{bmatrix} 0.9607 & 0 & 0 & 0 \\ 0 & 0.7997 & 0 & 0 \\ 0 & 0 & 0.4467 & 0 \end{bmatrix}$$

Consequently, the generalized singular values computed are:

$$0.2889, \quad 0.7508, \quad 2.0029, \quad \text{Inf.}$$

The computed  $U$ ,  $V$  and  $X$  matrix are

$$U = \begin{bmatrix} 0.4822 & -0.4489 & -0.4467 & -0.0610 & -0.6023 \\ -0.1159 & 0.4162 & -0.8671 & 0.0905 & 0.2309 \\ -0.4773 & -0.6367 & -0.2125 & -0.4819 & 0.2989 \\ -0.4208 & 0.4107 & 0.0348 & -0.5232 & -0.6159 \\ 0.5909 & 0.2261 & 0.0475 & -0.6943 & 0.3396 \end{bmatrix}$$

$$V = \begin{bmatrix} 0.4946 & -0.3285 & -0.8046 \\ -0.8089 & -0.5125 & -0.2880 \\ -0.3178 & 0.7934 & -0.5192 \end{bmatrix}$$

$$X = \begin{bmatrix} 0.8758 & 4.8394 & -5.1611 & -2.0437 \\ -4.8715 & -1.2203 & -5.5489 & -1.7290 \\ 1.8753 & -2.2244 & -4.2415 & -7.4528 \\ -2.0184 & 1.7541 & -1.1683 & -4.5260 \end{bmatrix}$$

The residual norms are

$res_A = \frac{\ A - \tilde{U}\tilde{C}\tilde{X}^T\ _1}{\max(m,n)\ A\ _1\varepsilon}$	0.5222
$res_B = \frac{\ B - \tilde{V}\tilde{S}\tilde{X}^T\ _1}{\max(p,n)\ B\ _1\varepsilon}$	1.3036

#### (4). Findings

- (a) The generalized singular values returned by “LAPACK-GSVD”, “GSVD-Julia1.3” and “MATLAB-GSVD” are the same, but are in different order.
- (b) All quantities computed by “LAPACK-GSVD” and “GSVD-Julia1.3” are the same, up to a sign difference.
- (c) The eigenvalues of  $(A^T A, B^T B)$  computed by MATLAB’s function `eig(A'*A, B'*B)` are

$$0.08343777448439993, \quad 0.5636963529903901, \quad 4.011557310890648, \quad \text{Inf}.$$

The square roots are

$$0.2888559753309596, \quad 0.7507971450334572, \quad 2.002887243678647, \quad \text{Inf}.$$

These values are equal to the gsvs computed by LAPACK-GSVD and MATLAB-GSVD.

Note: the “inf” eigenvalue is due to the fact  $B^T B$  is rank deficient.

- (d) The eigenvalues of  $(A^T A, B^T B)$  computed by MATLAB function `dsygvic(n, A'*A, B'*B, tol)`<sup>1</sup> are

$$4.0116, \quad 0.0834, \quad 0.5637.$$

The square roots are

$$2.0029, \quad 0.2889, \quad 0.7508.$$

---

<sup>1</sup><http://cmjiang.cs.ucdavis.edu/xsygvic.html>

(e) The  $X$  matrix produced by MATLAB is non-singular.

$$X = \begin{bmatrix} 0.8758 & 4.8394 & -5.1611 & -2.0437 \\ -4.8715 & -1.2203 & -5.5489 & -1.7290 \\ 1.8753 & -2.2244 & -4.2415 & -7.4528 \\ -2.0184 & 1.7541 & -1.1683 & -4.5260 \end{bmatrix}$$

(f) The eigenvectors of  $(A^T A, B^T B)$  computed by MATLAB function `dsygvic(n, A'*A, B'*B, tol)` are

$$X = \begin{bmatrix} 0.2106 & -0.0690 & 0.1377 \\ 0.2064 & 0.1052 & -0.0762 \\ 0.0904 & -0.0935 & -0.1074 \\ -0.3228 & 0.1450 & 0.1439 \end{bmatrix}$$

(g) From (0.4), the eigenvectors of  $(A^T A, B^T B)$  by GSVD properties in Julia are

$$X = \begin{bmatrix} 0.0373724 & -0.094078 & 0.110086 & -0.0663202 \\ 0.0452403 & -0.0922051 & -0.0609441 & 0.10104 \\ -0.062943 & -0.0403829 & -0.0859114 & -0.0898572 \\ -0.151457 & 0.144201 & 0.115041 & 0.139314 \end{bmatrix}$$

**Example 0.2.** Consider a 3-by-4 matrix  $A$  and a 4-by-4 matrix  $B$  but with rank deficiency:

$$A = \begin{bmatrix} 1 & 2 & 1 & 0 \\ 2 & 3 & 1 & 1 \\ 3 & 4 & 1 & 2 \end{bmatrix}, \quad B = \begin{bmatrix} 4 & 5 & 1 & 3 \\ 5 & 6 & 1 & 4 \\ 6 & 7 & 1 & 5 \\ 7 & 1 & -6 & 13 \end{bmatrix}$$

where  $\text{rank}([A; B]) = 2$  and  $\text{rank}(B) = 2$ .

(1). The LAPACK-GSVD (0.1) computed by “JuliaGSVD”:

$k = 0$  and  $\ell = 2$ . Since  $m = 3 > k + \ell = 0 + 2$ , the structure of  $C$  and  $S$  is “case 1”:

$$C = \begin{bmatrix} 0.476231 & 0.0 \\ 0.0 & 0.0697426 \\ 0.0 & 0.0 \end{bmatrix}, \quad S = \begin{bmatrix} 0.87932 & 0.0 \\ 0.0 & 0.997565 \\ 0.0 & 0.0 \\ 0.0 & 0.0 \end{bmatrix}$$

Consequently, the computed gsvs are

$$0.5415903238738987, \quad 0.06991284853891487.$$

The computed matrices  $U$ ,  $V$ ,  $Q$  and  $R$  are:

$$U = \begin{bmatrix} -0.409031 & 0.816105 & -0.408248 \\ -0.56342 & 0.126058 & 0.816497 \\ -0.71781 & -0.563988 & -0.408248 \end{bmatrix}$$

$$V = \begin{bmatrix} -0.472375 & -0.0876731 & -0.390874 & -0.785107 \\ -0.55599 & -0.135916 & -0.53894 & 0.618017 \\ -0.639606 & -0.184159 & 0.745532 & 0.0342253 \\ 0.242159 & -0.969498 & -0.0307137 & -0.0221441 \end{bmatrix}$$

$$Q = \begin{bmatrix} -0.436701 & -0.689898 & 0.299328 & 0.493696 \\ 0.563299 & 0.126599 & 0.793024 & 0.194368 \\ -0.689898 & 0.436701 & 0.493696 & -0.299328 \\ -0.126599 & 0.563299 & -0.194368 & 0.793024 \end{bmatrix}$$

$$R = \begin{bmatrix} 0.0 & 0.0 & -12.2133 & -8.28663 \\ 0.0 & 0.0 & 3.55271e-15 & -18.1154 \end{bmatrix}$$

We tested residues of  $A$  and  $B$  with the computed products  $\tilde{U}$ ,  $\tilde{V}$ ,  $\tilde{Q}$ ,  $\tilde{C}$ ,  $\tilde{S}$  and  $\tilde{R}$ .

$res_A = \frac{\ \tilde{U}^T A \tilde{Q} - \tilde{C} \tilde{R}\ _1}{\max(m,n)\ A\ _1 \varepsilon}$	0.6172649988387877
$res_B = \frac{\ \tilde{V}^T B \tilde{Q} - \tilde{S} \tilde{R}\ _1}{\max(p,n)\ B\ _1 \varepsilon}$	0.40979208210904405

(2). MATLAB GSVD (0.8) computed by “`gsvd(A,B)`”:

Since  $m + p = 3 + 4 > n=4$ ,  $m = 3 < n = 4$  and  $p = 4 = n = 4$ , the structures of  $C$  and  $S$  should be the same as case 1(a) of MATLAB GSVD (0.8) as follows:

$$C = \begin{bmatrix} 0 & 0.0460 & 0 & 0 \\ 0 & 0 & 0.6490 & 0 \\ 0 & 0 & 0 & 0.9946 \end{bmatrix}, \quad S = \begin{bmatrix} 1.0000 & 0 & 0 & 0 \\ 0 & 0.9989 & 0 & 0 \\ 0 & 0 & 0.7608 & 0 \\ 0 & 0 & 0 & 0.1039 \end{bmatrix}$$

Four computed generalized singular values are

$$0, \quad 0.0460, \quad 0.8531, \quad 9.5769.$$

The computed orthogonal matrices  $U$ ,  $V$  and the  $X$  matrix are:

$$U = \begin{bmatrix} 0.0438 & 0.0710 & 0.9965 \\ -0.7618 & -0.6430 & 0.0793 \\ 0.6464 & -0.7626 & 0.0259 \end{bmatrix}$$

$$V = \begin{bmatrix} 0.0621 & 0.0228 & -0.8563 & 0.5121 \\ -0.1574 & 0.3650 & -0.4722 & -0.7868 \\ -0.4326 & 0.8097 & 0.1962 & 0.3445 \\ 0.8855 & 0.4589 & 0.0720 & -0.0075 \end{bmatrix}$$

$$X = \begin{bmatrix} 3.0643 & 9.9974 & -5.3968 & 1.2397 \\ -2.7768 & 8.4399 & -7.4530 & 2.3475 \\ -5.8412 & -1.5575 & -2.0562 & 1.1078 \\ 8.9055 & 11.5549 & -3.3406 & 0.1319 \end{bmatrix}$$

We checked the residues of  $A$  and  $B$  with the computed  $\tilde{U}$ ,  $\tilde{V}$ ,  $\tilde{X}$ ,  $\tilde{C}$  and  $\tilde{S}$ .

$res_A = \frac{\ A - \tilde{U}\tilde{C}\tilde{X}^T\ _1}{\max(m,n)\ A\ _1\epsilon}$	5.5139
$res_B = \frac{\ B - \tilde{V}\tilde{S}\tilde{X}^T\ _1}{\max(p,n)\ B\ _1\epsilon}$	1.0600

### (3). Findings

- (a) The matrix  $X$  in MATLAB GSVD (0.8) is singular, with rank 2.
- (b) Neither do the diagonal entries of  $C$  and  $S$  nor the generalize singular values produced in LAPACK GSVD (0.1) and MATLAB GSVD (0.8) bear any resemblance in terms of the number of gsvs and their numerical values.
- (c) The eigenvalues of  $(A^T A, B^T B)$  computed by MATLAB's function `eig(A'*A, B'*B)` are

$$-0.035807289211371204, \quad \mathbf{0.004887806390825194}, \quad 0.12085659170178971, \quad \mathbf{0.2933200789138342}$$

The square roots are

$$0.0+0.1892281406434339im, \quad \mathbf{0.06991284853891447}, \quad 0.3476443465695792, \quad \mathbf{0.5415903238738985}$$

Among these values, eigenvalues **0.06991284853891447** and **0.5415903238738985** are found in the computed gsvs of LAPACK GSVD.

- (d) **Note:** In this case, two of four eigenvalues computed by MATLAB “eig” are spurious ones. This is caused by the fact that the pencil  $A^T A - \lambda B^T B$  is singular, i.e.,  $A^T A$  and  $B^T B$  have a non-trivial common null space. The function “dsygvic.m” should return only two “corrected” eigenvalues.

The computation of the eigenvalues of  $(A^T A, B^T B)$  by MATLAB’s function `dsygvic(n, A'*A, B'*B, tol)` caused an exception, the error message is **Singular pencil, exit**.

- (4). By GSVD in Julia 1.3 (`svd(A, B)`), we have  $k = 0$  and  $\ell = 2$ .  $D1$  and  $D2$  (equivalent to  $C$  and  $S$  in the proposed version) are:

$$D1 = \begin{bmatrix} 0.476231 & 0.0 \\ 0.0 & 0.0697426 \\ 0.0 & 0.0 \end{bmatrix}, \quad D2 = \begin{bmatrix} 0.87932 & 0.0 \\ 0.0 & 0.997565 \\ 0.0 & 0.0 \\ 0.0 & 0.0 \end{bmatrix}$$

The computed orthogonal matrices  $U, V, Q$ , the  $R0$  matrix (equivalent to  $R$  in the proposed version) are:

$$U = \begin{bmatrix} 0.409031 & 0.816105 & -0.408248 \\ 0.56342 & 0.126058 & 0.816497 \\ 0.71781 & -0.563988 & -0.408248 \end{bmatrix}$$

$$V = \begin{bmatrix} 0.472375 & -0.0876731 & -0.390874 & -0.785107 \\ 0.55599 & -0.135916 & -0.53894 & 0.618017 \\ 0.639606 & -0.184159 & 0.745532 & 0.0342253 \\ -0.242159 & -0.969498 & -0.0307137 & -0.0221441 \end{bmatrix}$$

$$Q = \begin{bmatrix} -0.436701 & -0.689898 & -0.299328 & 0.493696 \\ 0.563299 & 0.126599 & -0.793024 & 0.194368 \\ -0.689898 & 0.436701 & -0.493696 & -0.299328 \\ -0.126599 & 0.563299 & 0.194368 & 0.793024 \end{bmatrix}$$

$$R0 = \begin{bmatrix} 0.0 & 0.0 & -12.2133 & 8.28663 \\ 0.0 & 0.0 & 0.0 & -18.1154 \end{bmatrix}$$

**All these quantities are essentially (up to a sign) the same with JuliaGSVD.**

Still, we tested residuals of  $A$  and  $B$  with the computed products  $\tilde{U}, \tilde{V}, \tilde{Q}, \tilde{D}1, \tilde{D}2$  and  $\tilde{R}0$ .

$res_A = \frac{\ \tilde{U}^T A \tilde{Q} - \tilde{D}1 \tilde{R}0\ _1}{\max(m,n)\ A\ _1 \varepsilon}$	0.5068000688771875
$res_B = \frac{\ \tilde{V}^T B \tilde{Q} - \tilde{D}2 \tilde{R}0\ _1}{\max(p,n)\ B\ _1 \varepsilon}$	0.5689371432283518

**Example 0.3.** Let  $A$  be a 3-by-4 matrix and  $B$  be a 4-by-4 matrix:

$$A = \begin{bmatrix} 1 & 4 & 1 & 0 \\ 5 & 3 & 1 & 1 \\ 3 & 0 & 1 & 2 \end{bmatrix}, \quad B = \begin{bmatrix} 4 & 5 & 1 & 3 \\ -2 & 0 & 1 & 4 \\ 3 & 2 & 1 & -5 \\ 1 & 1 & -6 & 3 \end{bmatrix}$$

(1). The LAPACK GSVD (0.1) computed by “JuliaGSVD”:

$k = 0$  and  $\ell = 4$ . Since  $m = 3$  and  $m < k + \ell$ ,  $C$  and  $S$  should be contained in case (2) in Section 0.1:

$$C = \begin{bmatrix} 0.99144 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.681061 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.167854 & 0.0 \end{bmatrix}, \quad S = \begin{bmatrix} 0.130566 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.732227 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.985812 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

The generalized singular values computed are

$$7.593384394490093, 0.930122554989402, 0.17026951585960612, 0.0.$$

The computed orthogonal matrices  $U$ ,  $V$ ,  $Q$ , and the  $R$  matrix are:

$$U = \begin{bmatrix} -0.519777 & 0.747619 & 0.413398 \\ 0.470025 & 0.654341 & -0.592381 \\ 0.713378 & 0.113599 & 0.691511 \end{bmatrix}$$

$$V = \begin{bmatrix} 0.259832 & 0.927018 & 0.177229 & -0.20424 \\ -0.733955 & 0.0402919 & 0.652334 & -0.184789 \\ -0.597084 & 0.369645 & -0.576157 & 0.418206 \\ -0.1931. & -0.0487437 & -0.459449 & -0.865588 \end{bmatrix}$$

$$Q = \begin{bmatrix} -0.685431 & -0.564405 & -0.459976 & -0.00724571 \\ 0.681731 & -0.704114 & -0.149854 & -0.130423 \\ -0.127188 & -0.380896 & 0.646684 & 0.648491 \\ -0.221923 & -0.201466 & 0.589716 & -0.749931 \end{bmatrix}$$

$$R = \begin{bmatrix} -3.71474 & -2.42556 & -0.179891 & -0.941672 \\ -7.20246e-16 & -9.84284 & -1.8323 & -0.522579 \\ -8.91076e-17 & 2.04711e-15 & 6.16149 & -1.43582 \\ 1.84152e-15 & 1.41087e-15 & 1.2978e-15 & 8.05363 \end{bmatrix}$$

We tested residues of  $A$  and  $B$  with the computed products  $\tilde{U}, \tilde{V}, \tilde{Q}, \tilde{C}, \tilde{S}$  and  $\tilde{R}$ .

$res_A = \frac{\ \tilde{U}^T A \tilde{Q} - \tilde{C} \tilde{R}\ _1}{\max(m,n) \ A\ _1 \varepsilon}$	0.41810247019514407
$res_B = \frac{\ \tilde{V}^T B \tilde{Q} - \tilde{S} \tilde{R}\ _1}{\max(p,n) \ B\ _1 \varepsilon}$	0.8941315014073346



(2). MATLAB GSVD (0.8) computed by “`gsvd(A,B)`”:

$m + p = 3 + 4 \geq n = 4$ ,  $m = 3 < n = 4$  and  $p = 4 \leq n = 4$ , the structures of  $C$  and  $S$  are the same as case 1.(a) in Section 0.3.

$$C = \begin{bmatrix} 0 & 0.1679 & 0 & 0 \\ 0 & 0 & 0.6811 & 0 \\ 0 & 0 & 0 & 0.9914 \end{bmatrix}, \quad S = \begin{bmatrix} 1.0000 & 0 & 0 & 0 \\ 0 & 0.9858 & 0 & 0 \\ 0 & 0 & 0.7322 & 0 \\ 0 & 0 & 0 & 0.1306 \end{bmatrix}$$

The generalized singular values computed are:

$$0, 0.1703, 0.9301, 7.5934.$$

The computed orthogonal matrices  $U$ ,  $V$  and the  $X$  matrix are given below.

$$U = \begin{bmatrix} 0.4134 & -0.7476 & 0.5198 \\ -0.5924 & -0.6543 & -0.4700 \\ 0.6915 & -0.1136 & -0.7134 \end{bmatrix}$$

$$V = \begin{bmatrix} 0.2042 & 0.1772 & -0.9270 & -0.2598 \\ 0.1848 & 0.6523 & -0.0403 & 0.7340 \\ -0.4182 & -0.5762 & -0.3696 & 0.5971 \\ 0.8656 & -0.4594 & 0.0487 & 0.1931 \end{bmatrix}$$

$$X = \begin{bmatrix} 0.0584 & -2.8237 & -6.4020 & -4.0048 \\ 1.0504 & -0.7361 & -7.2732 & 0.6748 \\ -5.2227 & 3.0534 & -2.2253 & -0.6694 \\ 6.0397 & 4.7103 & -1.2944 & -1.9132 \end{bmatrix}$$

We checked the residues of  $A$  and  $B$  with the computed  $\tilde{U}$ ,  $\tilde{V}$ ,  $\tilde{X}$ ,  $\tilde{C}$  and  $\tilde{S}$ .

$res_A = \frac{\ A - \tilde{U}\tilde{C}\tilde{X}^T\ _1}{\max(m,n)\ A\ _1\epsilon}$	3.5278
$res_B = \frac{\ B - \tilde{V}\tilde{S}\tilde{X}^T\ _1}{\max(p,n)\ B\ _1\epsilon}$	0.5000

### (3). Findings

- (a) The generalized singular values computed by “JuliaGSVD” and “MATLAB-SVD” are the same. However, the order are opposite.
- (b) The eigenvalues of  $(A^T A, B^T B)$  computed by MATLAB’s function `eig(A'*A,B'*B)` are

$$-1.8035125057805033e^{-15}, \quad \mathbf{0.028991708031064364}, \quad \mathbf{0.8651279673000131}, \quad \mathbf{57.659486562484}$$

The square roots are

$$0.0+4.2467781973874066e^{-8}\mathrm{i}, \quad \mathbf{0.17026951585960526}, \quad \mathbf{0.930122554989402}, \quad \mathbf{7.5933843944900}$$

Among these values, eigenvalues **0.17026951585960526**, **0.930122554989402** and **7.5933843944900** are found in the computed gsvs of LAPACK GSVD and MATLAB GSVD.

- (c) One of the eigenvalues computed by MATLAB's `eig` is spurious. The eigenvalues of  $(A^T A, B^T B)$  computed by MATLAB's function `dsygvic(n, A'*A, B'*B, tol)` are

$$57.6595, \quad 0.8651, \quad 0.0000, \quad 0.0290.$$

The square roots are

$$7.5934, \quad 0.9301, \quad 0.0000, \quad 0.1703.$$

all of which are identical to the computed gsvs of LAPACK-GSVD and MATLAB-GSVD.

- (d) The  $X$  matrix produced by MATLAB is non-singular.

$$X = \begin{bmatrix} 0.0584 & -2.8237 & -6.4020 & -4.0048 \\ 1.0504 & -0.7361 & -7.2732 & 0.6748 \\ -5.2227 & 3.0534 & -2.2253 & -0.6694 \\ 6.0397 & 4.7103 & -1.2944 & -1.9132 \end{bmatrix}$$

- (e) The eigenvectors of  $(A^T A, B^T B)$  computed by MATLAB function `dsygvic(n, A'*A, B'*B, tol)` are

$$X = \begin{bmatrix} 1.4132 & 0.0162 & 0.0097 & -0.0667 \\ -1.4056 & 0.1595 & -0.0292 & 0.0051 \\ 0.2622 & 0.0413 & 0.1070 & 0.1166 \\ 0.4576 & 0.0078 & -0.0681 & 0.1006 \end{bmatrix}$$

- (f) From (0.4), the eigenvectors of  $(A^T A, B^T B)$  by GSVD properties in Julia are

$$X = \begin{bmatrix} 0.184517 & 0.0118715 & -0.0657359 & 0.00972571 \\ -0.183521 & 0.11676 & 0.005043 & -0.0291771 \\ 0.0342387 & 0.0302604 & 0.114954 & 0.106983 \\ 0.0597412 & 0.00574635 & 0.0991631 & -0.06808 \end{bmatrix}$$

- (4). Similarly, we test GSVD in Julia 1.3 with the same inputs. For the numerical rank,  $k = 0$  and  $\ell = 4$ .  $D1$  and  $D2$  (equivalent to  $C$  and  $S$  in the proposed version) are:

$$D1 = \begin{bmatrix} 0.99144 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.681061 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.167854 & 0.0 \end{bmatrix}, \quad D2 = \begin{bmatrix} 0.130566 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.732227 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.985812 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

The computed orthogonal matrices  $U$ ,  $V$ ,  $Q$ , the  $R0$  matrix (equivalent to  $R$  in the proposed

version) are:

$$\begin{aligned}
U &= \begin{bmatrix} 0.519777 & 0.747619 & 0.413398 \\ -0.470025 & 0.654341 & -0.592381 \\ -0.713378 & 0.113599 & 0.691511 \end{bmatrix} \\
V &= \begin{bmatrix} -0.259832 & 0.927018 & 0.177229 & 0.20424 \\ 0.733955 & 0.0402919 & 0.652334 & 0.184789 \\ 0.597084 & 0.369645 & -0.576157 & -0.418206 \\ 0.1931 & -0.0487437 & -0.459449 & 0.865588 \end{bmatrix} \\
Q &= \begin{bmatrix} -0.685431 & 0.564405 & 0.459976 & 0.00724571 \\ 0.681731 & 0.704114 & 0.149854 & 0.130423 \\ -0.127188 & 0.380896 & -0.646684 & -0.648491 \\ -0.221923 & 0.201466 & -0.589716 & 0.749931 \end{bmatrix} \\
R0 &= \begin{bmatrix} 3.71474 & -2.42556 & -0.179891 & -0.941672 \\ 0.0 & 9.84284 & 1.8323 & 0.522579 \\ 0.0 & 0.0 & -6.16149 & 1.43582 \\ 0.0 & 0.0 & 0.0 & 8.05363 \end{bmatrix}
\end{aligned}$$

All these quantities are essentially (up to a sign) the same with JuliaGSVD.

Still, we tested residuals of  $A$  and  $B$  with the computed products  $\tilde{U}, \tilde{V}, \tilde{Q}, \tilde{D}1, \tilde{D}2$  and  $\tilde{R}0$ .

$res_A = \frac{\ \tilde{U}^T A \tilde{Q} - \tilde{D}1 \tilde{R}0\ _1}{\max(m,n) \ A\ _1 \varepsilon}$	0.3536371804643456
$res_B = \frac{\ \tilde{V}^T B \tilde{Q} - \tilde{D}2 \tilde{R}0\ _1}{\max(p,n) \ B\ _1 \varepsilon}$	0.59375

**Example 0.4.** Given a 3-by-5 matrix  $A$  and a 4-by-5 matrix  $B$  which are rank deficient:

$$A = \begin{bmatrix} 1 & 4 & 2 & 3 & 0 \\ 3 & 4 & 0 & -2 & 1 \\ 4 & 7 & 5 & 6 & 3 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 4 & 2 & 3 & 0 \\ 2 & 5 & 3 & 4 & 1 \\ 3 & 6 & 4 & 5 & 2 \\ 0 & 1 & -1 & 3 & 1 \end{bmatrix}$$

(1). The LAPACK GSVD (0.1) computed by “JuliaGSVD”:

$k = 1, \ell = 3$  and  $m = 3 < k + \ell = 4$ . Both  $B$  and  $[A; B]$  are not in full rank, the structures of  $C$  and  $S$  comply with those of case (2) in Section 0.1.

$$C = \begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.849235 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.605834 & 0.0 \end{bmatrix}, \quad S = \begin{bmatrix} 0.0 & 0.528015 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.795591 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

The generalized singular values computed are

$$\text{Inf}, 1.6083530545973714, 0.7614900645668164, 0.0.$$

The computed orthogonal matrices  $U$ ,  $V$ ,  $Q$ , and the  $R$  matrix are:

$$U = \begin{bmatrix} -2.22045e-16 & 0.355381 & -0.934722 \\ 1.0 & -1.74736e-16 & -1.8521e-16 \\ -2.2915e-16 & -0.934722 & -0.355381 \end{bmatrix}$$

$$V = \begin{bmatrix} 0.571577 & -0.711781 & 1.07608e-17 & -0.408248 \\ -0.120069 & -0.564727 & -2.13123e-16 & 0.816497 \\ -0.811716 & -0.417673 & -1.59451e-16 & -0.408248 \\ 1.38917e-16 & 1.22399e-16 & -1.0 & 3.46945e-17 \end{bmatrix}$$

$$Q = \begin{bmatrix} -0.735494 & -0.356936 & -0.479812 & 0.318474 & 3.59984e-16 \\ 0.29657 & -0.540179 & 0.367864 & 0.633716 & 0.288675 \\ 0.130491 & 0.610611 & -0.189162 & 0.700722 & -0.288675 \\ -0.237256 & 0.432143 & 0.0711454 & 0.0435931 & 0.866025 \\ 0.545689 & -0.145639 & -0.770462 & -0.0637737 & 0.288675 \end{bmatrix}$$

$$R = \begin{bmatrix} 0.0 & -4.24145 & -0.880735 & 3.33933 & -0.288675 \\ 0.0 & 0.0 & 2.7394 & -8.38306 & -5.97906 \\ 0.0 & 0.0 & -1.77636e-15 & -12.2122 & -8.79399 \\ 0.0 & 0.0 & -4.996e-16 & 2.22045e-16 & -3.4641 \end{bmatrix}$$

We can verify that  $R$  has a zero column in the leftmost since  $k + l < n$ .

We tested residues of  $A$  and  $B$  with the computed products  $\tilde{U}, \tilde{V}, \tilde{Q}, \tilde{C}, \tilde{S}$  and  $\tilde{R}$ .

$res_A = \frac{\ \tilde{U}^T A \tilde{Q} - \tilde{C} \tilde{R}\ _1}{\max(m, n) \ A\ _1 \varepsilon}$	0.36
$res_B = \frac{\ \tilde{V}^T B \tilde{Q} - \tilde{S} \tilde{R}\ _1}{\max(p, n) \ B\ _1 \varepsilon}$	0.589976856064605

(2). MATLAB GSVD (0.8) computed by “`gsvd(A,B)`”:

$m + p = 3 + 4 \geq n = 5$  and  $m = 3 < n = 5$ ,  $p = 4 < n = 5$ , the structures of  $C$  and  $S$  are the form of case 1.(d) in Section 0.3.

$$C = \begin{bmatrix} 0 & 0 & 0.8178 & 0 & 0 \\ 0 & 0 & 0 & 0.9995 & 0 \\ 0 & 0 & 0 & 0 & 1.0000 \end{bmatrix}, \quad S = \begin{bmatrix} 1.0000 & 0 & 0 & 0 & 0 \\ 0 & 1.0000 & 0 & 0 & 0 \\ 0 & 0 & 0.5755 & 0 & 0 \\ 0 & 0 & 0 & 0.0312 & 0 \end{bmatrix}$$

The generalized singular values computed are

$$0, 0, 1.4209, 32.0780, \text{Inf}.$$

The computed orthogonal matrices  $U$ ,  $V$  and the  $X$  matrix are:

$$U = \begin{bmatrix} -0.1968 & 0.9805 & 0.0000 \\ 0.0000 & -0.0000 & 1.0000 \\ -0.9805 & -0.1968 & -0.0000 \end{bmatrix}$$

$$V = \begin{bmatrix} -0.8338 & 0 & 0.3365 & 0.4376 \\ -0.5289 & 0.0000 & -0.2600 & -0.8079 \\ -0.1581 & 0.0000 & -0.9051 & 0.3947 \\ -0.0000 & -1.0000 & -0.0000 & -0.0000 \end{bmatrix}$$

$$X = \begin{bmatrix} -2.3660 & 0.0000 & -5.0363 & 0.1935 & 3.0000 \\ -6.9285 & -1.0000 & -9.3550 & 2.5457 & 4.0000 \\ -3.8868 & 1.0000 & -6.4759 & 0.9776 & 0.0000 \\ -5.4077 & -3.0000 & -7.9154 & 1.7617 & -2.0000 \\ -0.8451 & -1.0000 & -3.5968 & -0.5906 & 1.0000 \end{bmatrix}$$

We checked the residues of  $A$  and  $B$  with the computed  $\tilde{U}$ ,  $\tilde{V}$ ,  $\tilde{X}$ ,  $\tilde{C}$  and  $\tilde{S}$ .

$res_A = \frac{\ A - \tilde{U}\tilde{C}\tilde{X}^T\ _1}{\max(m,n)\ A\ _1\varepsilon}$	0.4800
$res_B = \frac{\ B - \tilde{V}\tilde{S}\tilde{X}^T\ _1}{\max(p,n)\ B\ _1\varepsilon}$	0.4000

### (3). Findings

- (a) The matrix  $X$  in MATLAB GSVD (0.8) is singular, whose rank is 4.
- (b) Neither do the diagonal entries of  $C$  and  $S$  nor the generalize singular values produced in LAPACK GSVD (0.1) and MATLAB GSVD (0.8) share any in common in terms of the number of gsvs and their numerical values.
- (c) The eigenvalues of  $(A^T A, B^T B)$  computed by MATLAB’s function `eig(A'*A,B'*B)` are

$$-0.34554912453318243, 1.3025318975863486e^{-16}, \mathbf{0.5798671184339763}, \mathbf{2.586799548232693}, \text{Inf}.$$

The square roots are

$0.0+0.5878342662121547im, 1.1412851955520796e^{-8}, \mathbf{0.7614900645668178}, \mathbf{1.6083530545973708}$ , Im.

Among these values, eigenvalues **0.7614900645668178** and **1.6083530545973708** are found in the computed gsvs of LAPACK GSVD.

- (d) The computation of the eigenvalues of  $(A^T A, B^T B)$  by MATLAB's function `dsygvic(n, A'*A, B'*B, tol)` threw an error at line 328. I tried to change  $k$  to  $k(1)$ , thus the square roots of the eigenvalues are

**0.0046, 0.7710.**

which makes no sense.

- (4). Again, same inputs are tested in Julia 1.3. For the numerical rank determination,  $k = 1$  and  $\ell = 3$ .  $D1$  and  $D2$  (equivalent to  $C$  and  $S$  in the proposed version) are:

$$D1 = \begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.849235 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.605834 & 0.0 \end{bmatrix}, \quad D2 = \begin{bmatrix} 0.0 & 0.528015 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.795591 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

The computed orthogonal matrices  $U$ ,  $V$ ,  $Q$ , the  $R0$  matrix (equivalent to  $R$  in the proposed version) are:

$$U = \begin{bmatrix} -2.22045e-16 & -0.355381 & -0.934722 \\ 1.0 & 1.74736e-16 & -1.8521e-16 \\ -2.2915e-16 & 0.934722 & -0.355381 \end{bmatrix}$$

$$V = \begin{bmatrix} -0.571577 & -0.711781 & 1.94289e-16 & -0.408248 \\ 0.120069 & -0.564727 & 2.35922e-16 & 0.816497 \\ 0.811716 & -0.417673 & -1.82146e-17 & -0.408248 \\ 7.69338e-17 & 2.44055e-16 & 1.0 & 3.46945e-17 \end{bmatrix}$$

$$Q = \begin{bmatrix} -0.735494 & -0.356936 & -0.479812 & -0.318474 & -1.66533e-16 \\ 0.29657 & -0.540179 & 0.367864 & -0.633716 & -0.288675 \\ 0.130491 & 0.610611 & -0.189162 & -0.700722 & 0.288675 \\ -0.237256 & 0.432143 & 0.0711454 & -0.0435931 & -0.866025 \\ 0.545689 & -0.145639 & -0.770462 & 0.0637737 & -0.288675 \end{bmatrix}$$

$$R0 = \begin{bmatrix} 0.0 & -4.24145 & -0.880735 & -3.33933 & 0.288675 \\ 0.0 & 0.0 & -2.7394 & -8.38306 & -5.97906 \\ 0.0 & 0.0 & 0.0 & 12.2122 & 8.79399 \\ 0.0 & 0.0 & 0.0 & 0.0 & -3.4641 \end{bmatrix}$$

It is clear that the leftmost column of  $R0$  is all zeros.

All these quantities are essentially (up to a sign) the same with JuliaGSVD.

Still, we tested residuals of  $A$  and  $B$  with the computed products  $\tilde{U}$ ,  $\tilde{V}$ ,  $\tilde{Q}$ ,  $\tilde{D}1$ ,  $\tilde{D}2$  and  $\tilde{R}0$ .

$res_A = \frac{\ \tilde{U}^T A \tilde{Q} - \tilde{D} \tilde{1} \tilde{R} \tilde{0}\ _1}{\max(m,n) \ A\ _1 \varepsilon}$	0.4449492156962062
$res_B = \frac{\ \tilde{V}^T B \tilde{Q} - \tilde{D} \tilde{2} \tilde{R} \tilde{0}\ _1}{\max(p,n) \ B\ _1 \varepsilon}$	0.305570013362164

## 0.6 GSVD algorithm

The algorithm consists of four steps. First step is a pre-processing step where the input matrix pair  $A, B$  is reduced to a triangular pair while revealing their ranks [5]. We further reduce two upper triangular matrices to one upper triangular matrix in the QR decomposition step. Next is the safe diagonalization of a matrix with orthonormal columns that is partitioned into two blocks. [39] The last step is post-processing to get the final products of the decomposition.

**Step 1. Pre-processing.** To reduce “regular matrices to their triangular form and reveal rank”, we employ the QR decomposition with column pivoting followed by RQ decomposition [16] as well as QR decomposition. We detail this in nine substeps below.

Step 1(1) QR decomposition with column pivoting of  $B$ :

$$BP = V \begin{matrix} & \ell & n - \ell \\ \ell & & \\ p - \ell & \begin{pmatrix} B_{11} & B_{12} \\ 0 & 0 \end{pmatrix} \end{matrix} \quad (0.11)$$

Step 1(2) Update  $A := AP$

Step 1(3) Set  $Q := I_n$  and update  $Q := QP$

Step 1(4) If  $n > \ell$ :

- RQ decomposition of  $(B_{11} \ B_{12})$ :

$$\begin{matrix} & \ell & n - \ell & & n - \ell & \ell \\ \ell & (B_{11} & B_{12}) & = \ell & \begin{pmatrix} 0 & B_{13} \end{pmatrix} Z \end{matrix} \quad (0.12)$$

- Update  $A := AZ^T$
- Update  $Q := QZ^T$

Step 1(5) Partition

$$A = \begin{matrix} & n - \ell & \ell \\ m & \begin{pmatrix} A_1 & A_2 \end{pmatrix}, \end{matrix} \quad (0.13)$$

the QR decomposition with column pivoting of  $A_1$  is:

$$A_1 P_1 = U \begin{matrix} & k & n - \ell - k \\ m - k & \begin{pmatrix} A_{11} & A_{12} \\ 0 & 0 \end{pmatrix} \end{matrix} \quad (0.14)$$

Step 1(6) Update  $A_2 := U^T A_2$

Step 1(7) Partition:

$$A = \begin{matrix} & k & n - \ell - k & \ell \\ m - k & \begin{pmatrix} A_{11} & A_{12} & A_{13} \\ 0 & 0 & A_{23} \end{pmatrix} \end{matrix} \quad (0.15)$$



Step 1(8) Update  $Q(:, 1 : n - \ell) := Q(:, 1 : n - \ell)P_1$

Step 1(9) If  $n - \ell > k$ :

- RQ decomposition of  $(A_{11} \ A_{12})$ :

$$\begin{matrix} k & n - \ell - k & n - \ell - k & k \\ k \left( \begin{matrix} A_{11} & A_{12} \end{matrix} \right) = k \left( \begin{matrix} 0 & A_{12} \end{matrix} \right) Z_1 \end{matrix} \quad (0.16)$$

and it results

$$A = \begin{matrix} & n - \ell - k & k & \ell \\ m - k \left( \begin{matrix} 0 & A_{12} & A_{13} \\ 0 & 0 & A_{23} \end{matrix} \right) \end{matrix} \quad (0.17)$$

- Update  $Q(:, 1 : n - \ell) = Q(:, 1 : n - \ell)Z_1^T$

Step 1(10) If  $m > k$ :

Let

$$A_2 = \begin{matrix} & \ell \\ m - k \left( \begin{matrix} A_{13} \\ A_{23} \end{matrix} \right) \end{matrix} \quad (0.18)$$

- QR decomposition of  $A_{23}$ :

$$A_{23} = U_1 \begin{matrix} & \ell \\ m - k - \ell \left( \begin{matrix} A_{23} \\ 0 \end{matrix} \right) \end{matrix} \quad (0.19)$$

- Update  $U(:, k + 1 : m) = U(:, k + 1 : m)U_1$

In summary, combining (0.11), (0.12), (0.14), (0.16), (0.17) and (0.19), we have the following decomposition at the end of the pre-processing:

$$A = UR_AQ^T, \quad B = VR_BQ^T \quad (0.20)$$

where

- If  $m - k - \ell \geq 0$ :

$$R_A = \begin{matrix} & n - k - \ell & k & \ell \\ k \left( \begin{matrix} 0 & A_{12} & A_{13} \\ \ell \left( \begin{matrix} 0 & 0 & A_{23} \end{matrix} \right) \\ m - k - \ell \left( \begin{matrix} 0 & 0 & 0 \end{matrix} \right) \end{matrix} \right), \quad R_B = \begin{matrix} & n - k - \ell & k & \ell \\ p - \ell \left( \begin{matrix} 0 & 0 & B_{13} \\ 0 & 0 & 0 \end{matrix} \right) \end{matrix}$$

where  $A_{12}$  and  $B_{13}$  are non-singular upper triangular matrix.  $\ell$  is the rank of  $B$ ,  $k + \ell$  is the rank of  $[A^T \ B^T]^T$ .  $A_{23}$  is  $\ell$ -by- $\ell$  upper triangular.

- If  $m - k - \ell < 0$ :

$$R_A = \begin{matrix} & n-k-\ell & k & \ell \\ & & & \\ k & \begin{pmatrix} 0 & A_{12} & A_{13} \\ 0 & 0 & A_{23} \end{pmatrix} \\ m-k & \end{matrix}, \quad R_B = \begin{matrix} & n-k-\ell & k & \ell \\ & & & \\ \ell & \begin{pmatrix} 0 & 0 & B_{13} \\ 0 & 0 & 0 \end{pmatrix} \\ p-\ell & \end{matrix},$$

where  $A_{12}$  and  $B_{13}$  are non-singular upper triangular matrix.  $\ell$  is the rank of  $B$ ,  $k + \ell$  is the rank of  $[A^T \ B^T]^T$ .  $A_{23}$  is  $(m - k)$ -by- $\ell$  upper trapezoidal.

Note:  $R_A$  and  $R_B$  overwrite  $A$  and  $B$ , respectively.

**Step 2. QR decomposition of  $[A_{23}^T \ B_{13}^T]^T$ .**

- If  $m - k - \ell \geq 0$ :

$$\begin{matrix} & \ell \\ \ell & \begin{pmatrix} A_{23} \\ B_{23} \end{pmatrix} \\ \ell & \end{matrix} = \begin{matrix} & \ell \\ \ell & \begin{pmatrix} Q_1 \\ Q_2 \end{pmatrix} \\ \ell & \end{matrix} R_{23}$$

- If  $m - k - \ell < 0$ :

$$\begin{matrix} & \ell \\ m-k & \begin{pmatrix} A_{23} \\ B_{23} \end{pmatrix} \\ \ell & \end{matrix} = \begin{matrix} & \ell \\ m-k & \begin{pmatrix} Q_1 \\ Q_2 \end{pmatrix} \\ \ell & \end{matrix} R_{23}$$

Thus, (0.20) can be rewritten as:

$$A = U(Q_A \hat{R})Q^T, \quad B = V(Q_B \hat{R})Q^T \quad (0.21)$$

where

- If  $m - k - \ell \geq 0$ :

$$Q_A = \begin{matrix} & k & \ell \\ & & \\ k & \begin{pmatrix} I & 0 \\ 0 & Q_1 \end{pmatrix} \\ m-k-\ell & \end{matrix}, \quad Q_B = \begin{matrix} & k & \ell \\ & & \\ \ell & \begin{pmatrix} 0 & Q_2 \\ 0 & 0 \end{pmatrix} \\ p-\ell & \end{matrix}, \quad \hat{R} = \begin{matrix} & n-k-\ell & k & \ell \\ & & & \\ k & \begin{pmatrix} 0 & A_{12} & B_{13} \\ 0 & 0 & R_{23} \end{pmatrix} \\ \ell & \end{matrix}$$

- If  $m - k - \ell < 0$ :

$$Q_A = \begin{matrix} & k & \ell \\ & & \\ k & \begin{pmatrix} I & 0 \\ 0 & Q_1 \end{pmatrix} \\ m-k & \end{matrix}, \quad Q_B = \begin{matrix} & k & \ell \\ & & \\ \ell & \begin{pmatrix} 0 & Q_2 \\ 0 & 0 \end{pmatrix} \\ p-\ell & \end{matrix}, \quad \hat{R} = \begin{matrix} & n-k-\ell & k & \ell \\ & & & \\ k & \begin{pmatrix} 0 & A_{12} & B_{13} \\ 0 & 0 & R_{23} \end{pmatrix} \\ \ell & \end{matrix}$$

**Step 3. Safe Diagonalization of  $Q_1$  and  $Q_2$ .** Use the algorithm described in section 0.7, compute the CS decomposition:

$$Q_1 = U_1 \Sigma_1 Z_1^T, \quad Q_2 = V_1 \Sigma_2 Z_1^T \quad (0.22)$$

where

- If  $m - k - \ell \geq 0$ ,

$Q_1, Q_2$  are  $\ell$ -by- $\ell$ ,  $U_1, Z_1$  and  $V_1$  are  $\ell$ -by- $\ell$  orthogonal matrices, and  $\Sigma_1$  and  $\Sigma_2$  are  $\ell$ -by- $\ell$  diagonal matrices:

$$\Sigma_1 = \text{diag}(\alpha_1, \alpha_2, \dots, \alpha_\ell), \quad \Sigma_2 = \text{diag}(\beta_1, \beta_2, \dots, \beta_\ell)$$

where  $1 \geq \alpha_1 \geq \alpha_2 \geq \dots \geq \alpha_\ell \geq 0$ ,  $0 \leq \beta_1 \leq \beta_2 \leq \dots \leq \beta_\ell \leq 1$  and  $\Sigma_1^T \Sigma_1 + \Sigma_2^T \Sigma_2 = I_\ell$ .

- If  $m - k - \ell < 0$ ,

$Q_1$  is  $(m - k)$ -by- $\ell$ ,  $Q_2$  are  $\ell$ -by- $\ell$ ,  $U_1$  is  $(m - k)$ -by- $(m - k)$  orthogonal, and  $Z_1$  and  $V_1$  are  $\ell$ -by- $\ell$  orthogonal.  $\Sigma_1$  is  $(m - k)$ -by- $\ell$  diagonal and  $\Sigma_2$  is  $\ell$ -by- $\ell$  diagonal:

$$\Sigma_1 = \begin{matrix} m-k & \ell-m+k \\ \left( \begin{array}{cc} \Pi_1 & 0 \end{array} \right) \end{matrix}, \quad \Sigma_2 = \begin{matrix} m-k & \ell-m+k \\ \ell-m+k \left( \begin{array}{cc} \Pi_2 & 0 \\ 0 & I \end{array} \right) \end{matrix}$$

where  $\Pi_1 = \text{diag}(\alpha_1, \alpha_2, \dots, \alpha_{m-k})$ ,  $\Pi_2 = \text{diag}(\beta_1, \beta_2, \dots, \beta_{m-k})$ , and  $1 \geq \alpha_1 \geq \alpha_2 \geq \dots \geq \alpha_{m-k} \geq \alpha_{m-k+1} = \dots = \alpha_\ell = 0$  and  $0 \leq \beta_1 \leq \beta_2 \leq \dots \leq \beta_{m-k} \leq \beta_{m-k+1} = \dots = \beta_\ell = 1$ .  $\Sigma_1^T \Sigma_1 + \Sigma_2^T \Sigma_2 = I_\ell$ .

Combining (0.21) and (0.22), we have

$$A = U(\hat{U}C\hat{Q}^T)\hat{R}Q^T, \quad B = V(\hat{V}S\hat{Q}^T)\hat{R}Q^T \quad (0.23)$$

where

- If  $m - k - \ell \geq 0$ :

$$\hat{U} = \begin{matrix} k & \ell & m-k-\ell \\ \begin{matrix} k \\ \ell \\ m-k-\ell \end{matrix} \left( \begin{array}{ccc} I & 0 & 0 \\ 0 & U_1 & 0 \\ 0 & 0 & I \end{array} \right) \end{matrix}, \quad \hat{V} = \begin{matrix} \ell & p-\ell \\ p-\ell \left( \begin{array}{cc} V_1 & 0 \\ 0 & I \end{array} \right) \end{matrix}, \quad \hat{Q}^T = \begin{matrix} k & \ell \\ p-\ell \left( \begin{array}{cc} I & 0 \\ 0 & Z_1^T \end{array} \right) \end{matrix}$$

and

$$C = \begin{matrix} k & \ell \\ \begin{matrix} k \\ \ell \\ m-k-\ell \end{matrix} \left( \begin{array}{cc} I & 0 \\ 0 & \Sigma_1 \\ 0 & 0 \end{array} \right) \end{matrix}, \quad S = \begin{matrix} k & \ell \\ p-\ell \left( \begin{array}{cc} 0 & \Sigma_2 \\ 0 & 0 \end{array} \right) \end{matrix}$$

- If  $m - k - \ell < 0$ :

$$\hat{U} = \begin{matrix} k & m-k \\ \begin{matrix} k \\ m-k \end{matrix} \left( \begin{array}{cc} I & 0 \\ 0 & U_1 \end{array} \right) \end{matrix}, \quad \hat{V} = \begin{matrix} \ell & p-\ell \\ p-\ell \left( \begin{array}{cc} V_1 & 0 \\ 0 & I \end{array} \right) \end{matrix}, \quad \hat{Q}^T = \begin{matrix} k & \ell \\ p-\ell \left( \begin{array}{cc} I & 0 \\ 0 & Z_1^T \end{array} \right) \end{matrix}$$

and

$$C = \begin{matrix} k & \ell \\ \begin{matrix} k \\ m-k \end{matrix} \left( \begin{array}{cc} I & 0 \\ 0 & \Sigma_1 \end{array} \right) \end{matrix}, \quad S = \begin{matrix} k & \ell \\ p-\ell \left( \begin{array}{cc} 0 & \Sigma_2 \\ 0 & 0 \end{array} \right) \end{matrix}$$

**Step 4. Post-processing to form the desired GSVD (0.1).**

- $U := U\hat{U}$ .
- $V := V\hat{V}$ .
- Formulate  $R$  by RQ decomposition:  $\hat{Q}^T \hat{R} = RQ_3$
- $Q := QQ_3^T$
- $C$  and  $S$  defined in (0.23).

**0.7 GSVD of  $Q_1$  and  $Q_2$**

Why do we call it GSVD rather than safe diagonalization in the subsection title? typo? In this section, we detail algorithm of Step 2 of the GSVD algorithm for computing the safe diagonalization of  $Q_1$  and  $Q_2$ , where if  $m - k - \ell \geq 0$ ,

$$Q_1, Q_2 \in \mathbb{R}^{\ell \times \ell},$$

and if  $m - k - \ell < 0$ ,

$$Q_1 \in \mathbb{R}^{(m-k) \times \ell} \quad \text{and} \quad Q_2 \in \mathbb{R}^{\ell \times \ell}.$$

This is a special case of the 2-by-1 Cosine-sine decomposition. [33]

Step 2(1). Compute the SVD of  $Q_2$ :

$$Q_2 = V_1 \Sigma_2 Z_1^T, \tag{0.24}$$

where  $V_1$  is  $\ell$ -by- $\ell$  and  $Z_1$  is  $\ell$ -by- $\ell$ , both are orthogonal matrices.  $\Sigma_2 = \text{diag}(\beta_\ell, \dots, \beta_1)$  such that  $1 \geq \beta_\ell \geq \beta_{\ell-1} \geq \dots \geq \beta_1 \geq 0$ .

Step 2(2). Reverse the order of  $\beta_i$  in non-decreasing order:

- Reorder the diagonal entries of  $\Sigma_2$  non-decreasingly:  $\Sigma_2 = \text{diag}(\beta_1, \dots, \beta_\ell)$ ;
- Reverse the columns of  $V_1$ :  $V_1 := V_1(:, \ell : -1 : 1)$ ;
- Reverse the columns of  $Z_1$ :  $Z_1 := Z_1(:, \ell : -1 : 1)$ .

Step 2(3). Determine  $r$  such that  $0 \leq \beta_1 \leq \dots \leq \beta_r \leq \frac{1}{\sqrt{2}} < \beta_{r+1} \leq \dots \leq \beta_\ell \leq 1$ .

The justification for  $r$  see Remark 0.5.

Step 2(4). Matrix-matrix multiply:

$$T = Q_1 Z_1 \tag{0.25}$$

Step 2(5). QR decomposition of  $T$ :

$$T = U_1 R, \tag{0.26}$$

- If  $m - k - \ell \geq 0$ :  
since  $Q_1$  is  $\ell$ -by- $\ell$ ,  $U_1$  is an  $\ell$ -by- $\ell$  orthogonal matrix, and

$$R = \begin{matrix} & r & \ell - r \\ r & \Pi & \epsilon \\ \ell - r & 0 & R_{22} \end{matrix}, \quad (0.27)$$

where  $\Pi = \text{diag}(\alpha_1, \dots, \alpha_r)$ . The bottom  $(\ell - r)$  rows vanish if  $\ell - r \leq 0$ .  
For the explanation of why  $R_{22}$  may not be diagonal, see Remark 0.5.

- If  $m - k - \ell < 0$ :  
since  $Q_1$  is  $(m - k)$ -by- $\ell$ ,  $U_1$  is an  $(m - k)$ -by- $(m - k)$  orthogonal matrix, and

$$R = \begin{matrix} & r & m - k - r & \ell - m + k \\ r & \Pi & \epsilon & \epsilon \\ m - k - r & 0 & R_{22} & R_{23} \end{matrix}, \quad (0.28)$$

where  $\Pi = \text{diag}(\alpha_1, \dots, \alpha_r)$ . The bottom  $(m - k - r)$  rows vanish if  $m - k - r \leq 0$ .  
For the explanation of why  $\begin{bmatrix} R_{22} & R_{23} \end{bmatrix}$  may not be diagonal, see Remark 0.5.

Combining (0.25) and (0.26), we obtain:

$$Q_1 = U_1 R Z_1^T \quad (0.29)$$

As stated in (0.27) and (0.28), the last few rows of  $R$  might not exist. To be precise,

- If  $m - k - \ell \geq 0$  and  $\ell \leq r$ :  $R = \text{diag}(\alpha_1, \dots, \alpha_\ell)$ .
- If  $m - k - \ell < 0$  and  $m - k \leq r$ :

$$R = \begin{matrix} & m - k & \ell - m + k \\ m - k & \Pi & 0 \end{matrix}, \quad (0.30)$$

where  $\Pi = \text{diag}(\alpha_1, \dots, \alpha_{m-k})$ .

In either case,  $\Sigma_1 = R$ , the algorithm ends here since we already obtain (0.22).

Step 2(6). Refine  $R$  in (0.26) to diagonal if necessary:

- If  $m - k - \ell \geq 0$  and  $\ell - r > 0$ :  
– SVD of the block  $R_{22}$  of  $R$  in (0.27):

$$R_{22} = U_r C_r Z_r^T \quad (0.31)$$

where  $U_r$  is an  $(\ell - r)$ -by- $(\ell - r)$  orthogonal matrix,  $Z_r$  is an  $(\ell - r)$ -by- $(\ell - r)$  orthogonal matrix and  $C_r = \text{diag}(\alpha_{r+1}, \dots, \alpha_\ell)$ .

- Update the  $(r + 1)$ -th to  $\ell$ -th columns of  $U_1$ :

$$U_1 := U_1 \begin{matrix} & r & \ell - r \\ r & I & 0 \\ \ell - r & 0 & U_r \end{matrix}$$

- Update the last  $(\ell - r)$  columns of  $Z_1$ :

$$Z_1 := Z_1 \begin{matrix} & r & \ell - r \\ & \begin{pmatrix} I & 0 \\ 0 & Z_r \end{pmatrix} \end{matrix}$$

- Rewrite  $R$  to formulate  $\Sigma_1$ :

$$\Sigma_1 = \begin{matrix} & r & \ell - r \\ & \begin{pmatrix} \Pi & 0 \\ 0 & C_r \end{pmatrix} \end{matrix}$$

- If  $m - k - \ell < 0$  and  $m - k - r > 0$ :

- SVD of the block  $\begin{bmatrix} R_{22} & R_{23} \end{bmatrix}$  of  $R$  in (0.28):

$$\begin{bmatrix} R_{22} & R_{23} \end{bmatrix} = U_r C_r Z_r^T \quad (0.32)$$

where  $U_r$  is a  $(m - k - r)$ -by- $(m - k - r)$  orthogonal matrix,  $Z_r$  is an  $(\ell - r)$ -by- $(\ell - r)$  orthogonal matrix and  $C_r$  is a  $(m - k - r)$ -by- $(\ell - r)$  matrix with the main diagonal entries storing non-zero  $\alpha_{r+1}, \dots, \alpha_{m-k}$ .

- Update the  $(r + 1)$ -th to  $(m - k)$ -th columns of  $U_1$ :

$$U_1 := U_1 \begin{matrix} & r & m - k - r \\ & \begin{pmatrix} I & 0 \\ 0 & U_r \end{pmatrix} \end{matrix}$$

- Update the last  $(\ell - r)$  columns of  $Z_1$ :

$$Z_1 := Z_1 \begin{matrix} & r & \ell - r \\ & \begin{pmatrix} I & 0 \\ 0 & Z_r \end{pmatrix} \end{matrix}$$

- Rewrite  $R$  to formulate  $\Sigma_1$ :

$$\Sigma_1 = \begin{matrix} & r & \ell - r \\ & \begin{pmatrix} \Pi & 0 \\ 0 & C_r \end{pmatrix} \\ m - k - r \end{matrix}$$

Thus, the final decomposition of  $Q_1$  is the following:

$$Q_1 = U_1 \Sigma_1 Z_1^T \quad (0.33)$$

Step 2(7). If  $R$  is refined in Step 2(6), i.e.,  $Z_1$  is modified, we need to update  $V$  as well:

- Set  $W$ :  $W = \text{diag}(\beta_{r+1}, \dots, \beta_\ell) Z_r$
- QR decomposition of  $W$ :

$$W = Q_w R_w \quad (0.34)$$

- Update last  $(\ell - r)$  columns of  $V$ :

$$V := V \begin{matrix} & r & \ell - r \\ r & \begin{pmatrix} I & 0 \\ 0 & Q_w \end{pmatrix} \\ \ell - r & \end{matrix}$$

**Remark 0.5.** This algorithm extends the one proposed by Van Loan [38].

First, we briefly justify here why we choose  $\frac{1}{\sqrt{2}}$  as the threshold. Since  $Q_1^T Q_1 + Q_2^T Q_2 = I$  and  $\|Q_1^T Q_1 + Q_2^T Q_2\|_1 = 1$ , the singular values of  $Q_1$  and  $Q_2$  lie between 0 and 1.  $\frac{1}{\sqrt{2}}$  is the exact midpoint in between ( $(\frac{1}{\sqrt{2}})^2 + (\frac{1}{\sqrt{2}})^2 = 1$ ) so that it nicely separates large singular values from tiny singular values. This theoretical yet empirical choice is first suggested by Van Loan in [38] because the midpoint balances the backward error defined in (1.2) - (1.6) and performance.

We then explain why  $R$  may not be diagonal in finite precision arithmetic.

In Step 2(4), in exact arithmetic, it follows that:

$$\begin{aligned} T^T T &= (Q_1 Z_1)^T Q_1 Z_1 \\ &= Z_1^T Q_1^T Q_1 Z_1 \\ &= Z_1^T (I - Q_2^T Q_2) Z_1 \\ &= Z_1^T (I - Z_1 \Sigma_2^T V_1^T V_1 \Sigma_2 Z^T) Z_1 \\ &= Z_1^T (I - Z_1 \Sigma_2^T \Sigma_2 Z_1^T) Z_1 \\ &= I - \Sigma_2^T \Sigma_2 \\ &= \text{diag}(1 - \beta_1^2, 1 - \beta_2^2, \dots, 1 - \beta_\ell^2) \end{aligned} \tag{0.35}$$

Therefore,  $QR$  of  $T$  leads to diagonal  $R$ , the reasoning is the follows: by Step 2(5)  $T^T T = (U_1 R)^T U_1 R = R^T R$  is diagonal by (0.35) and  $R$  is upper triangular, thus  $R$  must be diagonal.

But in numerical computation, we cannot ignore the upper-diagonal entries of  $R_{22}$  and  $\begin{bmatrix} R_{22} & R_{23} \end{bmatrix}$  in (0.27) and (0.28) respectively.

Before we proceed, we present the following theorem and its proof.

**Theorem 0.6.** *Let  $X$  be an  $m$ -by- $n$  matrix whose rank =  $\min(m, n)$  and*

$$X^T X = D^T D + E,$$

*where  $\|E\| = O(\epsilon)$  and  $D = \text{diag}(\|x_1\|, \|x_2\|, \dots, \|x_n\|)$  and  $x_i$  denotes the  $i$ -th column of  $X$ .*

*Applying the full QR decomposition of  $X$ , one obtains:*

$$X = QR, \tag{0.36}$$

*where  $Q$  is an  $m$ -by- $m$  orthogonal matrix.  $R$  is an  $m$ -by- $n$  upper triangular matrix if  $m \geq n$ . Otherwise,  $R$  is upper trapezoidal.*

*Let  $X_i$  be the first  $i$  columns of  $X$ , then for  $i, j$  where  $i < j \leq \min(m, n)$ , we have*

$$|R(i, j)| \leq \min\{\|x_j\|, \frac{\|E\|}{\sigma_{\min}(X_i)}\}. \tag{0.37}$$

*Proof.* This theorem is proved by Van Loan in **Theorem 3.2** [38, pp. 484–485]. □

From this theorem along with the  $\frac{1}{\sqrt{2}}$  threshold we justified above, it follows that, for  $i = 1, 2, \dots, r$ ,

$$|R(i, j)| \leq \frac{\epsilon}{\sigma_{\min}(T_i)} \leq \frac{\epsilon}{\sqrt{1 - \beta_r^2}} \leq \sqrt{2}\epsilon.$$

By that, we know that the upper-diagonal block matrices of the first  $r$  rows of  $R$  in both (0.27) and (0.28) are effectively zero matrices.

On the contrary, we have:

$$|R_{22}| \simeq \sqrt{1 - \beta_{r+1}} < \frac{1}{\sqrt{2}}, \quad (0.38)$$

$$| \begin{bmatrix} R_{22} & R_{23} \end{bmatrix} | \simeq \sqrt{1 - \beta_{r+1}} < \frac{1}{\sqrt{2}} \quad (0.39)$$

for (0.27) and (0.28), respectively, both of which suggest that these two submatrices are not well-conditioned and hence their upper-diagonal entries cannot be neglected.

## 0.8 Remarks

**Remark 0.7.** Michael Stewart [32] describes an alternative rank revealing mechanism of  $[A; B]$  and claims that it can more reliably determine the partitioning of a GSVD and shows improved numerical reliability.

**Remark 0.8.** The algorithm presented here is “similar to” that introduced by Golub and Van Loan [16, pp. 502–503] to compute GSVD using CS decomposition for tall, full-rank matrix pairs.

Assume that  $A$  is  $m$ -by- $n$  and  $B$  is  $p$ -by- $n$  with  $m \geq n$  and  $p \geq n$ , computes an  $m$ -by- $m$  orthogonal matrix  $U$ , a  $p$ -by- $p$  orthogonal matrix  $V$ , an  $n$ -by- $n$  nonsingular matrix  $X$  and  $m$ -by- $n$  diagonal matrix  $C$ ,  $p$ -by- $n$  diagonal matrix  $S$  such that  $U^T A X = C$  and  $V^T B X = S$ .

*Step 1* Compute the regular QR decomposition of  $\begin{pmatrix} A \\ B \end{pmatrix}$ :  $\begin{pmatrix} A \\ B \end{pmatrix} = \begin{pmatrix} Q_1 \\ Q_2 \end{pmatrix} R$

*Step 2* Compute the CS decomposition of  $Q_1$  and  $Q_2$ :

$$U^T Q_1 Z = C = \text{diag}(\alpha_1, \dots, \alpha_n), \quad V^T Q_2 Z = S = \text{diag}(\beta_1, \dots, \beta_n).$$

*Step 3* Solve  $RX = Z$  for  $X$ .

**Remark 0.9.** LAPACK GSVD algorithm [2, pp. 51–53] has two phases. First is a pre-processing step as described in Section 0.6. Next is a **Jacobi-style method** [26, 4] to compute the GSVD of two square upper triangular matrices, namely,  $A_{23}$  and  $B_{13}$  in (0.20) such that

$$A_{23} = U_1 C R Q_1^T, \quad B_{13} = V_1 S R Q_1^T. \quad (0.40)$$

Here  $U_1$ ,  $V_1$  and  $Q_1$  are orthogonal matrices,  $C$  and  $S$  are both real nonnegative matrices satisfying  $C^T C + S^T S = I$ ,  $S$  is nonsingular, and  $R$  is upper triangular and nonsingular.



# 1 Software

This section starts with a justification of why we choose Julia to implement the GSVD algorithm described in section 0.6. Then we discuss the design of the interface, and implementation details. Finally, we list the GSVD available in other programming languages.

## 1.1 Why Julia

**Two-language problem.** Julia is a new programming language that aims to address the so-called “two-language problem”, which designers and software developers need to face: they have to make a trade-off when designing or choosing a language – it can either be relatively easy for humans to write as a dynamic language, or relatively easy for computers to run as a statically-typed language, but not both [27].

When it comes to the realm of numerical and scientific computing, the first language Fortran [25] dated back to 1957. Over six decades, on one hand, the landscape of computing has changed dramatically due to the advance of hardware, algorithms, and tremendous increased amount of data, among others. On the other hand, an unfortunate outcome is that the most challenging areas of scientific computing have benefited the least from the enhanced abstraction and productivity offered by higher-level languages. Modern scientific computing environments such as Python (Numpy) [36], R [20], Mathematica [42], and MATLAB [24], to name a few, have grown in popularity among researchers and scientists of various disciplines, all of which are weakly or dynamically typed. Still, when it comes to performance, none of these favored languages can compete against Fortran and C for solving computationally intensive problems because weakly or dynamically typed language needs type check for variables, statement, field, and even object and class hierarchy at runtime [7].

**Julia – introduction.** As a dynamic language, Julia allows programmers to write high-level, generic and abstract code that closely resembles mathematical formulas [7, 14]. By making use of the following techniques, Julia can generate optimized native code for multiple architectures and can approach the speed of Fortran and C. [31]

- Multiple dispatch using type system to select implementations:  
“Multiple” means that a function can have multiple implementations, called methods, distinguished by the type annotations added to parameters of the function. “Dispatch” means that at run-time, a function call is dispatched to the most specific method applicable to the types of the arguments.
- Code specialization against run-time types
- JIT (Just-In-Time) compilation using the LLVM compiler framework

**Julia – features that we’re particularly interested in.**

### 1. A powerful approach to linear algebra.

Vectors and matrices in Julia are based on and can be extended from the `Array` type. While in C++ and Python matrices are stored in a row-first manner, they are stored in column-major in Julia, like Fortran. Elementary operations such as addition, subtraction, and multiplication are built-in in Julia. Julia also provides native implementations for many common linear

algebra operations in the `LinearAlgebra` library. Basic matrix operations are implemented with calls to BLAS and LAPACK routines; the default implementation is OpenBLAS.

- Elementary operations:
  - Julia features component-wise operations. For example, component-wise multiplication writes as `.*`. If `f` is a function defined for single arguments and `A` is a matrix, then `f.(A)` is the matrix that results from component-wise operation of `f` to the entries of `A`. This is also known as broadcasting.
  - Julia also offers matrix division from left `B\A` and right `A/B`, which is equivalent to computing  $B^{-1}A$  and  $AB^{-1}$  respectively.
- LAPACK wrappers are implemented fully in Julia code, using `ccall` — calling C and Fortran code, without any “glue” code, code generation, or compilation. All LAPACK routines are available in Julia. Julia users can write user-extensible wrappers for BLAS and LAPACK on top of the `LinearAlgebra` library [7]. For instance, LAPACK provides BLAS-1 routine `ddot` for computing the dot product of two vectors of double precision floating point type. This is already implemented in Julia as one method under the `*` operation family. Still, we can write `ddot` wrapper purely in Julia. Here’s how it is implemented.

```
function compute_dot(DX::Vector{Float64}, DY::Vector{Float64})
    @assert length(DX) == length(DY)
    n = length(DX)
    incx = incy = 1
    product = ccall((:ddot_, "libLAPACK"),
                    Float64,
                    (Ref{Int32}, Ptr{Float64}, Ref{Int32},
                     Ptr{Float64}, Ref{Int32}),
                    n, DX, incx, DY, incy)
    return product
end
```

We declare the function with two arguments of `Vector{Float64}` type and name it `compute_dot`. In this function, we first declare input arguments necessary to `ddot`. Then we use `ccall` to call LAPACK’s `ddot` routine. `ccall` has four arguments:

- `(:function_name, "library_name")` pair: in our example, `(:ddot_, "libLAPACK")`.
- return type of the function: `Float64` in our example.
- a tuple of input types to the function in the library: in our example, `(Ref{Int32}, Ptr{Float64}, Ref{Int32}, Ptr{Float64}, Ref{Int32})`.
- the actual argument values to be passed to the function: `n, DX, incx, DY, incy` respectively in our example.

Finally, we take the return value of `ccall` as the return for the wrapper function.

- Matrices with special symmetries and structures arise often in linear algebra and are frequently associated with various matrix factorizations. Julia has a collection of special matrix types. Table 1 below summarizes the types of some special matrices that have been implemented in Julia.

Type	Description
Symmetric	Symmetric matrix
Hermitian	Hermitian matrix
UpperTriangular	Upper triangular matrix
LowerTriangular	Lower triangular matrix
UpperHessenberg	Upper Hessenberg matrix
Tridiagonal	Tridiagonal matrix
SymTridiagonal	Symmetric tridiagonal matrix
Bidiagonal	Upper/lower bidiagonal matrix
Diagonal	Diagonal matrix

Table 1: Commonly used special matrix types in Julia

Furthermore, Julia provides fast computation with specialized routines that are specially developed for particular matrix types. For example, for the SVD of a general dense matrix, Julia will use the default SVD method `svd!(A::StridedMatrix{T})`, which essentially calls `gesvd!` in LAPACK. The key part of the source code of this function is shown below.

```
function svd!(A::StridedMatrix{T}; full::Bool = false,
alg::Algorithm = default_svd_alg(A)) where T<:BlasFloat
    m,n = size(A)
    if m == 0 || n == 0
        u,s,vt = (Matrix{T}(I, m, full ? m : n), real(zeros(T,0)),
            Matrix{T}(I, n, n))
    else
        u,s,vt = _svd!(A,full,alg)
    end
    SVD(u,s,vt)
end

_svd!(A::StridedMatrix{T}, full::Bool, alg::DivideAndConquer)
where T<:BlasFloat = LAPACK.gesdd!(full ? 'A' : 'S', A)
function _svd!(A::StridedMatrix{T}, full::Bool, alg::QRIteration)
where T<:BlasFloat
    c = full ? 'A' : 'S'
    u,s,vt = LAPACK.gesvd!(c, c, A)
end
```

On the contrary, if we want to compute the SVD of a bidiagonal matrix, Julia will use an optimized `svd!(M::Bidiagonal{<:BlasReal})` instead. This function is a wrapper on top of LAPACK's `bdsdc!` routine. We also show the key part of the source code of this function.

```
svdvals!(M::Bidiagonal{<:BlasReal}) = LAPACK.bdsdc!(M.uplo,
```

```
'N', M.dv, M.ev)[1]
```

```
function svd!(M::Bidiagonal{<:BlasReal}; full::Bool = false)
    d, e, U, Vt, Q, iQ = LAPACK.bdsdc!(M.uplo, 'I', M.dv, M.ev)
    SVD(U, d, Vt)
end
```

- Julia has support for sparse vectors and sparse matrices in the `SparseArrays` module. In Julia, sparse matrices are stored in the Compressed Sparse Column (CSC) format. The CSC is represented as follows:

```
struct SparseMatrixCSC{Tv,Ti<:Integer} <: AbstractSparseMatrix{Tv,Ti}
    m::Int                # Number of rows
    n::Int                # Number of columns
    colptr::Vector{Ti}    # Column j is in colptr[j):(colptr[j+1]-1)
    rowval::Vector{Ti}    # Row indices of stored values
    nzval::Vector{Tv}     # Stored values, typically nonzeros
end
```

Moreover, sparse matrix solvers call functions from SuiteSparse [11]. Choleskey factorization, LU factorization and QR factorization of sparse matrices are now available.

## 2. Ease-of-use.

- Julia is dynamically-typed, which means users need not worry about which types fit best in a given scenario.
- Julia features garbage collection, thus memory management (allocation and destruction) will not be a manual effort, unlike in Fortran when we need to consider workspace requirement.
- Generic, abstract, and extendible syntax by multiple dispatch

In Fortran, multiplication between two scalars, scalar and vector, matrix and vector, and two matrices have different method names. This is counter-intuitive to the abstraction of math: all of this operations are multiplication, why can't they share the same name? Fortunately, in Julia, with the help of multiple dispatch, which allows users to depart from the limitations of type-based designs, we can leverage the expressive power of generic functions. For instance, elementary operations like addition and multiplication in Julia are intuitive and generic, as simple as math, shown below in Table 2.

Operator	Function	Operand
+	add numbers/vectors/matrices	Int, Float, Vector, Abstract Matrix, Dense Matrix, ...
*	multiply scale or compose	Number, Function

Table 2: Generic commands of addition and multiplication of different types in Julia

Further, with multiple dispatch, users can extend the SVD family by implementing the SVD of other types, tensor, for instance. These new methods will have separate implementation, but share the same method name `svd`.

- Julia is interactive with a full-featured interactive command-line REPL (read-eval-print loop) built into the Julia executable. It resembles to the interactive programming environment Python(IPython) and R(RStudio). In 2015, a web-based interactive computational platform Jupyter (named after Julia, Python and R) that spun off from IPython was launched.
- Syntax similarity to MATLAB:
  - Both Julia and MATLAB use `end` keyword to indicate code block structure, e.g. loops, functions.
  - In linear algebra, MATLAB programmers may find Julia more friendly because the array index in Julia also starts at 1, not 0.
  - Table 3 below summarizes a list of Julia commands for frequently used matrix factorizations, and their counterparts in MATLAB.

Julia command	MATLAB command	Description
<code>schur(A::StridedMatrix) -&gt; F::Schur</code>	<code>T = schur(A)</code>	Schur decomposition
<code>lu(A, pivot=Val(true); check = true) -&gt; F::LU</code>	<code>[L,U] = lu(A)</code>	LU decomposition
<code>qr(A, pivot=Val(false)) -&gt; F</code>	<code>[Q,R] = qr(A)</code>	QR decomposition
<code>eigen(A; permute::Bool=true, scale::Bool=true, sortby) -&gt; Eigen</code>	<code>[V,D] = eig(A)</code>	Eigenvalues and eigenvectors
<code>svd(A; full::Bool = false) -&gt; SVD</code>	<code>[U,S,V] = svd(A)</code>	Singular value decomposition

Table 3: Interfaces of commonly used matrix decompositions in Julia and MATLAB

### 3. Speed.

The core design decision, type inference through method specialization via multiple dispatch is what allows Julia to be very easy for a compiler to make into efficient code.

- Method specialization. In short, method specialization is to run the right code at the right time. Julia’s compilation strategy is built on runtime type information. Every time a method is called with a new tuple of argument types, it is specialized to these types. Optimizing methods at invocation time, rather than ahead of time, provides the JIT with key pieces of information: the memory layout of all arguments is known, allowing for unboxing of objects and direct field access. Specialization, in turn, allows for devirtualization. Devirtualization replaces method dispatch with direct calls to a specialized method. This reduces dispatch overhead and enables inlining. The specialized method is added to the function’s dispatch table so that future calls with the same combination of argument types can reuse the generated code [6].
- Type inference. Types of program expressions and variables are inferred by forward data-flow analysis [8]. Julia uses a set constraint-based analysis with constraints arising from return values, method dereferences, and argument types. Type requirements need to be satisfied at function call sites and field assignments. The system propagates constraints forward to satisfy requirements, inferring the types for intermediate values along the way.

## 1.2 Interfaces of GSVD in Julia

The outputs of the GSVD algorithm consist of six matrices and two integers. To follow Julia’s convention as an object-oriented language, we encapsulate all the outputs into a immutable composite type defined by `struct` keyword, followed by a block of field names, optionally annotated with types using the `::` operator. In addition, `<:` keyword connects a type with its supertype (“parent”) on the right hand side. With composite type, users do not need to explicitly enumerate every matrix or integer in the return statement when they only want to access part of the outputs. We name the composite type `GeneralizedSVD`.

```
struct GeneralizedSVD{T} <: Factorization{T}
    U::AbstractMatrix{T}
    V::AbstractMatrix{T}
    Q::AbstractMatrix{T}
    C::AbstractMatrix{T}
    S::AbstractMatrix{T}
    k::Int
    l::Int
    R::AbstractMatrix{T}
end
```

Interfaces in Julia, particularly in “`LinearAlgebra`” library, are two-fold: one makes a copy of the input matrices or vectors, which is convenient for testing purposes and is suitable for small inputs; the other one overwrites the inputs to save space and reduce communication overhead between memory and cache.

When designing the interface of the GSVD, we also take multiple dispatch into consideration. Multiple dispatch allows a function to be written generically and thus maintains the language’s expressiveness. It enables SVD of one matrix and GSVD of a matrix pair to share a single interface name, but with different number or types of arguments.

Therefore, we provide two interfaces for the GSVD.

```
svd(A, B) -> GeneralizedSVD
```

or

```
svd!(A, B) -> GeneralizedSVD
```

**Common features of two interfaces.** Both interfaces compute the generalized SVD of  $A$  and  $B$ , returning a `GeneralizedSVD` object  $F$ , such that

$$A = F.U*F.C*F.R*F.Q' \quad \text{and} \quad B = F.V*F.S*F.R*F.Q'.$$

For an  $m$ -by- $n$  matrix  $A$  and  $p$ -by- $n$  matrix  $B$ ,

- $U$  is an  $m$ -by- $m$  orthogonal matrix,
- $V$  is a  $p$ -by- $p$  orthogonal matrix,
- $Q$  is an  $n$ -by- $n$  orthogonal matrix,

- $C$  is an  $m$ -by- $(k+1)$  diagonal matrix with 1s in the first  $k$  entries,
- $S$  is a  $p$ -by- $(k+1)$  matrix whose top right 1-by-1 block is diagonal,
- $R$  is a  $(k+1)$ -by- $n$  matrix whose rightmost  $(k+1)$ -by- $(k+1)$  block is nonsingular upper block triangular,
- $k+1$  is the effective numerical rank of the matrix  $[A; B]$ . One can get  $k+1$  from  $F.k + F.1$

**Differences of two interfaces.** After calling `svd(A, B)`,  $A$  and  $B$  will remain the same. By contrast, after calling `svd!(A, B)`,  $A$  and  $B$  will be overwritten. Specifically, let  $R$  be the  $R$  matrix of the return object  $F$ ,

1. If  $m - k - \ell \geq 0$

$$R = \begin{matrix} & n-k-\ell & k+\ell \\ k+\ell & \left( \begin{array}{cc} 0 & R_0 \end{array} \right) \end{matrix}$$

$R_0$  is the same as  $A[1:k+1, n-k-1+1:n]$ .

2. If  $m - k - \ell < 0$

$$R = \begin{matrix} & n-k-\ell & k & m-k & k+\ell-m \\ & k & & & \\ m-k & \left( \begin{array}{cccc} 0 & R_{11} & R_{12} & R_{13} \\ 0 & 0 & R_{22} & A_{23} \\ 0 & 0 & 0 & R_{33} \end{array} \right) \\ k+\ell-m & & & & \end{matrix}$$

$\begin{pmatrix} R_{11} & R_{12} & R_{13} \\ 0 & R_{22} & R_{23} \end{pmatrix}$  is the same as  $A[1:m, n-k-1+1:n]$  and  $R_{33}$  is equal to  $B[m-k+1:1, n+m-k-1+1:n]$ .

### 1.3 GSVD module and implementation details

1. The structural unit called **module** is native in Julia to group relevant functions and definitions. We create two modules for the GSVD algorithm.
2. The main module of the GSVD algorithm is **GSVD**. In **GSVD** module, we first declare the composite type **GeneralizedSVD**, create its constructor, and a function for neat display. This module also contains three functions below:
  - **svd**: main routine,
  - **preproc**: pre-processing subroutine,
  - **householderqr**: Householder QR decomposition subroutine.
3. In addition, considering that the safe diagonalization not only serves as a building block for the GSVD algorithm, but is also a tool in other applications, we have separated the safe diagonalization as a stand alone module called **SafeDiag**. It contains routine **safeDiag**.
4. The GSVD algorithm starts from the main function **svd** under module **GSVD**. It then calls **preproc**. Once return, it calls function **safeDiag** intermodularly. Finally, upon return, the main function post-processes to formulate the outputs. Figure 1 shows the sequence of the major function calls.

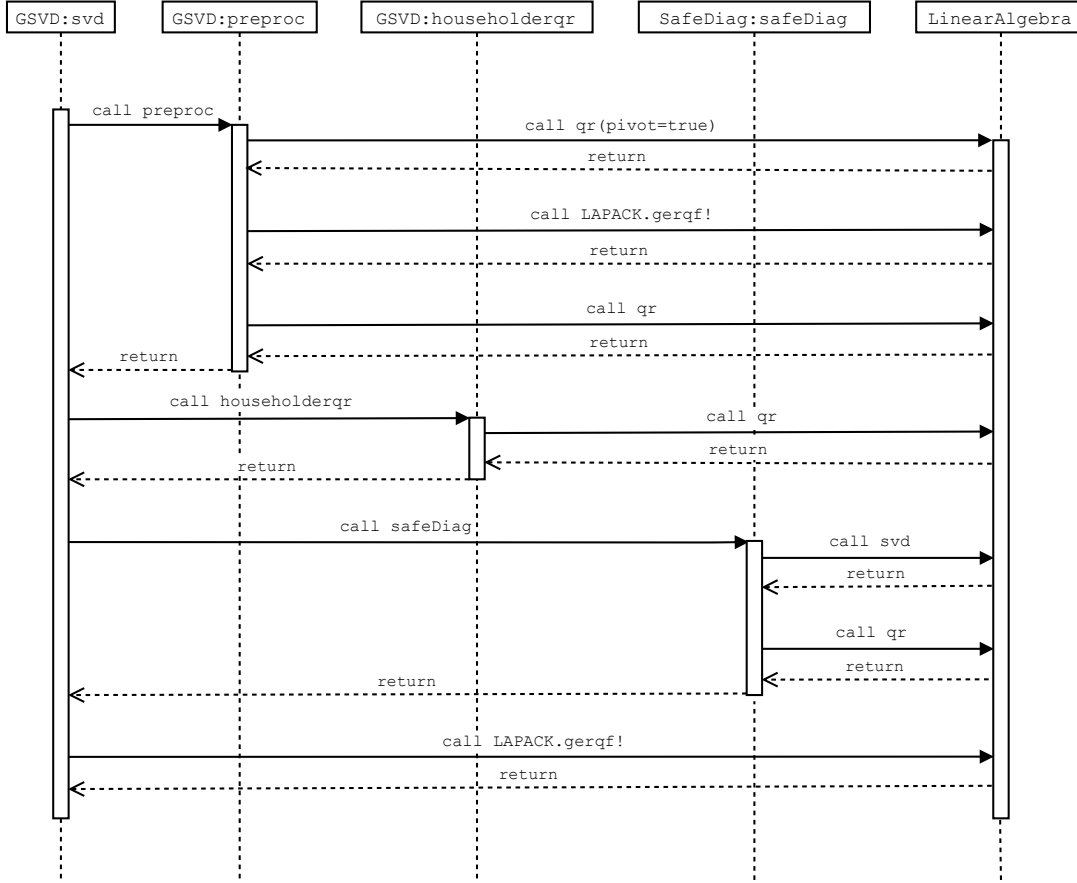


Figure 1: Sequence diagram for the GSVD

5. The details of each major functions are discussed below.

- **GSVD:preproc**

This step is to reduce two input matrices  $A$  and  $B$  into two upper triangular forms. This is done via a call to **preproc**. This function makes use of three fundamental orthogonal decompositions.

- First is QR decomposition with column pivoting to reveal the numerical rank of  $B$  and  $[A; B]$  without forming the matrix explicitly. This is done by a call to **qr**( $A$ , **pivot=Val(true)**). Let  $tol_B$  as the tolerance to determine the effective rank of  $B = \ell$ .

$$tol_B = \max\{p, n\} \|B\|_1 \epsilon$$

where  $\epsilon$  is the machine precision of **Float64**.

We also use QR decomposition with column pivoting again on the leftmost  $n - \ell$



columns of  $A$ . Similarly, by defining

$$tol_A = \max\{m, n\} \|A\|_1 \epsilon$$

we compute the effective rank of  $[A; B] = k + \ell$ .

- (b) Second is RQ decomposition of the top  $\ell$  rows of  $B$  if  $n > \ell$  via a call to `LAPACK.gerqf!`. It is called a second time on  $A$  if  $n - \ell > k$ .
- (c) Third is QR decomposition of  $A$  when  $m > k$  by calling `qr`.

Upon return to `svd`, two of the upper triangular matrices overwrites  $A$  and  $B$ , the orthogonal matrices are placed in  $U$ ,  $V$ , and  $Q$  and rank information is stored in  $k$  and  $\ell$ .

- **GSVD:householderqr**

This step is to reduce two upper triangular matrices to one and is done by calling `householderqr`. On entry, two triangular matrices are stacked together and passed as the arguments of `qr`. On exit,  $Q_1$  and  $Q_2$  overwrites inputs.

- **SafeDiag:safediag**

This step calls `safeDiag` from module `SafeDiag`. This function requires SVD, and QR decomposition. This is done by calls to `svd`, `qr` respectively.

- (a) We first compute SVD of  $Q_2$ . To preserve the order of  $\{\cos \theta\}$ , we have to reverse the order of the singular values of  $Q_2$ .
- (b) Since  $\{\cos \theta\}$  are already sorted, we take advantage of binary search to find the threshold  $r$ .
- (c) QR decomposition of the multiply of  $Q_1$  and right singular vectors of  $Q_2$ .  $R$  is not only triangular but diagonal. However, sanitization is necessary to assure the non-negativity of the diagonal entries.

It return  $U_1, V_1, Z_1, C, S$  on exit.

- Post-processing:

In this step, we update matrix  $U$ ,  $V$  and  $Q$  by matrix-matrix multiply. To formulate  $R$ , we utilize RQ decomposition via a call to `LAPACK.gerqf!`. Finally, we put matrices  $U, V, C, S, Q$  and  $k, \ell$  into the constructor of `GeneralizedSVD` as return.

6. We implement the GSVD algorithm in Julia 1.3 using `Float64` data. Our current implementation is able to be extended for `Float32` type in the future.

## 1.4 Lessons and caveats

We capture some pitfalls that might be a bottleneck, and thus are worth mentioning.

- **Some inconsistent interface design.**

As a modern programming language, Julia supports multiple dispatch, that is, parametric polymorphism. A good example is that the GSVD interface reuses the name of the SVD of a single matrix  $A$  by taking two matrices as arguments. Unfortunately, this principle is not fully implemented. For instance, when computing the matrix norm of `[1 2 3; 4 5 6; 7 8 9]`, one may intuitively call `norm([1 2 3; 4 5 6; 7 8 9])`. However, this will not produce

the desired result as shown below. What it actually computes is the 2-norm of the vector formed by iterating every entry of this matrix. Equivalently, it computes `norm([1 2 3 4 5 6 7 8 9])`. To get the proper matrix norm, instead, one may want to use `opnorm([1 2 3; 4 5 6; 7 8 9])`.

Julia command/result	MATLAB command/result
<code>norm([1 2 3; 4 5 6; 7 8 9])</code> 16.881943016134134	<code>norm([1 2 3;4 5 6;7 8 9])</code> 16.8481
<code>opnorm([1 2 3; 4 5 6; 7 8 9])</code> 16.84810335261421	N/A

- Performance relies on “proper” implementation.

Our GSVD algorithm involves many operations with subarrays and submatrices. Naturally, we would use `[start_i:end_i,start_j:end_j]` to get a block matrix (slicing) from the entire matrix. However, in Julia, this could be a significant performance issue since slicing an array/matrix creates a copy of the selected subarray/submatrix, which is computational expensive. For instance, if we want to get the sum of subarray, we might do the following in Julia:

```
x = rand(10^6);
fcopy(x) = sum(x[2:end-1]);
@time fcopy(x);
0.003051 seconds (7 allocations: 7.630 MB)
```

An alternative is to create a “view” of the array, which is an array object that actually references the data of the original array in-place, without making a copy. This can be done for individual slices by calling function `view`, or more simply for a whole expression or block of code by putting macro `@views` in front of that expression. This could improve the performance significantly. In the following code, we rewrite the function to calculate the sum of suarray with macro `@views`. One can tell that by doing so, the speedup is three-fold.

```
x = rand(10^6);
@views fview(x) = sum(x[2:end-1]);
@time fview(x);
0.001020 seconds (6 allocations: 224 bytes)
```

Many other features of Julian have not been studied and explored by the author, most notably, parallelism. This could be a good direction for future work.

## 1.5 GSVD in other languages

A number of numerical computing platforms feature the GSVD is listed in the following table.

Language	GSVD Documentation
Native Julia (proposed)	<code>svd(A, B) -&gt; GeneralizedSVD</code> Computes the generalized SVD of A and B, returning a <code>GeneralizedSVD</code> factorization object F, such that $A = F.U*F.C*F.R*F.Q'$ and $B = F.V*F.S*F.R*F.Q'$ .
Julia 1.3 (LAPACK wrapper)	<code>svd(A, B) -&gt; GeneralizedSVD</code> Computes the generalized SVD of A and B, returning a <code>GeneralizedSVD</code> factorization object F, such that $A = F.U*F.D1*F.R*F.Q'$ and $B = F.V*F.D2*F.R*F.Q'$ .
MATLAB (2019b)	<code>[U,V,X,C,S] = gsvd(A,B)</code> Returns unitary matrices U and V, a (usually) square matrix X, and nonnegative diagonal matrices C and S so that $A = U*C*X'$ , $B = V*S*X'$ , $C'*C + S'*S = I$ .
Mathematica	<code>SingularValueDecomposition[m,a]</code> Gives a list of matrices $\{u, ua, w, wa, v\}$ such that m can be written as $u.w.Conjugate[Transpose[v]]$ and a can be written as $ua.wa.Conjugate[Transpose[v]]$ .
R (geigen v2.3, LAPACK wrapper)	<code>z &lt;- gsvd(A, B)</code> Computes The Generalized Singular Value Decomposition of matrices A and B such that $A = U D_1 [0 \ R] Q^T$ and $B = V D_2 [0 \ R] Q^T$ . Note that the return value is the same as the output of LAPACK 3.6 and above.
Python (R. Luo's thesis)	Didn't disclose API design. The author defined GSVD as follows: Given two $M_i$ -by- $N$ column-matched but row-independent matrices $D_i$ , each with full column rank and $N \leq M_i$ , the GSVD is an exact simultaneous factorization $D_i = U_i \Sigma_i V^T$ , $i = 1, 2$ . $U_i$ is $M_i$ -by- $N$ and are column-wise orthonormal and $V$ is $N$ -by- $N$ nonsingular matrix with normalized rows. $diag(\Sigma_i)$ returns two lists of $N$ positive values and the ratios are called the generalized singular values.

## 1.6 Accuracy (backward stability)

**Metric.** We define the following metrics to test backward stability:

$$res_A = \frac{\|U^T A Q - C R\|_1}{\max(m, n) \|A\|_1 \epsilon} \quad (1.1)$$

$$res_b = \frac{\|V^T B Q - S R\|_1}{\max(p, n) \|B\|_1 \epsilon} \quad (1.2)$$

$$orth_{CS} = \frac{\|C^T C + S^T S - I\|_1}{\max(m, n, p) \epsilon} \quad (1.3)$$

$$orth_U = \frac{\|U^T U - I\|_1}{m \epsilon} \quad (1.4)$$

$$orth_V = \frac{\|V^T V - I\|_1}{p \epsilon} \quad (1.5)$$

$$orth_Q = \frac{\|Q^T Q - I\|_1}{n \epsilon} \quad (1.6)$$

where  $\epsilon$  is the machine precision of input data type.

### 1.6.1 Numerical examples of small matrices

For the four examples we illustrated in 0.5, we also compute the stability metrics by “JuliaGSVD” and Julia 1.3 in Table 4.

	Version	$res_A$	$res_B$	$orth_{CS}$	$orth_U$	$orth_V$	$orth_Q$
Example 1	JuliaGSVD	0.2956	0.7057	0.4	0.5308	1.0417	1.1790
	Julia 1.3	0.3599	0.5714	0.025	0.9117	1.7083	1.3250
Example 2	JuliaGSVD	0.6173	0.4098	0.625	1.5000	0.5613	1.3998
	Julia 1.3	0.5068	0.5689	0.125	1.4583	0.9245	1.2483
Example 3	JuliaGSVD	0.4181	0.8941	0.375	0.7500	1.3940	1.3277
	Julia 1.3	0.3536	0.5938	0.125	1.4791	1.9540	1.1062
Example 4	JuliaGSVD	0.3600	0.5900	0.9	0.6558	0.5385	1.4362
	Julia 1.3	0.4449	0.3056	0.0	1.3225	0.7205	1.1814

Table 4: Stability profiling for small matrices

### 1.6.2 Random dense matrices

**Test matrix generation.** As discussed in Section 0.1, we test stability on four cases depending on the row and column size of the input matrix pair. In this section, we test random dense matrices of `Float64`. For each case, we choose four subcases from low to high matrix size. We generate a total of 320 random matrix pairs, 20 for each subcase.

**Results.** As a demonstration, we list the results of six stability metrics for each subcase of a single test run in Table 5. All 320 test runs yield results no greater than 1.5.

	$m$	$p$	$n$	$k + l$	$res_A$	$res_B$	$orth_{CS}$	$orth_U$	$orth_V$	$orth_Q$
$m \geq n$ $p \geq n$	60	50	40	40	0.1140	0.2331	0.2083	0.7117	1.0742	0.4980
	300	250	200	200	0.0372	0.0633	0.0267	0.4815	0.6189	0.3214
	900	750	600	600	0.0151	0.0238	0.0106	0.3254	0.4347	0.2334
	1500	1250	1000	1000	0.0103	0.0157	0.007	0.2751	0.3648	0.1924
$m \geq n > p$	60	40	50	50	0.1254	0.4945	0.1083	0.8612	1.1608	0.5863
	300	200	250	250	0.0397	0.0697	0.035	0.5227	0.7458	0.3262
	900	600	750	750	0.0169	0.0314	0.0128	0.3433	0.5318	0.2478
	1500	1000	1250	1250	0.0107	0.0193	0.01	0.2866	0.4738	0.2028
$p \geq n > m$	40	60	50	50	0.1834	0.1645	0.1417	1.1144	0.9979	0.4617
	200	300	250	250	0.0511	0.0445	0.0333	0.7188	0.5814	0.3294
	600	900	750	750	0.0212	0.0183	0.0133	0.4400	0.3941	0.2111
	1000	1500	1250	1250	0.0173	0.0132	0.012	0.3897	0.3450	0.1772
$n > m$ $n > p$	20	30	60	50	0.0480	0.0392	0.0	0.4960	0.5349	0.3483
	200	300	600	500	0.0114	0.0110	0.0	0.3478	0.2948	0.1958
	400	600	1200	1000	0.0065	0.0065	0.0	0.2564	0.2313	0.1578
	1000	1500	3000	2500	0.0031	0.0032	0.0	0.1901	0.1670	0.1100

Table 5: Stability profiling for random dense matrices

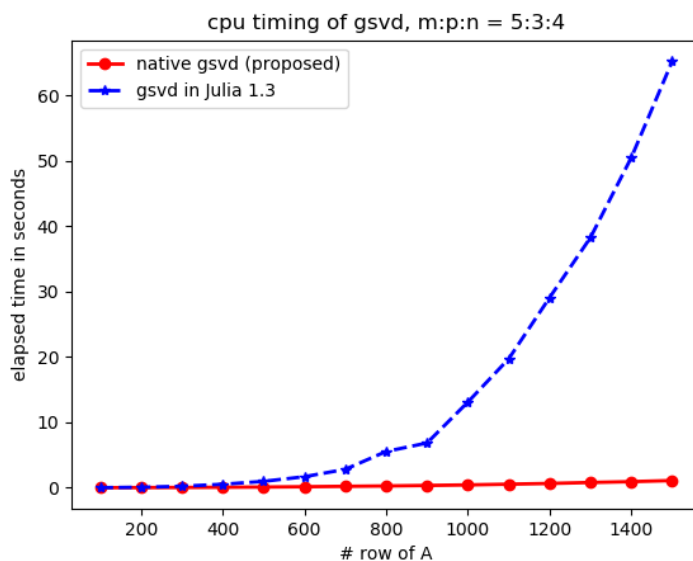
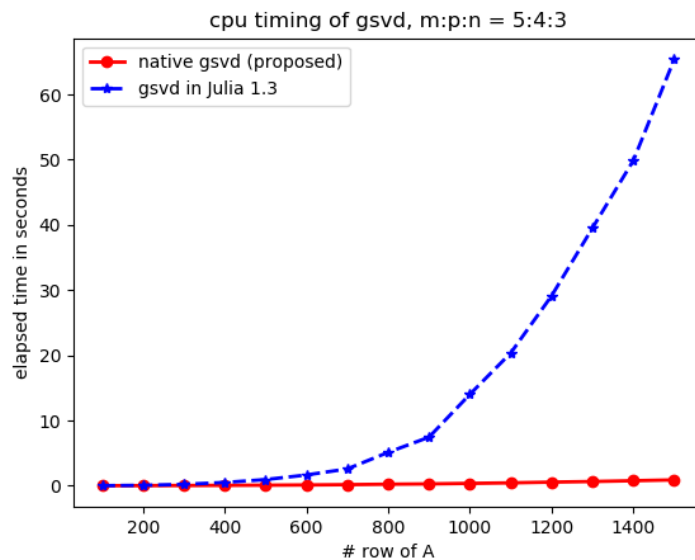
### 1.6.3 Special types of matrices

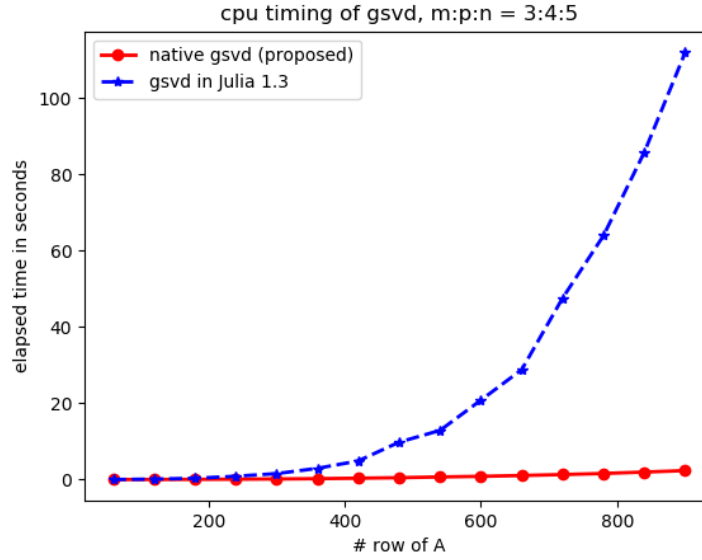
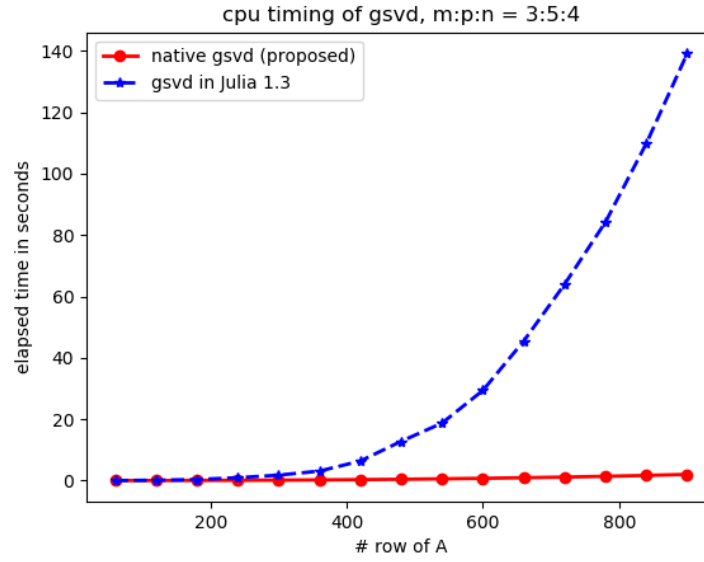
TODO?

## 1.7 Timing

We want to evaluate the timing performance of our implementation between current version in Julia and MATLAB.

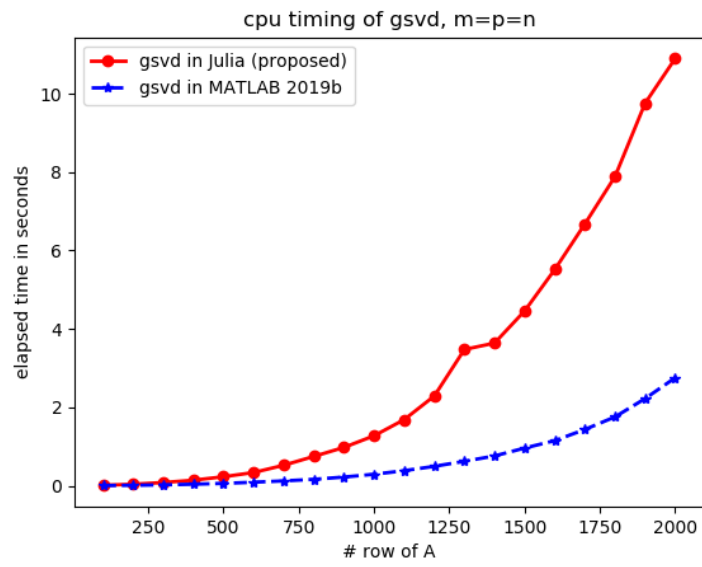
**vs. Julia 1.3** For the comparison with Julia 1.3, we also spilt into four cases. Each case, we calculated the average CPU timing of 10 runs. In all cases, we can see that the speedup is exponential when input size is greater than a few hundreds.





**vs. MATLAB.** For the comparison with MATLAB 2019b, we specify the input as square matrix. Our implementation is still slower than MATLAB. The major reason is due to the significant difference of decomposition discussed in 0.1 and 0.3.





**Profile.** As detailed in 0.6, our algorithm insists of four parts: pre-processing, QR, CSD and post-processing. Here, we measure the CPU time spent in the first three parts and total time, denoted as  $t_{pre}, t_{qr}, t_{csd}$  and  $t_{all}$  and calculated the percentages that each part spent to total time, denoted as  $p_{pre}, p_{qr}, p_{csd}$ . Still, we separate our test into four cases and record the average of 10 test runs. **In most cases, pre-processing dominates the computation effort.**

	$m$	$p$	$n$	$t_{pre}$	$p_{pre}$	$t_{qr}$	$p_{qr}$	$t_{csd}$	$p_{csd}$	$t_{all}$
$m \geq n$ $p \geq n$	1500	1200	1000	0.6242	41.13%	0.1683	11.09%	0.6011	39.61%	1.5175
	500	500	500	0.0651	26.78%	0.0347	14.29%	0.1191	48.94%	0.2433
	650	310	230	0.0418	54.63%	0.0084	11.08%	0.0195	25.47%	0.0766
	430	610	210	0.0345	47.65%	0.0067	9.25%	0.0247	34.11%	0.0725
$m \geq n > p$	1500	1000	1200	1.500	60.09%	0.1815	7.27%	0.6811	27.28%	2.4963
	720	220	540	0.1182	73.65%	0.0074	4.61%	0.0256	15.94%	0.1605
	440	180	440	0.0651	65.84%	0.0053	5.37%	0.0221	22.41%	0.0989
	370	290	350	0.0659	51.61%	0.0123	9.65%	0.0400	31.34%	0.1278
$p \geq n > m$	1000	1500	1200	0.5234	23.23%	0.2789	12.37%	1.2630	56.06%	2.2529
	250	300	300	0.0205	24.96%	0.0129	15.75%	0.0397	48.25%	0.0822
	360	660	600	0.0645	18.33%	0.0436	12.39%	0.2103	59.72%	0.3521
	130	520	480	0.0311	14.52%	0.0215	10.02%	0.1391	64.79%	0.2146
$n > m$ $n > p$	1000	1200	1500	1.7532	48.51%	0.2038	5.64%	1.4467	40.03%	3.6136
	260	600	770	0.2791	38.86%	0.0441	6.14%	0.3459	48.17%	0.7181
	370	250	700	0.1385	86.69%	0	0%	0	0%	0.1598
	120	120	400	0.0296	96.70%	0	0%	0	0%	0.0307

Table 6: Time profiling for GSVD

The Generalized Singular Value Decomposition is powerful in solving many numerical linear algebra problems as well as problems in other disciplines, such as statistics and signal processing. We briefly introduce several prominent applications and highlight one application in genome analysis in this section.

## 1.8 Prominent applications of the GSVD

**Tikhonov regularization.** Tikhonov regularization in general form can be analyzed with the truncated GSVD when we are to solve the ill-posed linear least squares problem. [17] [13] [40] Computerized ionospheric tomography [9] is one of the applications in this regard.

**Matrix pencil  $A - \lambda B$ .** The GSVD is also used in the field of the canonical structure of matrix pencil  $A - \lambda B$ . [21] More specifically, the column and row nullities of  $A$  and  $B$  and common null space reveal the information about the Kronecker structure of  $A - \lambda B$ .

**Generalized total least squares problem.** By making use of the GSVD, one can solve the generalized TLS problem. TLS is also called error-in-variable regression in statistics domain. The great advantage of the GSVD is that it replaces these implicit transformation of data procedures by one, which is numerically reliable and can more easily handle (nearly) singular associated error covariance matrix. [37] [3]

**Genome analysis.** The GSVD is applicable for comparative analysis of genome-scale expression datasets of two different organisms [1] and is further extended to tensor [29]. We will elaborate the role of the GSVD in genomes in Section 1.9.

**Oriented energy and oriented signal-to-signal ratio.** In the context of oriented energy, one of the concerns is to characterize the signal-to-signal ratio of two given sequences of  $m$ -vectors  $\{a_k\}$ ,  $\{b_k\}$ ,  $k = 1, \dots, n$  with associated  $m$ -by- $n$  matrices  $A$  and  $B$ . [12] In other words, we're primarily interested in how to separate the desired signal (for instance  $\{a_k\}$ ) from the undesired one ( $\{b_k\}$ ). More specifically, given that  $\text{rank}(B) = l$ , the question transforms to find the optimal  $l$ -dimensional subspace where the desired signal sequence  $\{a_k\}$  can be optimally distinguished from the corrupting sequence  $\{b_k\}$ .

**Linear discriminant analysis.** Howland and Park [19] [22] applied the GSVD to discriminant analysis to overcome the limitation of nonsingular covariance matrices that are used to represent the scatter within and between clustered text data.

**One Way ANOVA (Analysis of variance).** A commonly used statistics test is to decide whether a proposed clustering of a vector  $v$  is justified. The test takes the average square component in the  $U_2$  direction and divides it by the average square component in the  $U_3$  direction. [41]

## 1.9 GSVD in genome analysis

With the advance of modern genomic technologies, scientists and researchers are able to acquire a wide range of molecular biological data, such as DNA-sequence and mRNA-expression, on a

genomic scale at ease. For instance, DNA microarray, invented by Patrick O. Brown [30], is a collection of microscopic DNA spots attached to a solid surface. Scientists use DNA microarrays to measure the expression levels of large numbers of genes simultaneously or to genotype multiple regions of a genome. DNA microarrays can be used to detect DNA (as in comparative genomic hybridization), or detect RNA (most commonly as cDNA after reverse transcription) that may or may not be translated into proteins.

What's more exciting is that once these data is ready, we can apply comparative analysis to understand the universality and specialization of molecular biology mechanisms. Specifically, such analysis helps us to distinguish the similarity and dissimilarity among two or more large-scale data sets. To this end, a mathematical framework is needed to remove the barriers of transferring genetic sequencing results to understandable information. Fortunately, the GSVD fits this purpose well:

1. Two data sets are column-matched but row-independent matrices, where rows are genes and columns are samples. They exactly match the type of the input matrix pair of the GSVD.
2. The GSVD simultaneously reduce the two “genes”  $\times$  “samples” spaces to two diagonalized “samplelets”  $\times$  “genelets” spaces. The column space is shared by both datasets. We can interpret the generalized singular values as the significance in one data set relative to that in the other.

### 1.9.1 Mathematical framework: the GSVD

The GSVD is the simultaneous linear transformation of the two expression data sets  $A$  and  $B$  from  $m$ -genes-by- $n$ -samples and  $p$ -genes-by- $n$ -samples spaces to two reduced  $n$ -genelets-by- $n$ -samplelets spaces, where both  $m \gg n$  and  $p \gg n$ .

$$A = UC \begin{bmatrix} 0 & R \end{bmatrix} Q^T, \quad B = VS \begin{bmatrix} 0 & R \end{bmatrix} Q^T \quad (1.7)$$

Notably, we have

1.  $C^T C = \text{diag}(\alpha_1^2, \dots, \alpha_n^2)$ ,  $S^T S = \text{diag}(\beta_1^2, \dots, \beta_n^2)$ .
2.  $Q$  is an  $n$ -by- $n$  orthogonal matrix. The rows of  $Q$  form a basis for this row (or gene) space, and are denoted genelets.

We then define the antisymmetric angular distance between two data sets:

$$\theta_i = \arctan(\alpha_i/\beta_i) - \pi/4, \quad \text{for } 1 \leq i \leq n \quad (1.8)$$

It indicates the relative significance of the  $i$ -th genelet, i.e., its significance in the first data set relative to that in the second. The  $\pi/4$  offset means that an angular distance of 0 indicates a genelet of equal significance in both data sets. The angular distances are arranged in decreasing order of significance in the first data set relative to the second such that  $\pi/4 > \theta_1 > \theta_2 > \dots > \theta_n > -\pi/4$ .

The notion of antisymmetric angular distance answers the question we have in general: given two matrices with equal columns, how can we classify the basis vectors in the rows of  $Q$  according to its source. Mathematically, if  $Q_i^T$  denotes the  $i$ -th row of  $Q$ , then the total projection of both data sets onto the genelet  $Q_i^T$  can be thought of as the vector sum of these two orthogonal vectors,  $\vec{\gamma}_i = \vec{\alpha}_i + \vec{\beta}_i$ . The angle between the total projection  $\vec{\gamma}_i$  and that of the second data set  $\vec{\beta}_i$ ,  $\theta_i$  above, measures the extent to which the total projection  $\vec{\gamma}_i$  lies in the direction of the projection of either data set,  $\vec{\alpha}_i$  or  $\vec{\beta}_i$ , and quantifies the relative contribution of each data set to this total projection. [1] (Appendix)

### 1.9.2 Experiment

We now illustrate this framework with the comparison between yeast and human cell cycle-expression data sets. [34] We also present experimental results, implemented in Julia.

**Input data sets** In our experiment, a single microarray probes the relative expression levels of 4523 genes of yeast in a single sample. A series of 18 arrays probes the genome-scale expression levels in 18 different samples, i.e., under 18 different experimental conditions. Similarly. Another set of microarrays probe the relative expression levels of 12056 genes of human under 18 experimental conditions.

In other words, we have the following two matrices as inputs.

1.  $A$ : a 4523-by-18 matrix representing yeast cell expression
2.  $B$ : a 12056-by-18 matrix representing human cell expression

**Data pre-processing** Unfortunately, microarray might fail to record gene expression in some samples. In other words, missing data might tabulate into some portion of the data set. In our experiment, 1825 out of 4523 rows have NULL entries for matrix  $A$  and 7696 out of 12056 rows contain invalid entries in matrix  $B$ .

One of the methods to impute missing data for genomic expressions is to utilize SVD to approximate the expression of all genes in the data set. [18] [35]

We briefly describe the iterative methods to solve this problem.

1. Set the missing elements as as the mean of the non-missing elements for each row, let the complete matrix be  $X_0$ . Initialize  $i = 0$ .
2. Compute the SVD of  $X_i$ , and use the first 5 singular vectors to approximate  $X_{i+1}$  by replacing the missing values in  $X_i$  with the fitted values from this solution.

**Results** Once the data pre-processing step is done, we can apply the GSVD of the two data sets. As a result, we produce matrices  $U, V, C, S, \begin{bmatrix} 0 & R \end{bmatrix}, Q$ . The heatmap below **TODO: will replace with a real one** visualizes the simultaneous linear transformation done by the GSVD. Here, statistically, overexpression (red) pixels are data that are above mean plus one standard deviation, underexpression (green) pixels are data that are below mean plus one standard deviation, while no change (black) pixels are data that lie in between.

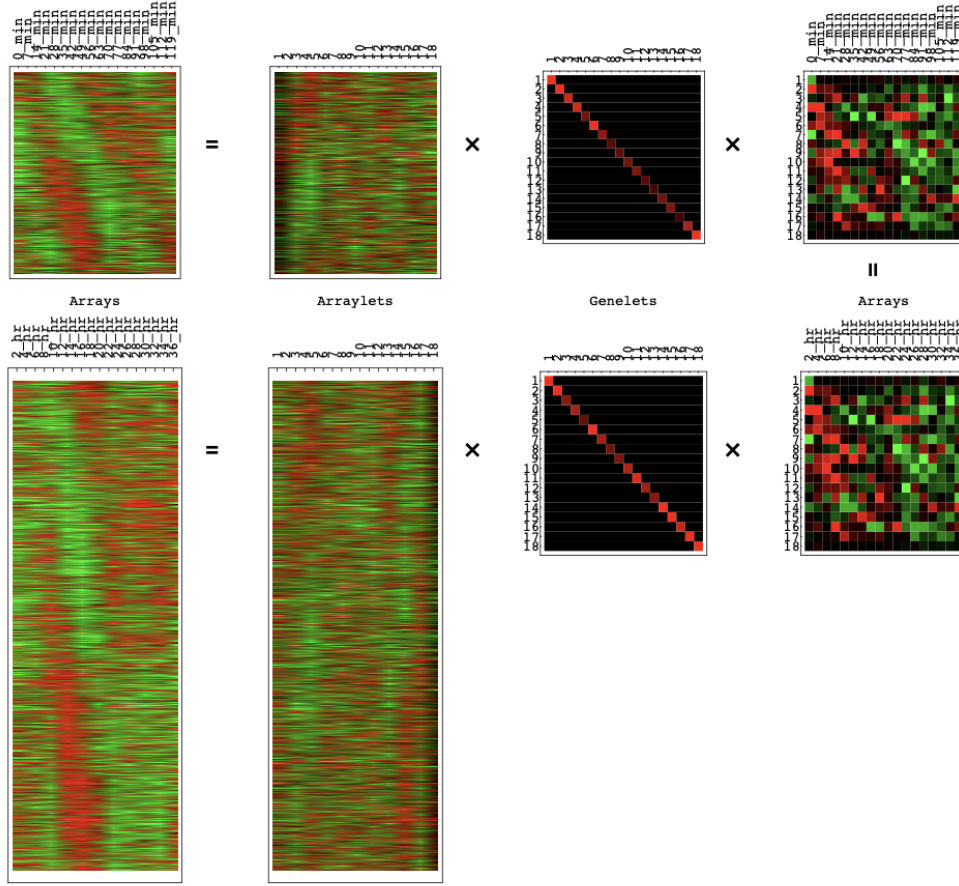


Figure 2: GSVD of the yeast and human cell-cycle expression data sets. Shown is a heatmap with overexpression (red), no change in expression (black), and underexpression (green)

We then compute the antisymmetric angular distance defined by (1.8). By the bar chart of the angular distance, we know that the first and second genelets are highly significant in the yeast data relative to the human data. The third, fourth, and fifth genelets are almost equally significant in both data sets (slightly more in the yeast data), with  $0 < \theta_3, \theta_4, \theta_5 < \pi/16$ . The 14th, 15th, and 16th genelets, which are also almost equally significant in both data sets (slightly more in the human data), with  $\pi/6 < \theta_{14}, \theta_{15}, \theta_{16} < 0$ . The 17th and 18th genelets are highly significant in the human data relative to the yeast data. All other genelets are significant in neither the yeast data nor the human data.

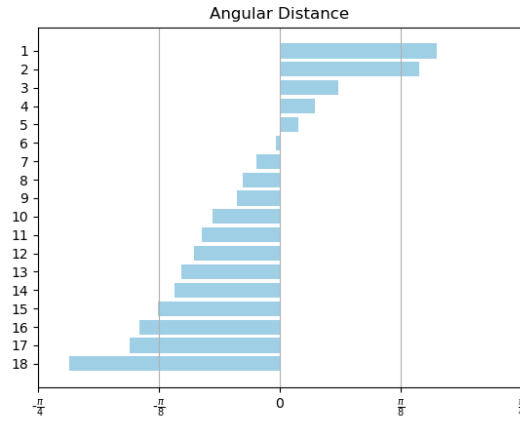


Figure 3: Bar chart of the angular distances

The GSVD provides a natural solution by creating a single coherent model from the two datasets recording different aspects of interrelated phenomena by simultaneously identifying the similar and dissimilar between the two corresponding column-matched but row-independent matrices. More recently, research conducted by Alter’s lab at the University of Utah dive the GSVD into tensor level [10] and apply it to the study of tumor, e.g. glioblastoma. [28]

## References

- [1] Orly Alter, Patrick O Brown, and David Botstein. Generalized singular value decomposition for comparative analysis of genome-scale expression data sets of two different organisms. Proceedings of the National Academy of Sciences, 100(6):3351–3356, 2003.
- [2] Edward Anderson, Zhaojun Bai, Christian Bischof, L Susan Blackford, James Demmel, Jack Dongarra, Jeremy Du Croz, Anne Greenbaum, Sven Hammarling, Alan McKenney, et al. LAPACK Users’ guide. SIAM, 1999.
- [3] Zhaojun Bai. CSD, GSVD, their applications and computations. IMA Preprint Series # 958, 1992.
- [4] Zhaojun Bai and James W Demmel. Computing the generalized singular value decomposition. SIAM Journal on Scientific Computing, 14(6):1464–1486, 1993.
- [5] Zhaojun Bai and Hongyuan Zha. A new preprocessing algorithm for the computation of the generalized singular value decomposition. SIAM Journal on Scientific Computing, 14(4):1007–1012, 1993.
- [6] Jeff Bezanson, Jiahao Chen, Benjamin Chung, Stefan Karpinski, Viral B Shah, Jan Vitek, and Lionel Zoubritzky. Julia: Dynamism and performance reconciled by design. Proceedings of the ACM on Programming Languages, 2(OOPSLA):1–23, 2018.
- [7] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. SIAM review, 59(1):65–98, 2017.
- [8] Jeff Bezanson, Stefan Karpinski, Viral B Shah, and Alan Edelman. Julia: A fast dynamic language for technical computing. arXiv preprint arXiv:1209.5145, 2012.
- [9] K Bhuyan, SB Singh, and PK Bhuyan. Application of generalized singular value decomposition to ionospheric tomography. Annales Geophysicae, 22(10):3437–3444, 2004.
- [10] Matthew W Bradley, Katherine A Aiello, Sri Priya Ponnappalli, Heidi A Hanson, and Orly Alter. Gsvd-and tensor gsvd-uncovered patterns of dna copy-number alterations predict adenocarcinomas survival in general and in response to platinum. APL bioengineering, 3(3):036104, 2019.
- [11] Timothy A Davis et al. Suitesparse: A suite of sparse matrix software, 2015.
- [12] BL De Moor. Mathematical concepts and techniques for modelling of static and dynamic systems. Ph. D. dissertation, Dep. of Elect. Eng., Katholieke Univ., 1988.
- [13] Laura Dykes and Lothar Reichel. Simplified GSVD computations for the solution of linear discrete ill-posed problems. Journal of Computational and Applied Mathematics, 255:15–27, 2014.
- [14] Alan Edelman. Julia: A fresh approach to technical computing and data processing. Technical report, MASSACHUSETTS INST OF TECH CAMBRIDGE CAMBRIDGE, 2019.



- [15] Alan Edelman and Yuyang Wang. The GSVD: Where are the ellipses?, matrix trigonometry, and more. arXiv preprint arXiv:1901.00485, 2019.
- [16] Gene H. Golub and Charles F. Van Loan. Matrix Computations 4th Edition The Johns Hopkins University Press. Johns Hopkins University Press, USA, 2013.
- [17] Per Christian Hansen. Regularization, gsvd and truncatedgsvd. BIT numerical mathematics, 29(3):491–504, 1989.
- [18] Trevor Hastie, Robert Tibshirani, Gavin Sherlock, Michael Eisen, Patrick Brown, and David Botstein. Imputing missing data for gene expression arrays. 1999.
- [19] Peg Howland, Moongu Jeon, and Haesun Park. Structure preserving dimension reduction for clustered text data based on the generalized singular value decomposition. SIAM Journal on Matrix Analysis and Applications, 25(1):165–179, 2003.
- [20] Ross Ihaka and Robert Gentleman. R: a language for data analysis and graphics. Journal of computational and graphical statistics, 5(3):299–314, 1996.
- [21] Bo Kågström. The generalized singular value decomposition and the general  $(\mathbf{A} - \lambda\mathbf{B})$ -problem. BIT Numerical Mathematics, 24(4):568–583, 1984.
- [22] Hyunsoo Kim, Peg Howland, and Haesun Park. Dimension reduction in text classification with support vector machines. Journal of machine learning research, 6(Jan):37–53, 2005.
- [23] MATLAB. Generalized singular value decomposition documentation. The MathWorks Inc., Natick, Massachusetts, 2019. Available at <https://www.mathworks.com/help/matlab/ref/gsvd.html>.
- [24] MATLAB. Matlab, 2019. Available at <https://www.mathworks.com/>.
- [25] James M. Ortega. Introduction to FORTRAN 90 for Scientific Computing. Oxford University Press, Inc., USA, 1st edition, 1994.
- [26] CC Paige. Computing the generalized singular value decomposition. SIAM Journal on Scientific and Statistical Computing, 7(4):1126–1146, 1986.
- [27] Jeffrey M Perkel. Julia: come for the syntax, stay for the speed. Nature, 572(7768):141–143, 2019.
- [28] Sri Priya Ponnappalli, Matthew W Bradley, Karen Devine, Jay Bowen, Sara E Coppens, Kristen M Leraas, Brett A Milash, Fuqiang Li, Huijuan Luo, Shi Qiu, et al. Retrospective clinical trial experimentally validates glioblastoma genome-wide pattern of dna copy-number alterations predictor of survival. APL bioengineering, 4(2):026106, 2020.
- [29] Preethi Sankaranarayanan, Theodore E Schomay, Katherine A Aiello, and Orly Alter. Tensor gsvd of patient-and platform-matched tumor and normal dna copy-number profiles uncovers chromosome arm-wide patterns of tumor-exclusive platform-consistent alterations encoding for cell transformation and predicting ovarian cancer survival. PloS one, 10(4):e0121396, 2015.

- [30] Mark Schena, Dari Shalon, Ronald W Davis, and Patrick O Brown. Quantitative monitoring of gene expression patterns with a complementary dna microarray. Science, 270(5235):467–470, 1995.
- [31] Avik Sengupta. Julia High Performance. Packt, USA, 2nd edition, 2019.
- [32] Michael Stewart. Rank decisions in matrix quotient decompositions. SIAM Journal on Matrix Analysis and Applications, 37(4):1729–1746, 2016.
- [33] Brian D Sutton. Computing the complete cs decomposition. Numerical Algorithms, 50(1):33–65, 2009.
- [34] Saeed Tavazoie, Jason D Hughes, Michael J Campbell, Raymond J Cho, and George M Church. Systematic determination of genetic network architecture. Nature genetics, 22(3):281–285, 1999.
- [35] Olga Troyanskaya, Michael Cantor, Gavin Sherlock, Pat Brown, Trevor Hastie, Robert Tibshirani, David Botstein, and Russ B Altman. Missing value estimation methods for dna microarrays. Bioinformatics, 17(6):520–525, 2001.
- [36] Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux. The numpy array: a structure for efficient numerical computation. Computing in science & engineering, 13(2):22–30, 2011.
- [37] Sabine Van Huffel and Joos Vandewalle. Analysis and properties of the generalized total least squares problem  $\mathbf{AX} \approx \mathbf{B}$  when some or all columns in  $\mathbf{A}$  are subject to error. SIAM Journal on Matrix Analysis and Applications, 10(3):294, 1989.
- [38] Charles Van Loan. Computing the CS and the generalized singular value decompositions. Numerische Mathematik, 46(4):479–491, 1985.
- [39] Charles F Van Loan. Generalizing the singular value decomposition. SIAM Journal on Numerical Analysis, 13(1):76–83, 1976.
- [40] Yimin Wei, Pengpeng Xie, and Liping Zhang. Tikhonov regularization and randomized GSVD. SIAM Journal on Matrix Analysis and Applications, 37(2):649–675, 2016.
- [41] Wikipedia. One-way analysis of variance, Jun 2020. Available at [https://en.wikipedia.org/wiki/One-way\\_analysis\\_of\\_variance](https://en.wikipedia.org/wiki/One-way_analysis_of_variance).
- [42] Wolfram Research, Inc. Mathematica 8.0.