A note on incremental SVD: reorthogonalization

Yangwen Zhang

April 13, 2022

Abstract

Incremental singular value decomposition (SVD) was proposed by Brand to efficiently compute the SVD of a matrix. The algorithm updates the SVD of a matrix when one or more columns are added to the matrix. However, in practice, small numerical errors cause a loss of orthogonality; hence a reorthogonalization is needed and the computational cost can be high. In this paper, we modify Brand’s algorithm to reduce the expense of the reorthogonalization. Numerical experiments are presented to illustrate the efficiency of the modification.

1 Introduction

The singular value decomposition (SVD) of a matrix has many applications, such as proper orthogonal decomposition model order reduction and principal component analysis.

One drawback of the SVD is the cost of its computation. Let $U$ be a $m$ by $n$ dense matrix of low rank $r$, the computational complexity of traditional methods \cite{7} is $O(mn^2 + m^2n + n^3)$ time, which is unfeasible for a large size matrix. Lanczos methods \cite{2} yield thin svds and the complexity is $O(mrn^2)$ time, but we need to know the rank in advance. These methods are referred to as batch methods because they need to store the matrix.

In some scenarios the data sets will be produced incrementally, such as the snapshots of a time dependent partial differential equations (PDEs). It may be advantageous to perform the SVD as the columns of a matrix become available, instead of waiting until all data sets are available before doing any computation. These characteristics on the availability of data sets have given rise to a class of incremental methods.

In 2002, Brand \cite{3} proposed a new algorithm to find the SVD of a matrix incrementally. Given the SVD of a matrix $U = Q\Sigma R^\top$, the goal is to update the SVD of of the related matrix $[\begin{array}{c} U \\ B \end{array}]$, by using the SVD of $U$ and the new adding matrix $B$. In this paper, we restricted ourselves the case $B$ is a column, denoted by $c$. Then the SVD of $[\begin{array}{c} U \\ c \end{array}]$ can be constructed by

1. letting $e = c - QQ^\top c$ and $p = \|e\|$,

2. finding the full SVD of $[\begin{array}{cc} \Sigma & U^\top c \\ 0 & p \end{array}] = \tilde{Q}\tilde{\Sigma}\tilde{Q}$, and then

3. updating the SVD of $[\begin{array}{c} U \\ c \end{array}]$ by

$$[\begin{array}{c} U \\ c \end{array}] = \left(\begin{array}{cc} Q & e/p \end{array}\right)\tilde{Q}\tilde{\Sigma}\left(\begin{array}{cc} R & 0 \\ 0 & 1 \end{array}\right)\tilde{R}^\top.$$
For many of the motivating applications, only the dominant singular vectors and values of $U$ are needed. Hence, in practice, we perform truncation when $p$ or the last singular value of $\tilde{\Sigma}$ is small.

Theoretically, the matrix $[ Q \ e/p \ ] \tilde{Q}$ is orthogonal. However, in practice, small numerical errors cause a loss of orthogonality. Without a reorthogonalization, the decomposition of the incremental SVD algorithm is not orthogonal. Therefore, the singular values are not true and truncation is not reliable. Hence a reorthogonalization is needed and the computational cost can be high if the rank of the matrix is not extremely low. Specifically, Brand recommended a Gram-Schmidt orthogonalization [3] while Oxberry et al. chose thin QR in [8]. Our numerical test shows that thin QR is faster than the Gram-Schmidt, see Example 1.

If the data arising from a Galerkin-type simulation of a PDE, one has to deal with Cholesky factorization of a weighted matrix if one directly apply Brand’s algorithm. To avoid this, Fareed et al [6] extended Brand’s algorithm to accommodate data of this type without computing Cholesky factorization. More specifically, if the data lies in a finite element space and is expressed using a collection of basis functions. Then the SVD of the data is equivalent to find a so-called core SVD of their coefficient matrix.

**Definition 1.** [6] A core SVD of a matrix $U \in \mathbb{R}^{m \times n}$ is a decomposition $U = Q\Sigma R^\top$, where $Q \in \mathbb{R}^{m \times d}, \Sigma \in \mathbb{R}^{d \times d}$, and $R \in \mathbb{R}^{n \times d}$ satisfy

$$Q^\top W Q = I, \quad R^\top R = I, \quad \Sigma = \text{diag} (\sigma_1, \ldots, \sigma_d),$$

where $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_d > 0$. The values $\{\sigma_i\}$ are called the (positive) singular values of $U$ and the columns of $Q$ and $R$ are called the corresponding singular vectors of $U$.

We note that if $W = I$, the above core SVD is reduced to the standard SVD of the matrix $U$. In [6], $W$ is the standard finite element mass or stiffness matrix. For more information about the incremental SVD in a weighted norm setting, see [4, 5].

There is no doubt that the reorthogonalization is also needed in this situation. Since there is no thin QR with a weighted norm setting available, Fareed et al. in [6] used the $W$-weighted Gram–Schmidt for the reorthogonalization. They claimed that the reorthogonalization step is a large part of the computational cost of the incremental SVD algorithm.

In this paper, we observe that the reorthogonalization of the whole matrix $[ Q \ e/p \ ] \tilde{Q}$ is not necessary. We note that the orthogonality of $\tilde{Q}$ can be always preserved in a reasonable numerical precision while the matrix $[ Q \ e/p \ ]$ cannot. By the construction of $[ Q \ e/p \ ]$, we can reorthogonalize it recursively: only apply the Gram-Schmidt to the new adding vector $e/p$ since the matrix $Q$ has already been reorthogonalized; see Algorithm 3 for more details. By this simply modification, the computational is reduced up to 60% of the CPU time; see Examples 1 and 2.

### 2 Incremental SVD

We begin by introducing notation needed throughout this work. For convenience, we adopt Matlab notation herein. Given a matrix $A$, we use $A(:,1:r)$ to denote the first $r$ columns of $A$, and $A(1:r,1:r)$ be the $r$–th leading principal minor of $A$.

Next, we briefly discuss the incremental SVD algorithm.

**Step 1: Initialization.** Assume that the first column of $U$ is nonzero, i.e., $u_1 \neq 0$, we initialize the core SVD of $u_1$ by setting

$$\Sigma = (u_1^\top W u_1)^{1/2}, \quad Q = u_1 \Sigma^{-1}, \quad R = 1.$$
Algorithm 1 (Initialize incremental SVD)

**Input:** $u_1 \in \mathbb{R}^m$, $W \in \mathbb{R}^{m \times m}$

1: $\Sigma = (u_1^T W u_1)^{1/2}$;
2: $Q = u_1 \Sigma^{-1}$;
3: $R = 1$;
4: return $Q, \Sigma, R$.

The algorithm is shown in Algorithm 1.

**Step 2: Core SVD of $U_\ell$.** Suppose we already have the rank-$k$ truncated core SVD of the first $\ell$ columns of $U$:

$$U_\ell = Q \Sigma R^T,$$

where $\Sigma \in \mathbb{R}^{k \times k}$ is a diagonal matrix with the $k$ (ordered) singular values of $U_\ell$ on the diagonal, $Q \in \mathbb{R}^{m \times k}$ is the matrix containing the corresponding $k$ left singular vectors of $U_\ell$, and $R \in \mathbb{R}^{\ell \times k}$ is the matrix of the corresponding $k$ right singular vectors of $U_\ell$.

**Step 3: Updates the core SVD of $U_{\ell+1}$.** Our goal next is to update the above SVD, i.e., we want to find the core SVD of $U_{\ell+1}$ by using the core SVD of $U_\ell$ and $U_{\ell+1}$. In other words, we update the core SVD of $U_{\ell+1}$ only use $Q$, $\Sigma$ and $R$ in (2.1), and the column $u_{\ell+1}$.

To do this, let $e \in \mathbb{R}^m$ be the residual of $U_{\ell+1}$ onto the space spanned by the columns of $Q$ with the weighted inner product $(\cdot, \cdot)_W$. Therefore,

$$e = u_{\ell+1} - QQ^T W u_{\ell+1}.$$

Let $p$ be the magnitude of $e$, i.e., $p = (e^T W e)^{1/2}$. If $p > 0$, we let $\bar{e}$ be the unit vector in the direction of $e$, i.e., $\bar{e} = e/p$. If $p = 0$, we set $\bar{e} = 0$. Then we have the fundamental identity

$$U_{\ell+1} = \begin{bmatrix} U_\ell & u_{\ell+1} \end{bmatrix} = \begin{bmatrix} Q \Sigma R^T & u_{\ell+1} \end{bmatrix} = \begin{bmatrix} Q \bar{e} \end{bmatrix} \begin{bmatrix} \Sigma & Q^T W u_{\ell+1} \\ 0 & p \end{bmatrix} \begin{bmatrix} R & 0 & 0 \\ 0 & 1 \end{bmatrix}^T.$$

We can find the SVD of the updated matrix $U_{\ell+1}$ by finding the SVD of the matrix $Y$ in the right hand side of the above identity. In practice, truncation is performed when $p$ is very small.

1. If $p < \text{tol}_1$, we approximate and set $p = 0$ and $\bar{e} = 0$. Let $Q_Y \Sigma_Y R_Y^T$ be the full SVD of $Y = \begin{bmatrix} \Sigma & Q^T W u_{\ell+1} \\ 0 & 0 \end{bmatrix}$.

Then the SVD of $U_{\ell+1} \in \mathbb{R}^{m \times (\ell+1)}$ is given by

$$U_{\ell+1} = \begin{bmatrix} U_\ell & u_{\ell+1} \end{bmatrix} = (Q \ 0) \Sigma_Y \left( \begin{bmatrix} R & 0 \\ 0 & 1 \end{bmatrix} R_Y \right)^T.$$

It is easy to see that $\Sigma_Y(k+1, k+1) = 0$ and the last column of $Q \ 0 \ Q_Y$ is zero. This suggests the following update

$$Q \leftarrow QQ_Y(1:k, 1:k), \quad \Sigma \leftarrow \Sigma_Y(1:k, 1:k), \quad R \leftarrow \begin{bmatrix} R & 0 \\ 0 & 1 \end{bmatrix} R_Y(:, 1:k).$$
(2) If \( p \geq \text{tol}_1 \), we let \( Q_Y \Sigma_Y R_Y^T \) be the full SVD of the middle matrix \( Y \) in (2.2), and let

\[
\tilde{Q} = \begin{bmatrix} Q & \bar{e} \end{bmatrix} \in \mathbb{R}^{m \times (k+1)}, \quad \tilde{\Sigma} = \Sigma_Y \in \mathbb{R}^{(k+1) \times (k+1)}, \quad \tilde{R} = \begin{bmatrix} R & 0 \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{(m+1) \times (k+1)}.
\]

(i) If \( \tilde{\Sigma}(k+1,k+1) > \text{tol}_2 \), this suggests the following update:

\[
Q \leftarrow \tilde{Q}, \quad \Sigma \leftarrow \tilde{\Sigma}, \quad R \leftarrow \tilde{R}.
\]

(ii) If \( \tilde{\Sigma}(k+1,k+1) \leq \text{tol}_2 \), this suggests the following update:

\[
Q \leftarrow \tilde{Q}(1:k,1:k), \quad \Sigma \leftarrow \Sigma(1:k,1:k), \quad R \leftarrow \tilde{R}(1:k,1:k).
\]

Now we summarize the algorithm in Algorithm 2.

Algorithm 2 (Update Incremental SVD)

**Input:** \( Q \in \mathbb{R}^{m \times k}, \Sigma \in \mathbb{R}^{k \times k}, R \in \mathbb{R}^{\ell \times k}, u_{\ell+1} \in \mathbb{R}^m, W \in \mathbb{R}^{m \times m}, \text{tol}_1, \text{tol}_2, \text{tol}_3 \)

1. Set \( d = Q^T(Wu_{\ell+1}); \)
2. Set \( e = u_{\ell+1} - Qd; \)
3. Set \( p = (e^TWe)^{1/2}; \)
4. if \( p < \text{tol}_1 \) then
   5. Set \( p = 0; \)
5. else
   6. Set \( e = e/p; \)
7. end if
8. Set \( Y = \begin{bmatrix} \Sigma & d \\ 0 & p \end{bmatrix}; \)
9. \( [Q_Y, \Sigma_Y, R_Y] = \text{svd}(Y); \)
10. if \( p < \text{tol}_1 \) then
   11. Set \( Q = QQ_Y(:,1:k), \quad \Sigma = \Sigma_Y(1:k,1:k), \quad R = \begin{bmatrix} R & 0 \\ 0 & 1 \end{bmatrix} R_Y; \)
12. else if \( p \geq \text{tol}_1 \) then
   13. Set \( Q = \begin{bmatrix} Q & e \end{bmatrix} Q_Y, \quad \Sigma = \Sigma_Y, \quad R = \begin{bmatrix} R & 0 \\ 0 & 1 \end{bmatrix} R_Y. \)
   14. if \( \Sigma(k+1,k+1) < \text{tol}_2 \) then
   15. Set \( Q = Q(:,1:k), \quad \Sigma = \Sigma(1:k,1:k), \quad R = R(:,1:k); \)
   16. end if
17. end if
18. return \( Q, \Sigma, R. \)

**Step 4: Reorthogonalization.** Theoretically, the above SVD update yields orthonormal left and right singular vectors. However, in practice, small numerical errors cause a loss of orthogonality. Then the incremental SVD algorithm is a non-orthogonal decomposition, and the singular values are not true hence truncation is not reliable. Hence a reorthogonalization is needed and the computational cost can be high if the rank of the matrix is not extremely low. Specifically, Brand recommended a Gram-Schmidt orthogonalization [3] while the authors chose thin QR in [8]. Our numerical test show that thin QR is faster than the Gram-Schmidt when the weighted matrix is an identity, see Example 1. However, since there is no thin QR with a weighted norm setting at hand, the authors in [6] use the \( W \)-weighted Gram–Schmidt for the reorthogonalization.
Algorithm 3 (Reorthogonalization)

Input: $Q \in \mathbb{R}^{m \times k}$, $W \in \mathbb{R}^{m \times m}$, $\text{tol}_3$

1: if $(Q(:,\text{end}))^\top W Q(:,1)) > \text{tol}_3$ then
2: for $i = 1$ to $k$ do
3: Set $\alpha = Q(:,i)$;
4: for $j = 1$ to $i - 1$ do
5: $Q(:,i) = Q(:,i) - (\alpha^\top W Q(:,j)) Q(:,j)$;
6: end for
7: Set $\text{norm} = ((Q(:,i))^\top W Q(:,i))^{1/2}$;
8: Set $Q(:,i) = Q(:,i)/\text{norm}$;
9: end for
10: end if
11: return $Q$.

Finally we conclude the full implementation of the incremental SVD with the $W$-weighted Gram–Schmidt for the reorthogonalization in Algorithm 4

Algorithm 4 (Fully incremental SVD)

Input: $W \in \mathbb{R}^{m \times m}$, $\text{tol}_1$, $\text{tol}_2$, $\text{tol}_3$

1: Get $u_1$;
2: $[Q, \Sigma, R] =$ InitializeIncrementalSVD ($u_1, W$); % Algorithm 1
3: for $\ell = 2, \ldots, n$ do
4: Get $u_\ell$;
5: $[Q, \Sigma, R] =$ UpdateIncrementalSVD ($Q, \Sigma, R, u_\ell, W, \text{tol}_1, \text{tol}_2$); % Algorithm 2
6: $Q =$ Reorthogonalization ($Q, W, \text{tol}_3$); % Algorithm 3
7: end for
8: return $Q, \Sigma, R$.

3 The modification

The reorthogonalization step (Algorithm 3) described above can be expensive if the rank is not extremely low due to the two for loops. We observe that the reorthogonalization of the whole matrix ($\left[ \begin{array}{c} Q \ e \end{array} \right] Y$) is not necessary. We note that the orthogonality of $\tilde{Q} Y$ can be always preserved in a reasonable numerical precision while the matrix $\left[ \begin{array}{c} Q \ e \end{array} \right]$ cannot. By the construction of this matrix, we can reorthogonalize it recursively: only apply the $W$-weighted Gram-Schmidt to the new adding vector $e$ since the matrix $Q$ has already been reorthogonalized; see Algorithm 5 for more details. Numerical experiments in next section show that this simple modification reduce the computational cost while keep the same accuracy.
Algorithm 5 (Incremental SVD Modification)

Input: $Q \in \mathbb{R}^{m \times k}$, $\Sigma \in \mathbb{R}^{k \times k}$, $R \in \mathbb{R}^{\ell \times k}$, $u_{\ell+1} \in \mathbb{R}^m$, $W \in \mathbb{R}^{m \times m}$, $\text{tol}_1, \text{tol}_2, \text{tol}_3$

1: Set $d = Q^T(Wu_{\ell+1})$;
2: Set $e = u_{\ell+1} - Qd$;
3: Set $p = (e^TWe)^{1/2}$;
4: if $p < \text{tol}_1$ then
5: Set $p = 0$;
6: else
7: Set $e = e/p$
8: if $|e^TWQ(:, 1)| > \text{tol}_3$ then
9: $e = e - Q(Q^T(We))$;
10: $p = (e^TWe)^{1/2}$;
11: $\tilde{e} = e/p$;
12: end if
13: end if
14: Set $Y = \begin{bmatrix} \Sigma & d \\ 0 & p \end{bmatrix}$;
15: $[Q_Y, \Sigma_Y, R_Y] = \text{svd}(Y)$;
16: if $p < \text{tol}_1$ then
17: Set $Q = QQ_Y(:, 1 : k)$, $\Sigma = \Sigma_Y(1 : k, 1 : k)$, $R = \begin{bmatrix} R & 0 & 0 \\ 0 & 1 \end{bmatrix} R_Y$;
18: else if $p \geq \text{tol}_1$ then
19: Set $Q = \begin{bmatrix} Q & e \end{bmatrix} Q_Y$, $\Sigma = \Sigma_Y$, $R = \begin{bmatrix} R & 0 & 0 \\ 0 & 1 \end{bmatrix} R_Y$;
20: if $\Sigma(k+1, k+1) < \text{tol}_2$ then
21: $Q = Q(:, 1 : k)$, $\Sigma = \Sigma(1 : k, 1 : k)$, $R = R(:, 1 : k)$;
22: end if
23: end if
24: return $Q, \Sigma, R$.

Algorithm 6 (Fully incremental SVD)

Input: $W \in \mathbb{R}^{m \times m}$, $\text{tol}_1, \text{tol}_2, \text{tol}_3$

1: Get $u_1$;
2: $[Q, \Sigma, R] = \text{InitializeIncrementalSVD}(u_1, W)$; % Algorithm 1
3: for $\ell = 2, \ldots, n$ do
4: Get $u_\ell$
5: $[Q, \Sigma, R] = \text{IncrementalSVDModification}(Q, \Sigma, R, u_\ell, W, \text{tol}_1, \text{tol}_2)$; % Algorithm 5
6: end for
7: return $Q, \Sigma, R$.

4 Numerical experiments.

In this section, we present two examples to show the efficiency of our modification. All the code for all examples in the paper has been made by the author using MATLAB R2020b and has been run on a laptop with MacBook Pro, 2.3 Ghz8-Core Intel Core i9 with 64GB 2667 Mhz DDR4.
Example 1. We let $\Omega \subset \mathbb{R}^2$ be an unit square and we partition it into 524288 uniform triangles. Let $\{\varphi_i\}_{i=1}^m$ be the standard linear finite element basis functions. Next we let $\{t_i\}_{i=1}^n$ be an equally space grid in $[0, 10]$ and time step $\Delta t = 10^{-4}$. Define

$$f(t, x, y) = \cos(t(x + y)),$$

$$(f(t_i), \varphi_j)_{j=1}^m, \quad F = [f_1 | f_2 | \ldots | f_n].$$

For comparisons, we use Algorithm 4, [8, Algorithm 2] and Algorithm 6 to find the SVD of the matrix $F$ by setting $\text{tol}_1 = \text{tol}_2 = \text{tol}_3 = 10^{-12}$ and the weighted matrix $W = I$. We show the first 12 singular values in Table 1. We see that the first 11 singular values are exactly the same for the three different approaches, only the last reported singular value have magnitude of $10^{-11}$ error (marked red).

| Algorithm 4 | Gram-Schmidt (Algorithm 4) | Thin QR [8, Algorithm 2] | New modification (Algorithm 6) |
|-------------|--------------------------|-------------------------|-------------------------------|
| CPU         | 69.2                     | 37.3                    | 1.3                           |

Table 1: The first 12 singular values of $F$ by using the three different algorithms.

Next, we test the efficiency of our modification. We show the CPU time (seconds) of the reorthogonalization part of each algorithm in Table 2 and the whole simulation in Table 3. We see that the CPU time of reorthogonalization part takes 37.9%, 24.2% and 1.2% for the three different algorithms. This means that our new modification saves at least 20% of the CPU time.

| CPU         | Algorithm 4 | [8, Algorithm 2] | Algorithm 6 |
|-------------|-------------|-----------------|-------------|
| Gram-Schmidt (Algorithm 4) | 182.6 | 153.9 | 112.2 |

Table 2: The CPU time (seconds) of reorthogonalization part for the three different algorithms.

Table 3: The whole CPU time (seconds) of three different algorithms.

Example 2. We use the same data settings as in Example 1 and we only test the differences between Algorithm 4 and Algorithm 6 with a weighted matrix. Let $\{(x_i, y_j)\}_{i,j=1}^m$ be the grids of $\Omega$. Define

$$M = [(\varphi_j, \varphi_i)]_{i,j=1}^m, \quad u_k = [f(t_k, x_i, y_j)]_{i,j=1}^m, \quad U = [u_1 | u_2 | \ldots | u_n].$$

Here $M$ is the finite element mass matrix and $u_k$ is the coefficient of the linear Lagrange interpolation of $f(t, x, y)$ at $t = t_k$. We use the two different ways to find the core SVD of the matrix $U$ by setting $\text{tol}_1 = \text{tol}_2 = \text{tol}_3 = 10^{-10}$ and the weighted matrix $W = M$. From Table 4 we see that the CPU time of reorthogonalization part takes almost 67% for Algorithm 4, while only 4% for Algorithm 6. The computational cost is greatly reduced while we can keep the accuracy; see their first 18 singular values in Table 5.
Yangwen Zhang

|                | Full incremental SVD | Reorthogonalization |
|----------------|-----------------------|---------------------|
| Algorithm 4    | 614.15                | 411.35              |
| Algorithm 6    | 206.30                | 5.0944              |

Table 4: The CPU time (seconds) of the full incremental SVD and reorthogonalization for the two different algorithms with a weighted matrix $M$ (Mass matrix).

|                | Algorithm 4             | Algorithm 6            |
|----------------|-------------------------|-------------------------|
|                | 3.7394E+01, 3.3774E+01, 3.0712E+01, 2.5717E+01, 2.2557E+01, 1.3758E+01 | 3.7394E+01, 3.3774E+01, 3.0712E+01, 2.5717E+01, 2.2557E+01, 1.3758E+01 |
|                | 1.1596E+01, 3.2255E+00, 4.4311E-01, 4.1619E-02, 2.9735E-03, 1.6961E-04 | 3.3695E-06, 3.4260E-08, 3.1518E-08, 2.4227E-08, 1.9458E-08, 1.4977E-08 |
|                | 7.9645E-06, 3.3026E-07, 3.4260E-08, 3.1518E-08, 2.4227E-08, 1.9458E-08 | 3.3695E-06, 3.4260E-08, 3.1518E-08, 2.4227E-08, 1.9458E-08, 1.4977E-08 |

Table 5: The first 18 singular values of $U$ of the two algorithms.

References

[1] C. Bach, D. Ceglia, L. Song, and F. Duddeck, Randomized low-rank approximation methods for projection-based model order reduction of large nonlinear dynamical problems, International Journal for Numerical Methods in Engineering, 118 (2019), pp. 209–241.

[2] M. W. Berry, Large-scale sparse singular value computations, The International Journal of Supercomputing Applications, 6 (1992), pp. 13–49.

[3] M. Brand, Incremental singular value decomposition of uncertain data with missing values, in European Conference on Computer Vision, Springer, 2002, pp. 707–720.

[4] H. Fareed and J. R. Singler, A note on incremental POD algorithms for continuous time data, Appl. Numer. Math., 144 (2019), pp. 223–233, https://doi.org/10.1016/j.apnum.2019.04.020.

[5] H. Fareed and J. R. Singler, Error analysis of an incremental proper orthogonal decomposition algorithm for PDE simulation data, J. Comput. Appl. Math., 368 (2020), pp. 112525, 14, https://doi.org/10.1016/j.cam.2019.112525.

[6] H. Fareed, J. R. Singler, Y. Zhang, and J. Shen, Incremental proper orthogonal decomposition for PDE simulation data, Comput. Math. Appl., 75 (2018), pp. 1942–1960, https://doi.org/10.1016/j.camwa.2017.09.012.

[7] G. H. Golub and C. F. Van Loan, Matrix computations, johns hopkins u, Math. Sci., Johns Hopkins University Press, Baltimore, MD, (1996).

[8] G. M. Oxberry, T. Kostova-Vassilevska, W. Arrighi, and K. Chand, Limited-memory adaptive snapshot selection for proper orthogonal decomposition, Internat. J. Numer. Methods Engrg., 109 (2017), pp. 198–217, https://doi.org/10.1002/nme.5283.