Adaptive SVD for use in Hankel factorisation in a LTV system

Tore Gude
School of Computation, Information and Technology
Technical University of Munich
Munich, Germany
toregud@stud.ntnu.no

***Abstract*—This paper introduces algorithms for the downdating and updating of the Singular Value Decomposition (SVD) of a matrix, A. The downdating handles the case where the first row of A is deleted, while the updating algorithm handles the case where a column is added from the right. These algorithms are applied sequentially resulting in a new SVD that constructs a matrix A' that consists of all but the first row of A with a new column appended from the right. This update and downdate scheme for the SVD is of great interest when dealing with Hankel matrices in a Linear Time-Varying (LTV) system for successive timesteps.**

I. Introduction

The Singular Value Decomposition (SVD) is a fundamental matrix factorization technique and is widely used in numerous numerical applications such as signal processing, computer vision, speech analysis and machine learning. Modern applications require modifying of the decomposition of a matrix as data is being added or removed. This enables real time computation of data as the data becomes available and allows deleting of data from the decomposition which is no longer of interest.

To address these requirements, algorithms have been developed for updating and downdating the SVD based on adding or removing of rows and columns. This paper focus on the updating and downdating of the SVD for a general matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$. The experiments in this paper, use a random matrix generated with the same seed in Matlab.

For both updating and downdating of the SVD, two algorithms are implemented depending on the size of m and n. Specifically, for both downdating and updating, separate algorithms are applied for the case $m \leq n$ and for the case $m > n$.

To the best of my knowledge based on the reviewed literature, no current method exists for simultaneous removal of row and adding of column which are particularly relevant in applications involving the extraction of Hankel matrices from a Toeplitz matrix in a Linear Time-Varying (LTV) system. Neither have I found experiments on running algorithms for row removal in parallel with column addition or the algorithms run successively after each other and analysis of numerical instability and computational efficiency thereof. This paper aims to present experiments conducted on such scenarios and providing insights into the numerical stability and computational efficiency of the proposed methods.

II. Area of Use

In a LTV system, a Toeplitz matrix is utilized in the state space realization of a system. The Toeplitz matrix maps input to output by measuring the impulse responses of a system and entering them along its diagonals. This establish an input-output mapping from $\boldsymbol{u}$ to $\boldsymbol{y}$.

$$\overbrace{\begin{bmatrix} \vdots \\ y_{-2} \\ y_{-1} \\ y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \end{bmatrix}}^{\boldsymbol{y}} = \overbrace{\begin{bmatrix} \ddots & \vdots & \vdots & \vdots & \vdots & \\ \ddots & t_{-1} & t_{-2} & t_{-3} & t_{-4} & \ddots \\ \ddots & t_0 & t_{-1} & t_{-2} & t_{-3} & \ddots \\ \ddots & t_1 & t_0 & t_{-1} & t_{-2} & \ddots \\ \ddots & t_2 & t_1 & t_0 & t_{-1} & \ddots \\ \ddots & t_3 & t_2 & t_1 & t_0 & \ddots \\ & \vdots & \vdots & \vdots & \vdots & \end{bmatrix}}^{\boldsymbol{T}} \overbrace{\begin{bmatrix} \vdots \\ u_{-2} \\ u_{-1} \\ u_0 \\ u_1 \\ u_2 \\ u_3 \\ \vdots \end{bmatrix}}^{\boldsymbol{u}} \tag{1}$$

The lower left part of the $\boldsymbol{T}$ matrix represents the casual part of the system, denoted $\boldsymbol{T_{pf}}$. The impulse responses in this part of the matrix maps past and present input to future and present outputs. The flipped version of the matrix $\boldsymbol{T_{pf}}$ is called a Hankel matrix, denoted as $\boldsymbol{\mathcal{H}}$. The Hankel matrix holds significant importance in analyzing the stability of an LTV system. Throug utilization of the SVD, the Hankel matrix can be factorized into observability and controlability maps in a balanced realization, $\boldsymbol{\mathcal{H}} = \boldsymbol{\mathcal{O}} \cdot \boldsymbol{\mathcal{C}}$.

At the initial timestep the Hankel matrix is formed by considering only the entire first column. The Hankel matrix for the next timestep expands to the two first columns, excluding the first row. This pattern is repeated until the the last row of $\boldsymbol{T_{pf}}$ is reached, assuming finite impulse responses.

This highlights the need to recalculate the SVD using the previous matrices $\boldsymbol{U}$, $\boldsymbol{\Sigma}$ and $\boldsymbol{V}$, when adding a new column, named updating and removing a row named downdating of the SVD.

III. Downdating the SVD

First consider the case of downdating the SVD by removing the first row of the matrix. The paper ”Downdating the Singular Value Decomposition” by Ming Gu and Stanley C. Eisenstad [1] presents a method for downdating the SVD when the last row is removed from the matrix with dimensions $m \leq n$. In this section a modified version of their algorithm is presented, so that the top row of the matrix is removed from the SVD.

$$\begin{bmatrix} \mathbf{a} \\ \mathbf{A}' \end{bmatrix} = \mathbf{U} \cdot \mathbf{\Sigma} \cdot \mathbf{V}^T = \begin{bmatrix} \mathbf{u_1} & \mu \\ \mathbf{U_{11}} & \mathbf{x} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{D} & \mathbf{0} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{V_1}^T \\ \mathbf{V_2}^T \end{bmatrix} \tag{2}$$

which gives

$$\mathbf{a} = \mathbf{u_1} \cdot \mathbf{D} \cdot \mathbf{V_1}^T \tag{3a}$$

$$\mathbf{A}' = \begin{bmatrix} \mathbf{U_{11}} & \mathbf{x} \end{bmatrix} \cdot \mathbf{D} \cdot \mathbf{V_1}^T \tag{3b}$$

The decomposition of $\mathbf{A}'$ is nearly a SVD, as $\begin{bmatrix} \mathbf{U11} & \mathbf{x} \end{bmatrix}$ being close to orthogonal, as it results from deleting the first row of an orthogonal matrix. Further, the orthogonal matrix $\mathbf{X}$ and a simple matrix $\mathbf{C}$ is constructed

$$\mathbf{X} = \mathbf{U_{11}} \left(\mathbf{I} - \frac{1}{1+\mu} \mathbf{u_1}^T \mathbf{u_1} \right) - \mathbf{x} \mathbf{u_1} \tag{4a}$$

$$\mathbf{C} = \begin{bmatrix} \mathbf{I} - \frac{1}{1+\mu} \mathbf{u_1}^T \mathbf{u_1} & -\mathbf{u_1}^T \end{bmatrix} \mathbf{D} \tag{4b}$$

such that $\mathbf{C}$ should require fewer computations to calculate its SVD. Let the SVD of $\mathbf{C}$ be $\mathbf{Q} \begin{bmatrix} \mathbf{\Omega} & \mathbf{0} \end{bmatrix} \mathbf{W}^T$, such that the SVD of $\mathbf{A}'$ becomes

$$\mathbf{A}' = (\mathbf{XQ}) \begin{bmatrix} \mathbf{\Omega} & \mathbf{0} \end{bmatrix} \begin{bmatrix} (\mathbf{V_1 W})^T \\ \mathbf{V_2}^T \end{bmatrix} \tag{5}$$

For further details and verification, please consult the paper by Ming Gu and Stanley C. Eisenstad [1].

The case $m > n$ is based on the paper, ”Downdating the Singular Value Decomposition” by Ming Gu and Stanley C. Eisenstad [2], a report with the same name and the same authors as [1], published two years later. Similar modifications were necessarily to the algorithm presented by the authors as for the case $m \leq n$ to downdate the first row. These modifications are presented here. Further decompose the SVD of A into

$$\mathbf{A} = \mathbf{U \Sigma V}^T = \begin{bmatrix} \mathbf{a}^T \\ \mathbf{A} \end{bmatrix} = \begin{bmatrix} \mathbf{u_1}^T & \mathbf{u_2}^T \\ \mathbf{U_{12}} & \mathbf{U_{22}} \end{bmatrix} \begin{bmatrix} \mathbf{D} \\ \mathbf{0} \end{bmatrix} \mathbf{V}^T \tag{6}$$

Define $\begin{bmatrix} \mathbf{z_2} & \mathbf{X_{12}} \end{bmatrix} = \mathbf{U_{12}} \mathbf{P}^T$, where $\mathbf{P}$ is an orthogonal rotation matrix such that $\mathbf{P u_2} = ||\mathbf{u_2}|| \mathbf{e_1}$, where $\mathbf{e_1} = \begin{bmatrix} 1 & 0 & \dots & 0 \end{bmatrix}^T$. The construction of matrix $\mathbf{P}$ is not shown in the paper by Ming Gu and Stanley C. Eisenstad, so in this project this implementation were used.

$$\mathbf{P} = \left(\mathbf{I} - 2 \cdot \frac{\mathbf{v v}^T}{\mathbf{v}^T \mathbf{v}} \right) \tag{7}$$

where $\mathbf{v} = \frac{\mathbf{u_2}}{||\mathbf{u_2}||} - \begin{bmatrix} 1 & 0 & \dots & 0 \end{bmatrix}^T$

The subsequent steps follow a straightforward and similar process as in the case $m \leq n$. Interested readers are encouraged to consult the paper [2] The algorithm also construct a simple matrix $\mathbf{C}$ and taken the SVD of.

It is worth noting that the algorithm turned out to be unstable for $m = n$. In this case, the vector which is to be rotated is in a fact a scalar, and rotation of a scalar without an orientation is meaningless. Therefore to avoid this instability, the rotation procedure for this scalar is simply excluded. The final step involves constructing a simple matrix $\mathbf{C}$ and computing its SVD, as in the case $m \leq n$

IV. Updating the SVD

The method used for updating the SVD in this paper is based on the paper ”A stable and fast algorithm for updating the Singular Value Decomposition” by Ming Gu and Stanley C. Eisenstad [3], the same authors which presented the algorithms used for downdating the SVD. The method presented in the paper by Gu and Eisenstad update the SVD when a row is added at the bottom of a matrix. For this project modifications were made to the algorithm so that the data were added as a column at the right side of the matrix. This was done exploiting the fact that appending a column to $\mathbf{A}$ is tantamount to appending a row to $\mathbf{A}^T$ so that

$$\mathbf{A}^T = \left(\mathbf{U \Sigma V}^T \right)^T = \mathbf{V \Sigma}^T \mathbf{U}^T \tag{8}$$

Let’s first consider the case $m \leq n$. This case is straight forward where $\mathbf{A}' = \begin{bmatrix} \mathbf{A} & \mathbf{a} \end{bmatrix} = \begin{bmatrix} \mathbf{U \Sigma V}^T & \mathbf{a} \end{bmatrix}$. The matrix $\mathbf{A}$ is then decomposed into $\mathbf{V} = \begin{bmatrix} \mathbf{V_1} & \mathbf{V_2} \end{bmatrix}$ and $\mathbf{\Sigma} = \begin{bmatrix} \mathbf{D} & \mathbf{0} \end{bmatrix}$, and set $\mathbf{z} = \mathbf{a}^T \mathbf{U}$.

Then the SVD is taken of the simple matrix

$$\mathbf{C} = \begin{bmatrix} \mathbf{D} \\ \mathbf{z} \end{bmatrix} \tag{9}$$

and the results here are used to construct the updated version of the SVD.

The case where $m > n$ is more complicated and also involve the orthogonal reflection in equation 7, but here the rotated vector is the norm of $\mathbf{z_2}$, given in equation 10b. Also in this paper the method of rotation is left undetermined and in this project the same structure of a rotation matrix as in equation 7 is used. Modifications made to the algorithm are presented below

$$\mathbf{z_1} = \mathbf{U_1 a}^T \tag{10a}$$

$$\mathbf{z_2} = \mathbf{U_2 a}^T \tag{10b}$$

$$\mathbf{U} = \begin{bmatrix} \mathbf{U_1} & \mathbf{U_2} \end{bmatrix} \tag{10c}$$

$$\mathbf{\Sigma} = \begin{bmatrix} \mathbf{D} \\ \mathbf{0} \end{bmatrix} \tag{10d}$$

Next, the simple matrix C is constructed and its SVD is computed, similar to the other three algorithms.

$$\mathbf{C}\begin{bmatrix}\mathbf{D} & \mathbf{0} \\ \mathbf{z_1}^T & ||\mathbf{z_2}||\end{bmatrix} \tag{11}$$

The rest of the algorithm is matrix manipulations and multiplications along with rotation of the vector $\mathbf{v_2}$ and interested readers are advised to the literature for further insight into the algorithm.

V. SVD of $\mathbf{C}$

All four algorithms presented utilize the SVD of the matrix $\mathbf{C}$, which is close to being a diagonal matrix. Theoretically, computing the SVD of this matrix requires less computations compeared to a dense matrix of same dimensions. The papers introduces methods for calculating the SVD of this matrix, however the attempts of implementing these algorithms in Matlab yielding satisfying results in terms of accuracy were unsuccessful. Consequently, the SVD of $\mathbf{C}$ is taken using the Matlab function `svd`, which provides accurate results for the analysis of reconstruction error presented in this paper. As the Matlab function `svd` is not optimized for taking the SVD of a matrix with as simple structure as $\mathbf{C}$, the results concerning computational complexity are biased and not representative of the computational complexity for the respective algorithms.

In theory, performing a single update of a SVD requires approximately $O\left((m+n)\min(m,n)\log_2^2(\epsilon)\right)$ floating point operations, where ϵ is machine precision. The algorithms for downdating can be executed in $O\left((m+n)\min(m,n)^2\log_2^2(\epsilon)\right)$. However, according to the same paper, these algorithms can be accelerated to the same computational complexity as the case for updating the SVD utilizing a fast multipole method. The computational complexity of the Matlab function `svd` is $O\left(\max(m,n)\min(m,n)^2\right)$

VI. Similar methods

There are also other and more recent algorithms to perform the updating and downdating of the SVD than the ones presented in this article. One method of performing updating of the SVD is the rank-one update. The rank one update is used when a rank one matrix is added or subtracted to the matrix $\mathbf{A}$.

$$\mathbf{A}' = \mathbf{A} + \mathbf{a}\mathbf{b}^T = \mathbf{U}^T + \mathbf{a}\mathbf{b}^T = \mathbf{U}'\mathbf{\Sigma}'\mathbf{V}'^T \tag{12}$$

One implementation of these method can be found in the paper ”On the Efficient Update of the Singular Value Decomposition Subject to Rank-One Modifications” written by Peter Stange [4]. As rank-one updates not directly change the dimensions of the existing matrix, $\mathbf{A}$, these methods were not used in this project, but are an important method in many applications such as image/video compression, signal processing and pattern recognition etc. Rank one update is specifically useful when data is distributed over a network of devices and you need a real time update of the SVD when more data arrives, as in the case of for example live streaming.

In the paper ”Fast low-rank modifications of the thin singular value decomposition” by Matthew Brand [5], a method is proposed for using rank-one update when a column is added to a thin matrix. To update the SVD of $\mathbf{A}'$ with the new column, $\mathbf{c}$ added, a row of zeros is appended to $\mathbf{V}$ and then then the rank-one modification is computed

$$\mathbf{U}'\mathbf{\Sigma}'\mathbf{V}'^T = \begin{bmatrix}\mathbf{A} & \mathbf{0}\end{bmatrix} + \mathbf{c}\begin{bmatrix}0 & 0 & \cdots & 0 & 1\end{bmatrix} \tag{13}$$

There also exists methods for updating the SVD which is based on the simpler problem of updating the QR decomposition, as presented in ”A Singluar Value Decomposition updating algorithm for subspace tracking” by Marc Moonen et.al [6]. The paper claims that only a few steps involving Jacobi-type SVD procedure after each QR update is sufficient to obtain an acceptable approximation of the SVD when a row is added to a matrix.

The algorithms in this project were chosen based upon their basis in linear algebra and that the methods for updating and downdating the SVD based on the same principles. For both the projects and the readers convenience, this were meant to be beneficial for understanding of the algorithms.

VII. Results

Testing of the implemented algorithms were run on a square matrix, where the SVD were calculated iterative from the first column all the way until the last row, successively removing a row then adding a column. This section is divided into three subsections, where small, medium and large matrices are considered, and the matrix $\mathbf{A}$ originally is square with the size 100, 500 and 1000 rows and columns, respectively. It should be stated that the four algorithms produce different matrices $\mathbf{C}$, which theoretically require less computational power to compute the SVD of compared to a non structured matrix of the same size. During these experiments, the SVD of both the matrix $\mathbf{C}$ and the original SVD of $\mathbf{A}$ calculated from scratch, were calculated with the implemented Matlab function `svd`. This function is a highly optimized function and doesn't consider the easier structure of the matrix $\mathbf{C}$. The presented results based on timing is therefore, as also discussed in Section V, not an accurate presentation of the computational cost of these algorithms, but is still interesting to analyze. The results and analysis around the accuracy of the method is however accurate and interesting.

A. Accuracy analysis

The accuracy analysis is performed by finding the maximum absolute value in the difference of the reconstructed matrix $\mathbf{A}'$ from the updated/downdated versions of $\mathbf{U}'$, $\mathbf{\Sigma}'$ and $\mathbf{V}'$ and the true submatrix from $\mathbf{A}$. This were done due to the fact that the choice of $\mathbf{U}$ and $\mathbf{V}$ are not unique so direct comparison of the matrices forming the SVD were not possible. Figure 1, 2 and 3 shows the semi-logarithmic plot of the corresponding largest error for respectively a small, medium and large matrix as defined.

The maximum error for all three cases are at machine precision in magnitude, 10^{-15}, for the first iteration, before the error drastically increases with about a magnitude of 10^2 the next few iterations. Interestingly, the error stabilize at this magnitude for the next iterations, regardless of how many iterations before drastically increasing over the last few iterations increasing to a magnitude of respectively 10^{-12} for the small and medium matrices and 10^{-11} for the big matrix.

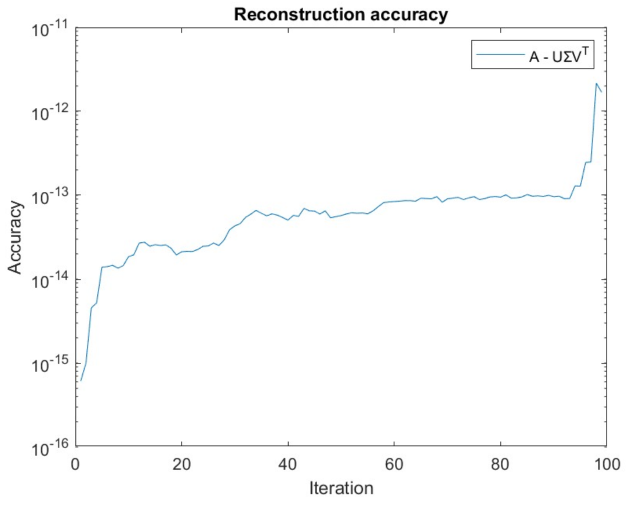

Fig. 1. Accuracy of reconstructed matrix of dimension 100 × 100

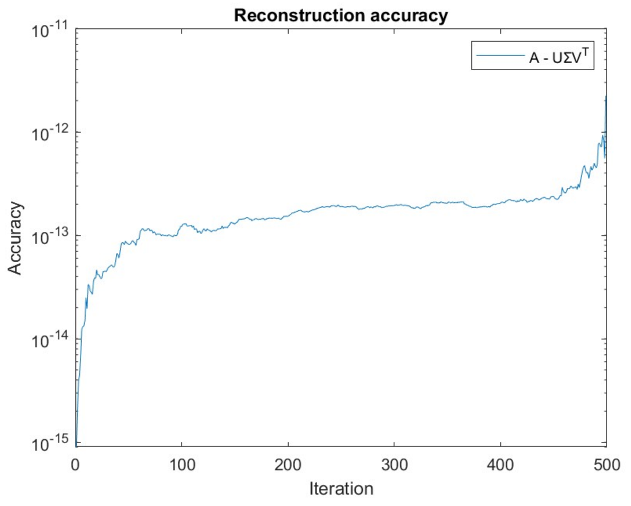

Fig. 2. Accuracy of reconstructed matrix of dimension 500 × 500

The algorithms are proven to be stable and well-conditioned in the literature, supported by the slowly increasing error during the middle iterations. The slowly increasing error in this section may be due to machine precision. However, the algorithms all seems ill-conditioned in terms of numerical stability for the cases where the matrix is either tall or wide, the first and last few iterations. The reconstruction error during these periods are rapidly increasing questioning the numerical stability of these algorithms.

B. Time analysis

As previously discussed in Section V, the timing analysis done in this project doesn't reflect the real benefit of the algorithms as no additional way of calculating the simple matrix $\mathbf{C}$ are implemented in the code. To take this project further, one should look into ways to do so. The attempts for this made during this project all turned out to yield non-satisfying result for the SVD of $\mathbf{C}$. As a result the Matlab function `svd`, which doesn't account for the simple structure of $\mathbf{C}$ were used. This is definitely a source of error and corrupts the result. The Matlab code is also run on a Intel core i7 7th processor and the results are believed to be sped up running on a faster computer. However, the results are presented as time taken per calculation of SVD, in means of the updated SVD of a matrix with row removal and column addition.

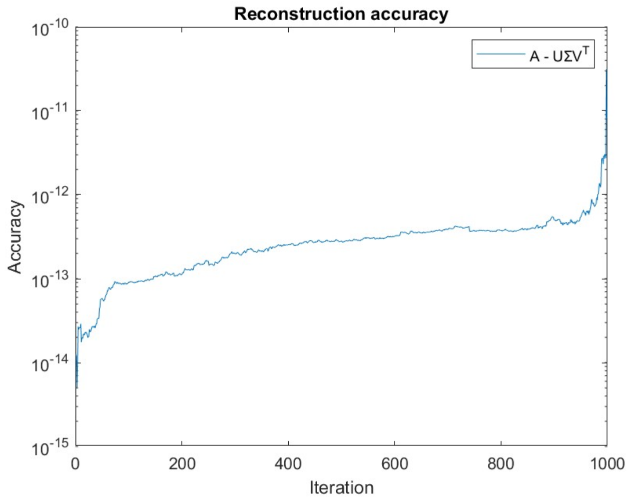

Fig. 3. Accuracy of reconstructed matrix of dimension 1000 × 1000

Figure 4 shows the time consumed for each iteration computing the `svd` of the successive matrices with row removal and column addition. As expected, the results are quite noisy as the experiment is run on a normal computer. Total time consumed for 500 iterations were in the case of the implemented Adaptive SVD algorithms 5.1319 seconds and for the Matlab function `svd` total time consumed were 2.7020, about twice as fast as for the Adaptive SVD case. The time difference for the methods for each iteration is also included in Figure 4 and peaks at iteration 250, excluding outliers, when the number of columns and rows are equal and the matrix in consideration is a square 250 × 250 matrix. The difference in time also follows a similar shape to the timing of the Matlab `svd` function, indicating that the Matlab `svd` function takes about the same amount of computations regardless of the structure of the input matrix. As discussed in Section V and earlier in this section, the results could have been different, as the timing complexity of the Adaptive SVD algorithms in theory is lower. The order of removal and addition of columns didn't show any noticeable difference in computational complexity and neither in the total execution time nor reconstruction accuracy. For this experiment, row removal is done prior to column addition. The results are only shown for the experiment done on a medium

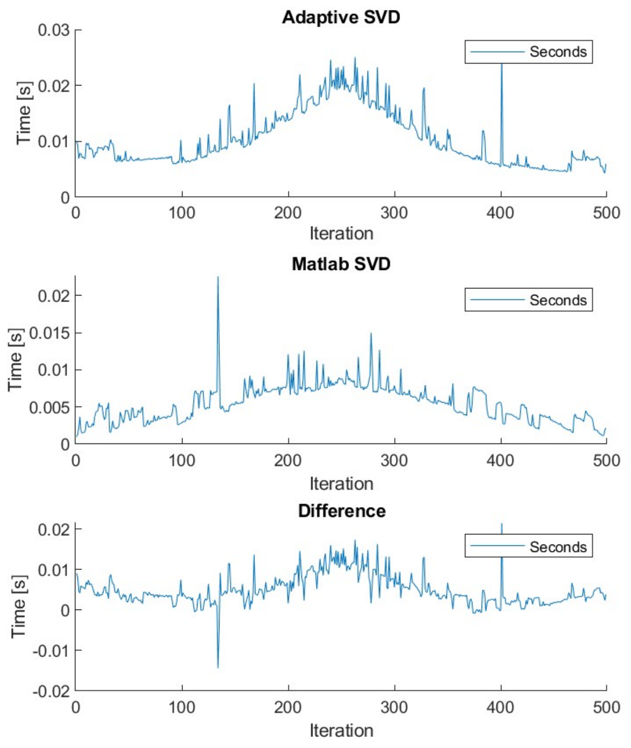

Fig. 4. Time consumed per `svd` iteration of a matrix of dimension 500 $\times$ 500

matrix, as similar results were obtained for the experiments on small and large matrices, also resulting in about twice the computational complexity for the adaptive SVD algorith.

C. *Reconstruction with noisy input*

When given noisy input, the adaptive SVD algorithm actually derives a more accurate solution over the iterations, as shown in Figure 5. Noise were added to the input matrices $\mathbf{U}$, $\mathbf{\Sigma}$ and $\mathbf{V}$. The noise added were a uniformly distributed matrix of corresponding size using Matlab's `randn` function scaled by a factor of 0.01.

The error reduction in Figure 5 is interesting to observe as it shows that the Adaptive SVD algorithms suppresses the noise instead of accumulating it before the error rises again at the last few iterations as were also the case when no noise were present. One reasonable explanation for this is that as more and more of the original matrix is added through columns, the input noise becomes smaller with regard to the size of the matrix. The noise is also quite small, with experiments did with the scale factor in the magnitude of 0.1, the reconstruction error grew during the iterations. It is yet interesting, the ability of the Adaptive SVD algorithm to suppress noise in the input data and construct a more accurate reconstruction of the original matrix as more data from the original matrix becomes available.

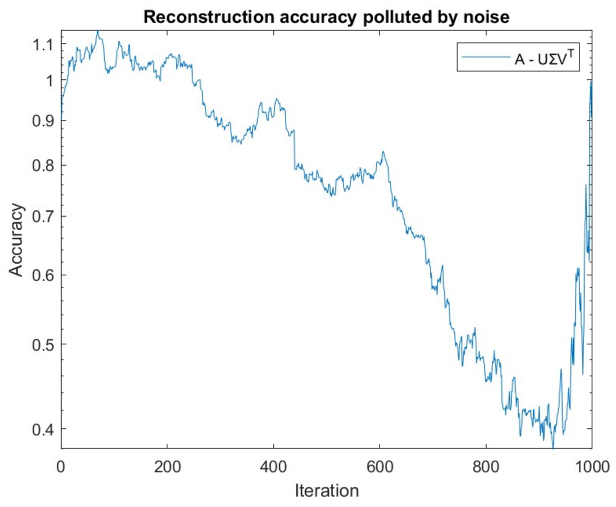

Fig. 5. Reconstruction of a matrix of dimension 1000 $\times$ 1000 given noisy inputs

VIII. GitHub implementation

The implementation of the algorithms are made available on GitHub for the readers interest. Instructions on how to run the files and explanations of the files are given in the ReadMe file.

References

[1] Gu, M. & Eisenstat, S. C. (1993). Downdating the Singular Value Decomposition. Yale University, Research Report YALEU/DCS/RR-966, Dept. of Computer Science, Yale University, New Haven, CT.

[2] Gu, M. & Eisenstat, S. C. (1995). DOWNDATING THE SINGULAR VALUE DECOMPOSITION. *SIAM Journal on Matrix Analysis and Applications* 16(3), 793-810.

[3] Gu, M. & Eisenstat, S. C. (1993). A Stable and Fast Algorithm for Updating the Singular Value Decomposition. Yale University, Research Report YALEU/DCS/TR966, Dept. of Computer Science, Yale University, New Haven, CT.

[4] Stange, P. (2008). On the Efficient Update of the Singular Value Decomposition. PAMM 8(1) 10827-10828

[5] Brand, M. (2006). Fast low-rank modifications of the thin singular value decomposition. *Linear Algebra and its Applications* 415(1), 20-30.

[6] Moonen, M., Van Dooren, P., & Vandewalle, J. (1992). A singular value decomposition updating algorithm for subspace tracking. *SIAM Journal on Matrix Analysis and Applications* 13, 1015-1038.