Electronic Transactions on Numerical Analysis.
Volume 58, pp. 164–176, 2023.

ISSN 1068–9613.
DOI: 10.1553/etna_vol58s164

ETNA
Kent State University and
Johann Radon Institute (RICAM)

EXPLICIT DEFLATION IN GOLUB-KAHAN-LANCZOS BIDIAGONALIZATION METHODS*

JAMES BAGLAMA† AND VASILIJE PEROVIĆ†

Abstract. We discuss a simple, easily overlooked, explicit deflation procedure applied to Golub-Kahan-Lanczos Bidiagonalization (GKLB)-based methods to compute the next set of the largest singular triplets of a matrix from an already computed partial singular value decomposition. Our results here complement the vast literature on this topic, provide additional insight, and highlight the simplicity and the effectiveness of this procedure. We demonstrate how existing GKLB-based routines for the computation of the largest singular triplets can be easily adapted to take advantage of explicit deflation, thus making it more appealing to a wider range of users. Numerical examples are presented including an application of singular value thresholding.

Key words. Lanczos bidiagonalization, (partial/truncated) singular value decomposition, deflation, thresholding

AMS subject classifications. 65F15, 65F50, 15A18

1. Introduction. An *s-partial singular value decomposition (s-PSVD)* of $A \in \mathbb{R}^{\ell\times n}$ with $s \ll \operatorname{rank}(A)$ is given by

$$A\,V_s \,=\, U_s\,\Sigma_s\,, \qquad A^T U_s \,=\, V_s\,\Sigma_s, \tag{1.1}$$

where $U_s = [u_1, u_2, \ldots, u_s] \in \mathbb{R}^{\ell\times s}$ and $V_s = [v_1, v_2, \ldots, v_s] \in \mathbb{R}^{n\times s}$ are matrices with orthonormal columns and $\Sigma_s = \operatorname{diag}(\sigma_1, \ldots, \sigma_s) \in \mathbb{R}^{s\times s}$ with $\sigma_1 \geq \ldots \geq \sigma_s > 0$. For $j = 1, \ldots, s$, the *singular triplets* of A are denoted by $\{\sigma_j, u_j, v_j\}$, and we refer to the largest singular triplets as those associated with the largest singular values. The prominent role that the SVD occupies in scientific computing can be attributed to two facts: very diverse application areas and the development of efficient numerical methods for its computation. Matrices arising in applications such as principal component analysis (PCA) [18], genomics [1, 32], data mining, data visualization, machine learning, pattern recognition [13], and directed networks [4], are often very large, sparse, and only accessible via matrix-vector routines, thus making it impractical for the computation of all singular triplets. Fortunately, with such matrices, one is often interested in computing only a few of the largest singular triplets—this has spurred a considerable amount of research (see, e.g., [6, 7, 19, 20, 21, 24, 25, 31, 33] and the references therein). This paper also deals with the computation of the largest singular triplets, though our starting point here is different when compared to the previously listed references. More specifically, we are interested in the following problem:

"Using a GKLB-based method and given the k_0 largest singular triplets of A, determine the next set of $\overline{k}_1$ largest singular triplets. That is, expand the given k_0-PSVD of A into a k_1-PSVD, where $k_1 = k_0 + \overline{k}_1$."

One could solve this problem by simply (re)computing all $k_1 = k_0 + \overline{k}_1$ largest singular triplets from scratch. We do not consider this approach here since it can be unnecessarily expensive and it takes no account of the available k_0-PSVD of A. Moreover, in applications where the number of needed singular triplets is not known in advance, one often has to solve this problem multiple times. For example, one approach for solving large-scale *linear discrete ill-posed* problems, $\min_{x\in\mathbb{R}^n} \|Ax - b\|$ with A being very ill-conditioned, relies on solving a smaller least-squares problem associated with an s-PSVD of A (1.1), where s is suitably

*Received January 13, 2023. Accepted January 20, 2023. Published online on February 7, 2023. Recommended by L. Reichel.

†Department of Mathematics and Applied Mathematical Sciences, University of Rhode Island, 02881, USA ({jbaglama, perovic@}uri.edu).

ETNA
Kent State University and
Johann Radon Institute (RICAM)

chosen in order to satisfy the discrepancy principle; see for example [27] and the references therein. But the index s is typically not known in advance, in which case one may resort to computing the largest singular triplets in increments until the desired s is determined. Another example where the number of desired singular triplets is not known in advance is the problem of *singular value thresholding*, which aims to determine all singular triplets of a matrix A with singular values exceeding a certain threshold—this in fact can be achieved by solving our proposed problem time after time until a singular value that falls below the threshold is identified. There are numerous applications where singular value thresholding arises, e.g., matrix completion problems [11], the identification of highly-correlated pairs of vectors in various machine learning algorithms [5], and the analysis of directed networks [4], and consequently, finding an efficient method for solving it will have an immediate impact.

One approach to determining the next set of the $\overline{k}_1$ largest singular triplets of A, given that the k_0 largest singular triplets are already known, is to transform the singular value problem into an equivalent symmetric eigenvalue problem associated with the matrices A^TA, AA^T, and $C = \left[\begin{smallmatrix} 0 & A \\ A^T & 0 \end{smallmatrix}\right]$ [15]. For example, the additional $\overline{k}_1$ largest singular triplets can be computed by solving the symmetric eigenvalue problem applied to $(I - V_{k_0}V_{k_0}^T)A^TA$ or $A^TA - V_{k_0}\Sigma_{k_0}^2 V_{k_0}^T$, [15, 23, 24, 28], or, applied as a subspace restriction technique, by orthogonalizing generated basis vectors against converged eigenvectors; see also [3, 28, 30]. In [22], the authors developed a MATLAB wrapper routine `svt`[1], which leverages MATLAB's `eigs` function and repeatedly applies an explicit deflation technique on C to compute additional sets of largest eigenpairs of C, and hence largest singular triplets of A, until a singular value below a user-specified threshold is found. It is the purpose of this note to apply a deflation process directly to A without the need to transform the problem to the equivalent symmetric eigenvalue problem. That is, in this paper, we use an explicit deflation of subspace restriction applied to methods based on the Golub-Kahan-Lanczos Bidiagonalization (GKLB) procedure [14] and thus eliminating the need to consider the equivalent symmetric eigenproblem as in [22]. Furthermore, the explicit deflation in many cases can be applied in the GKLB process on one-side only, thus reducing the overall computational cost with an often modest, manageable error growth. There are numerous GKLB-based methods, one of the most popular being the efficient thick-restarted GKLB algorithm often referred to as IRLBA [7]. The ubiquitousness of the IRLBA method for large matrices/data sets [32] is most evident by the existence of its robust implementations in numerous programming languages, e.g., in R code `irlba` [21], in Python code `irlbapy`, in MATLAB syntax code `irlba`, and in C++ code `Cppirlba`.[2] Moreover, starting in 2016, the MATLAB internal function `svds` [31] references the IRLBA method [7]. With the exception of the R code `irlba`, none of the other listed software packages have the option to input singular vectors to deflate computed singular triplets. However, at this time, the current version of the R code `irlba` does not take advantage of the analysis provided here.

The process of explicit deflation with subspace restriction for the GKLB process, although quite natural, does not appear to be well-investigated in the literature. We do remark that `svdifp` [24] and `rd2svds` [6] compute only one singular triplet at a time and apply an internal explicit deflation techniques to compute additional singular triplets. These differ from the focus of this paper. For discussions on techniques for internal deflation (locking/purging) while computing the $\overline{k}_1$ singular triplets, see for example [6, 9, 19, 24].

In Section 2 we show how the next set of the $\overline{k}_1$ largest singular triplets can be computed from an already existing k_0-PSVD (1.1) via the d-GKLB Algorithm 1 and how "one-sided"

---

[1] Code available at: https://github.com/Hua-Zhou/svt. Retreived on November 30, 2022.

[2] R `irlba`: https://github.com/bwlewis/irlba, `irlbapy`: https://github.com/bwlewis/irlbpy, MATLAB syntax `irlba`: http://www.netlib.org/numeralgo/na26.tgz, `Cppirlba`: https://github.com/LTLA/CppIrlba

explicit deflation can be used, and we provide a short analysis of the propagated error. Furthermore, like the approach in [22], the process can be implemented without modifying the existing codes by only requiring a simple modification to the matrix-vector product routines leading to the development of software wrapper routines like `svt` for threshold computations. We illustrate this approach using MATLAB's `svds` [31] and a modified version of `irlba` [2, 7] given in Appendix A. Numerical examples illustrating the error analysis and thresholding are presented in Section 3 along with concluding remarks in Section 4.

2. The deflated GKLB procedure. The simple explicit deflation process described below is a subspace restriction technique applied to the GKLB basis vectors and translates to only needing to modify the matrix-vector product routines without internal algorithmic modifications. Even though it is not essential (see Remark 2.1), we assume that the initial k_0-PSVD of A is obtained from the non-deflated m-GKLB factorization (2.1)–(2.2) with $U_{k_0} = 0$, $V_{k_0} = 0$, and that all subsequent k-PSVD's of A are obtained from the deflated m-GKLB factorization (2.1)–(2.2). We also make two common assumptions about the GKLB process:

(a) On each restart cycle, the methods produce an m-GKLB factorization (2.1)–(2.2) where orthogonality is maintained to the desired accuracy among the generated basis vectors and also with the converged singular vectors [7, 10, 20].

(b) The d-GKLB Algorithm 1 does not breakdown, i.e., $\alpha_i > 0$ and $\beta_i > 0$, so that one can build an m-GKLB factorization (2.1)–(2.2); see [7] for details. We provide further remarks on breakdowns in Remark 2.2.

Suppose a k_0-PSVD of A from a GKLB-based method is given. Then the d-GKLB Algorithm 1 produces

$$(I - U_{k_0}U_{k_0}^T)AP_m = Q_m B_m \quad \Longleftrightarrow \quad AP_m = Q_m B_m + U_{k_0}U_{k_0}^T AP_m, \tag{2.1}$$

$$(I - V_{k_0}V_{k_0}^T)A^T Q_m = P_m B_m^T + f_{\bar{k}_1} e_m^T \quad \Longleftrightarrow \quad A^T Q_m = P_m B_m^T + f_{\bar{k}_1} e_m^T + V_{k_0}V_{k_0}^T A^T Q_m, \tag{2.2}$$

where $P_m \in \mathbb{R}^{n\times m}$, $Q_m \in \mathbb{R}^{\ell\times m}$, $P_m^T P_m = Q_m^T Q_m = I_m$, $P_m^T f_{\bar{k}_1} = 0$, $V_{k_0}^T P_m = 0$, $V_{k_0}^T f_{\bar{k}_1} = 0$, $U_{k_0}^T Q_m = 0$, and $B_m \in \mathbb{R}^{m\times m}$ is an upper bidiagonal matrix (cf. steps 5, 9, 11).

Algorithm 1 Deflated GKLB (d-GKLB).

1: **Input:** A; p_1; U_{k_0}, V_{k_0};[§] m.
2: **Output:** $P_m, Q_m, B_m, f_{\bar{k}_1}$ (see (2.1)–(2.2))

3: $p_1 := (I - V_{k_0}V_{k_0}^T)p_1$; $P_1 := p_1/\|p_1\|$;
4: $q_1 := (I - U_{k_0}U_{k_0}^T)Ap_1$;
5: $\alpha_1 := \|q_1\|$; $q_1 := q_1/\alpha_1$; $Q_1 := q_1$;
6: **for** $j = 1 \ldots m$ **do**
7: $\quad f_{\bar{k}_1} := (I - V_{k_0}V_{k_0}^T)A^T q_j - \alpha_j p_j$;[§§]
8: $\quad$ **if** $j < m$ **then**
9: $\quad\quad \beta_j := \|f_{\bar{k}_1}\|$; $p_{j+1} := f_{\bar{k}_1}/\beta_j$;
10: $\quad\quad q_{j+1} := (I - U_{k_0}U_{k_0}^T)Ap_{j+1} - \beta_j q_j$;
11: $\quad\quad \alpha_{j+1} := \|q_{j+1}\|$; $q_{j+1} := q_{j+1}/\alpha_{j+1}$;
12: $\quad\quad P_{j+1} := [P_j, p_{j+1}]$;
13: $\quad\quad Q_{j+1} := [Q_j, q_{j+1}]$;
14: $\quad$ **end if**
15: **end for**

[§]If $k_0 = 0$, then $U_{k_0} := 0$ and $V_{k_0} := 0$.
[§§]Can be replaced by $f_{\bar{k}_1} := A^T q_j - \alpha_j p_j$; (see (2.9)).

Once an m-GKLB factorization is available, a variety of projected approximations can be used to estimate the singular triplets of A, e.g., Ritz methods [6, 7, 19, 20, 21, 31], harmonic Ritz methods [7], (iterative) refined Ritz and refined harmonic Ritz methods [6, 17, 19]. In this paper, in order to keep our analysis short, we focus on the Ritz method as one of the most commonly used approximation for the largest singular triplets.

If $\{\sigma_i^{(B_m)}, u_i^{(B_m)}, v_i^{(B_m)}\}$ is the largest computed singular triplet of B_m, then the *approximate* largest singular triplet of A, $\{\widetilde{\sigma}_i^{(A)}, \widetilde{u}_i^{(A)}, \widetilde{v}_i^{(A)}\}$, is given by

$$\widetilde{\sigma}_i^{(A)} := \sigma_i^{(B_m)}, \qquad \widetilde{u}_i^{(A)} := Q_m u_i^{(B_m)}, \qquad \text{and} \qquad \widetilde{v}_i^{(A)} := P_m v_i^{(B_m)}.$$

We use the matrices

$$\Sigma_{\overline{k}_1}^{(B_m)} := \operatorname{diag}(\sigma_1^{(B_m)}, \ldots, \sigma_{\overline{k}_1}^{(B_m)}), \qquad U_{\overline{k}_1}^{(B_m)} := [u_1^{(B_m)}, \ldots, u_{\overline{k}_1}^{(B_m)}], \qquad \text{and}$$
$$V_{\overline{k}_1}^{(B_m)} := [v_1^{(B_m)}, \ldots, v_{\overline{k}_1}^{(B_m)}]$$

to denote the $\overline{k}_1$-PSVD of the projected problem while their counterparts without the superscript (B_m) give us the next largest $\overline{k}_1$ singular triplets of A, i.e.,

$$\Sigma_{\overline{k}_1} = \Sigma_{\overline{k}_1}^{(B_m)}, \qquad U_{\overline{k}_1} := Q_m U_{\overline{k}_1}^{(B_m)}, \qquad V_{\overline{k}_1} := P_m V_{\overline{k}_1}^{(B_m)}. \tag{2.3}$$

REMARK 2.1. It is important to highlight that our subsequent developments are reliant on starting with and having the GKLB-like structure outlined in this paper. If this is not the case, e.g., if the initial k_0-PSVD is not obtained via non-deflated GKLB or the computed singular vectors do not have strong orthogonality, then a single, potentially expensive, pre-(post-)processing step enables us to still apply results from here. More specifically, if given $V \in \mathbb{R}^{n\times k}$ and after securing $V^TV = I$, one can then compute the SVD of AV, i.e.,

$$(AV)\bar{V} = \bar{U}\bar{\Sigma} \qquad \text{and} \qquad (AV)^T\bar{U} = \bar{V}\,\bar{\Sigma},$$

where $\bar{U} \in \mathbb{R}^{\ell\times k}$, $\bar{V} \in \mathbb{R}^{k\times k}$, and $\bar{\Sigma} \in \mathbb{R}^{k\times k}$, and we set $V := V\bar{V}$, $U := \bar{U}$, and $\Sigma := \bar{\Sigma}$. But this now resembles the structure of a GKLB-based k-PSVD approximation, where $AV = U\Sigma$ and $A^TU = V\Sigma + (I - VV^T)A^TU$; see [16, 26] for details on one-sided SVD projections.

We start by taking a closer look into the relations (2.1)–(2.2). Starting at the initial stage using the d-GKLB Algorithm 1 and assuming we are interested in computing the $\overline{k}_0$ largest singular triplets and that no singular triplets of A are known in advance, we obtain the k_0-PSVD of A, with $k_0 = 0 + \overline{k}_0$, by post-multiplying (2.1) and (2.2) by $V_{\overline{k}_0}^{(B_m)}$ and $U_{\overline{k}_0}^{(B_m)}$, respectively, to obtain

$$AV_{\overline{k}_0} = U_{\overline{k}_0}\Sigma_{\overline{k}_0} + \mathcal{V}_{\overline{k}_0}^{err} \qquad \text{and} \qquad A^TU_{\overline{k}_0} = V_{\overline{k}_0}\Sigma_{\overline{k}_0} + \mathcal{U}_{\overline{k}_0}^{err}, \tag{2.4}$$

where in exact arithmetic $\mathcal{V}_{\overline{k}_0}^{err} = 0$ and $\mathcal{U}_{\overline{k}_0}^{err} := f_{\overline{k}_0} e_m^T U_{\overline{k}_0}^{(B_m)}$ (or as given in Remark 2.1). For the purpose of staying consistent with our later notation, we let

$$\Sigma_{k_0} := \Sigma_{\overline{k}_0}, \quad U_{k_0} := U_{\overline{k}_0}, \quad V_{k_0} := V_{\overline{k}_0}, \quad \mathcal{V}_{k_0}^{err} := \mathcal{V}_{\overline{k}_0}^{err}, \quad \text{and} \quad \mathcal{U}_{k_0}^{err} := \mathcal{U}_{\overline{k}_0}^{err}$$

so that (2.4) becomes

$$AV_{k_0} = U_{k_0}\Sigma_{k_0} + \mathcal{V}_{k_0}^{err} \qquad \text{and} \qquad A^TU_{k_0} = V_{k_0}\Sigma_{k_0} + \mathcal{U}_{k_0}^{err}. \tag{2.5}$$

Next, for the rest of this paper, we assume that the k_i-PSVD of A, where $k_i = \sum_{j=0}^{i} \overline{k}_j$ with $i > 0$, is determined by starting with no knowledge of any singular triplets of A and by computing the largest singular triplets $\overline{k}_j$ at a time using the d-GKLB Algorithm 1. Analogous to (2.5), we have the factorization

$$AV_{k_i} = U_{k_i}\Sigma_{k_i} + \mathcal{V}_{k_i}^{err} \qquad \text{and} \qquad A^TU_{k_i} = V_{k_i}\Sigma_{k_i} + \mathcal{U}_{k_i}^{err}, \tag{2.6}$$

where, for $j = 1, 2, \ldots, i$, we have

$$\begin{aligned} &\Sigma_{k_j} := \text{diag}\left(\Sigma_{k_{j-1}}, \Sigma_{\overline{k}_j}\right), \quad U_{k_j} := \left[\, U_{k_{j-1}} \;\; U_{\overline{k}_j} \,\right], \quad V_{k_j} := \left[\, V_{k_{j-1}} \;\; V_{\overline{k}_j} \,\right], \\ &\mathcal{V}^{err}_{\overline{k}_j} := U_{k_{j-1}} U^T_{k_{j-1}} A V_{\overline{k}_j}, \qquad \mathcal{U}^{err}_{\overline{k}_j} := f_{\overline{k}_j} e^T_m U^{(B_m)}_{\overline{k}_j}, \\ &\mathcal{V}^{err}_{k_j} := \left[\, \mathcal{V}^{err}_{k_{j-1}} \;\; \mathcal{V}^{err}_{\overline{k}_j} \,\right], \qquad \mathcal{U}^{err}_{k_j} := \left[\, \mathcal{U}^{err}_{k_{j-1}} \;\; \mathcal{U}^{err}_{\overline{k}_j} \,\right]. \end{aligned} \tag{2.7}$$

Observe that the term $V_{k_{j-1}} V^T_{k_{j-1}} A^T Q_m U^{(B_m)}_{\overline{k}_j}$ is absent in $\mathcal{U}^{err}_{\overline{k}_j}$ (2.7), and we now argue that in exact arithmetic it is zero. From (2.6) we have that $V^T_{k_i} A^T = \Sigma_{k_i} U^T_{k_i} + \mathcal{V}^{err\,T}_{k_i}$, which together with (2.2), gives us

$$\begin{aligned} V_{k_i}(V^T_{k_i} A^T) Q_m &= V_{k_i}(\Sigma_{k_i} U^T_{k_i} Q_m + \mathcal{V}^{err\,T}_{k_i} Q_m) \\ &\overset{(a)}{=} V_{k_i}(0 + \mathcal{V}^{err\,T}_{k_i} Q_m) \overset{(b)}{=} V_{k_i}(0+0) \;=\; 0\,, \end{aligned} \tag{2.8}$$

where (2.8(a)) is due to the orthogonality requirement of the q_j's with U_{k_i} in the d-GKLB Algorithm 1 (cf. steps 4,10). Analogously, $\mathcal{V}^{err\,T}_{k_i} Q_m = 0$ in (2.8(b)) since

$$\mathcal{V}^{err}_{k_i} = \left[\mathcal{V}^{err}_{\overline{k}_0} \;\; \ldots \;\; \mathcal{V}^{err}_{\overline{k}_i}\right] \quad \text{and} \quad \mathcal{V}^{err\,T}_{\overline{k}_j} Q_m = V^T_{\overline{k}_j} A^T U_{k_{j-1}} U^T_{k_{j-1}} Q_m = 0,$$

for $j = 0, 1, \ldots, i$. Thus,

$$\left[(I - U_{k_i} U^T_{k_i}) A\right]^T Q_m = A^T (I - U_{k_i} U^T_{k_i}) Q_m = A^T Q_m = \left[(I - V_{k_i} V^T_{k_i}) A^T\right] Q_m. \tag{2.9}$$

Note that in exact arithmetic, (2.9) is trivially satisfied since

$$[(I - U_{k_i} U^T_{k_i}) A]^T = (I - V_{k_i} V^T_{k_i}) A^T$$

and that the explicit deflation $(I - U_{k_i} U^T_{k_i}) A$ has a simple effect on the singular triplets of A, namely it moves the largest k_i singular values of A to zero and leaves the rest unchanged.

Returning back to (2.9), we observe that given the k_i-PSVD of A, the output of the d-GKLB Algorithm 1 simply corresponds to the standard GKLB process operating on the deflated matrix $(I - U_{k_i} U^T_{k_i}) A$. Moreover, the corresponding m-GKLB factorization (2.1)–(2.2) becomes

$$A P_m = Q_m B_m + U_{k_i} U^T_{k_i} A P_m\,, \tag{2.10}$$

$$A^T Q_m = P_m B^T_m + f_{\overline{k}_{i+1}} e^T_m + V_{k_i} V^T_{k_i} A^T Q_m = P_m B^T_m + f_{\overline{k}_{i+1}} e^T_m\,, \tag{2.11}$$

which suggests that it is unnecessary to orthogonalize p_{j+1} against V_{k_i}. This means that one-sided orthogonality with the vectors q_j can be used to save computational costs, thus making it an appealing feature and in general appropriate. Despite that in certain rare cases the implicit orthogonality may fail and it also may be overcome (see Remark 2.2 and Example 3.1), we continue our analysis with the one-sided deflation equations (2.10)–(2.11).

In the reminder of this section we consider *two* types of error estimates related to the quality of the computed triplets of A: the error in computing a *single* set of the next $\overline{k}_{i+1}$ largest singular triplets given k_i-PSVD, denoted by $\mathcal{A}^{err}_{itr}$, and the total error in computing all k_{i+1} desired singular triplets $\overline{k}_0, \ldots, \overline{k}_i, \overline{k}_{i+1}$ at a time, denoted by $\mathcal{A}^{err}_{tot}$. Note that for the matrix B_m from (2.10)–(2.11) and for its $\overline{k}_{i+1}$ largest singular triplets, the following relations hold:

$$B_m V^{(B_m)}_{\overline{k}_{i+1}} = U^{(B_m)}_{\overline{k}_{i+1}} \Sigma^{(B_m)}_{\overline{k}_{i+1}} \qquad \text{and} \qquad B^T_m U^{(B_m)}_{\overline{k}_{i+1}} = V^{(B_m)}_{\overline{k}_{i+1}} \Sigma^{(B_m)}_{\overline{k}_{i+1}}.$$

Moreover, these relations together with (2.3) and (2.10) imply that the $\overline{k}_{i+1}$ largest singular triplets of A satisfy

$$\begin{aligned} \mathcal{AV}_{itr}^{err} &:= \|AV_{\overline{k}_{i+1}} - U_{\overline{k}_{i+1}}\Sigma_{\overline{k}_{i+1}}\| = \|U_{k_i}U_{k_i}^T AV_{\overline{k}_{i+1}}\| \\ &= \|U_{k_i}(V_{k_i}\Sigma_{k_i} + \mathcal{U}_{k_i}^{err})^T V_{\overline{k}_{i+1}}\| \;=\; \|(\mathcal{U}_{k_i}^{err})^T V_{\overline{k}_{i+1}}\| \;=\; \|\mathcal{V}_{\overline{k}_{i+1}}^{err}\|, \end{aligned} \tag{2.12}$$

where $V_{k_i}^T V_{\overline{k}_{i+1}} = 0$ in (2.12) since the newly computed vectors $V_{\overline{k}_{i+1}}$ are orthogonal to V_{k_i}. An analogous analysis using (2.3) and (2.11) shows that

$$\mathcal{ATU}_{itr}^{err} \;:=\; \|A^T U_{\overline{k}_{i+1}} - V_{\overline{k}_{i+1}}\Sigma_{\overline{k}_{i+1}}\| = \|f_{\overline{k}_{i+1}} e_m^T U_{\overline{k}_{i+1}}^{(B_m)}\| \;=\; \|\mathcal{U}_{\overline{k}_{i+1}}^{err}\|\,, \tag{2.13}$$

which, in conjunction with (2.12), gives the error $\mathcal{A}_{itr}^{err}$ defined by

$$\mathcal{A}_{itr}^{err} \;:=\; \sqrt{(\mathcal{AV}_{itr}^{err})^2 \,+\, (\mathcal{ATU}_{itr}^{err})^2}\,. \tag{2.14}$$

On the other hand, the total errors analogous to (2.12)–(2.13) associated with incrementally computing the *complete* set of the $\sum_{j=0}^{i+1}\overline{k}_j$ largest singular triplets of A are given by

$$\mathcal{AV}_{tot}^{err} := \|AV_{k_{i+1}} - U_{k_{i+1}}\Sigma_{k_{i+1}}\| \;=\; \|[\mathcal{V}_{\overline{k}_0}^{err} \;\ldots\; \mathcal{V}_{\overline{k}_{i+1}}^{err}]\| \le \textstyle\sum_{j=0}^{i+1}\|\mathcal{V}_{\overline{k}_j}^{err}\|\,, \tag{2.15}$$

$$\mathcal{ATU}_{tot}^{err} := \|A^T U_{k_{i+1}} - V_{k_{i+1}}\Sigma_{k_{i+1}}\| = \|[\mathcal{U}_{\overline{k}_0}^{err} \;\ldots\; \mathcal{U}_{\overline{k}_{i+1}}^{err}]\| \le \textstyle\sum_{j=0}^{i+1}\|\mathcal{U}_{\overline{k}_j}^{err}\|\,. \tag{2.16}$$

Finally, (2.15) together with (2.16), result in an upper bound for the total error, $\mathcal{A}_{tot}^{err}$, given by

$$\mathcal{A}_{tot}^{err} := \sqrt{(\mathcal{AV}_{tot}^{err})^2 + (\mathcal{ATU}_{tot}^{err})^2} \tag{2.17}$$

$$\le \sqrt{\big(\textstyle\sum_{j=0}^{i+1}\|\mathcal{V}_{\overline{k}_j}^{err}\|\big)^2 + \big(\textstyle\sum_{j=0}^{i+1}\|\mathcal{U}_{\overline{k}_j}^{err}\|\big)^2} =: \mathcal{A}_{bnd}^{err}. \tag{2.18}$$

2.1. Basic implementations with `svds` and `irlba`. We close this section with a simple illustration of how the ideas outlined in this paper can be used to compute the next $\overline{k}_1$ largest singular triplets of A, given that a k_0-PSVD of A is known. Figure 2.1 displays four such approaches: the first three (`M1`–`M3`) use MATLAB's `svds` as a GKLB-based SVD solver while the last method (`M4`) leverages `irlba_def`, an updated implementation of the thick-restarted IRLBA with deflation (see Appendix A). The method `M1` employs two-sided deflation using U_0 and V_0 as in (2.1)–(2.2), while `M2` also uses two-sided deflation but only with a matrix U_0 as in (2.9). Note that line 10 in Figure 2.1 provides an alternative to `M2` without the call of an external matrix-vector product routine. `M3` only performs one-sided deflation using U_0 as in (2.10)–(2.11), while `M4` is the same as `M3` except that `svds` is replaced by `irlba_def`.

In line 4 of Figure 2.1, an initial k_0-PSVD ($k_0 = 10$) of A is computed via `svds` as a way to initialize the process. The additional $\overline{k}_1 = 5$ largest singular triplets are computed using the methods `M1` through `M4` (resp., lines 6, 9, 13, 16), which is then followed by updates of U_{k_0} and V_{k_0} resulting in a k_1-PSVD of A with $k_1 = k_0 + \overline{k}_1 = 10 + 5 = 15$ (resp., lines 7, 11, 14, 17). It is now obvious how this process can be repeated to compute the next set of largest singular triplets and so on.

There are several points worth emphasizing in Figure 2.1. If the initial k_0-PSVD of A is known, then line 4 can be omitted; otherwise, any SVD solver can be used instead of `svds` though some pre-processing might be needed (Remark 2.1). Further, the `svds` in Figure 2.1 (lines 6, 9, 13) can be replaced by another GKLB-based method such as `irlba` [7, 8]. Finally,

```
1  A = rand(400,500); [m,n] = size(A); p1 = randn(n,1); sqe=sqrt(eps);
2  LS = 'largest'; tol = 'Tolerance'; rsv = 'RightStartVector';
3  % Intitial k_0 - PSVD
4  [U0,S0,V0] = svds(A, 10, LS, tol, sqe, rsv, p1); p1=p1-V0*(V0'*p1);
5  % Method 1 (M1): two-sided (I-UU')*A and (I-VV')*A'
6  [U1,S1,V1] = svds(@(x,t)Asvdp(x,t,A,U0,V0,1), [m n], 5, LS, tol, sqe, rsv, p1);
7  U = [U0 U1]; V = [V0 V1]; S = blkdiag(S0,S1);
8  % Method 2 (M2): two-sided (I-UU')*A and A'(I-UU')
9  [U1,S1,V1] = svds(@(x,t)Asvdp(x,t,A,U0,V0,2), [m n], 5, LS, tol, sqe, rsv, p1);
10 % Alternately:[U1,S1,V1] = svds(A-U*(U'*A), 5, LS, tol, sqe, rsv, p1);
11 U = [U0 U1]; V = [V0 V1]; S = blkdiag(S0,S1);
12 % Method 3 (M3): one-sided (I-UU')*A
13 [U1,S1,V1] = svds(@(x,t)Asvdp(x,t,A,U0,V0,3), [m n], 5, LS, tol, sqe, rsv, p1);
14 U = [U0 U1]; V = [V0 V1]; S = blkdiag(S0,S1);
15 % Method 4 (M4) irlba_def one-sided (I-UU')*A
16 [U1,S1,V1,~,~,~,~] = irlba_def(A,p1,U0,V0,5);
17 U = [U0 U1]; V = [V0 V1]; S = blkdiag(S0,S1);
18 % External Function Call
19 function y = Asvdp(x,t,A,U,V,method)
20 if strcmp(t,'notransp')
21     y = A*x; y = y - U*(U'*y);
22 else
23  if method == 2, x = x - U*(U'*x); end
24  y = A'*x;
25  if method == 1, y = y - V*(V'*y); end
26 end
27 end
```

FIG. 2.1. *Multiple approaches to computing the next $\overline{k}_1$ largest singular triplets given k_0-PSVD of A by using* `svds` *and* `irlba_def` *(Appendix A).*

lines 6, 9, and 13 also highlight a desirable and intrinsic feature of our proposed scheme—using a GKLB-based routine for the computation of the next set of the largest singular triplets requires *no modification* of the routine itself, and the explicit deflation can be handled by an external function call (lines 19–26). This is true in any programming environment and thus has a broad and immediate impact given a wide range of implementations of GKLB-based methods across disciplines; see, e.g., [32, p. 3] and the references therein.

REMARK 2.2. As an alternative to using `svds`, our method `M4` in Figure 2.1 utilizes a short and easily understandable MATLAB code `irlba_def` (see Appendix A) that implements one-sided explicit deflation in a straightforward manner. `irlba_def` in Appendix A can be viewed as a generalization of a simplistic implementation of `irlba` from [2], though a more robust implementation of `irlba` *with explicit deflation* analogous to the implementation[3] in [8] is currently under development. One advantage of `irlba_def` over `svds` is that it can easily access and rather cheaply compute the values $U_{k_i}^T AP_m$, which has a substantial payoff (lines 18, 44 in Appendix A). For example, the access to $U_{k_i}^T AP_m$ enables `irlba_def` to easily determine the error $\mathcal{A}_{itr}^{err}$ (2.14) and the error bound $\mathcal{A}_{bnd}^{err}$ (2.18) (lines 62, 64 in Appendix A), whereas the same error would have to be computed exactly in case of `svds`. Furthermore, in the restarting strategy, post-multiplying $U_{k_i}^T AP_m$ by $[v_1^{(B_m)}, \ldots, v_{j-1}^{(B_m)}]$ allows for a continued tracking of the error (see line 80 in Appendix A); for a through discussion on how restarting is set up, see [2, 7].

Another advantage of using `irlba_def` over `svds` in Figure 2.1 (`M3`–`M4`) becomes evident in case that a breakdown occurs in the d-GKLB Algorithm 1 (i.e., $\alpha_j \approx 0$ or $\beta_j \approx 0$). First note that this is a rare occurrence though it can happen in practice, for example, when there are numerically repeating singular values (see Example 3.1). Both `svds` and `irlba_def` handle this problem similarly, i.e., in order to continue to build a basis, a random vector is introduced, either α_j or β_j is set to zero, and then the GKLB procedure is continued. But, in order to secure one-sided explicit deflation, `irlba_def` takes an additional step and first

[3]Code available at http://www.netlib.org/numeralgo/na26.tgz.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

orthogonalizes the random vector against the converged singular vectors before continuing the GKLB procedure (lines 23, 36, 49, 87–90 in Appendix A). This in turn prevents a breakdown encountered by `svds` in `M3` when the computed set of the $\overline{k}_j$ singular vectors is not orthogonal to the previous k_{j-1} singular vectors (this is denoted by "–" in Table 3.2 in Example 3.1). For the sake of completeness, it is worth noting here that the methods `M1` and `M2` in Figure 2.1 do not suffer the same fate when using `svds` due to the fact that the two-sided explicit deflation is not impacted since the d-GKLB Algorithm 1 is applied explicitly to the deflated matrix where the deflated singular values are moved to zero. Finally, we note that when using `irlba_def` and a random vector is introduced (lines 23, 36, 49 in Appendix A), our recommendation is to compute $\mathcal{A}_{itr}^{err}$ exactly just as in case with `svds`. In our simple implementation we just used $\sqrt{\text{eps}} \approx 10^{-8}$ as an approximate zero to determine when a random vector is to be introduced (lines 22, 35, 48 in Appendix A). This introduces a new error that is not included by our previous error analysis, and therefore we simply compute $\mathcal{A}_{itr}^{err}$ exactly (see lines 70–72 in Appendix A). We note here that there are other, possibly more appropriate, alternatives that include, for example, the norm and/or the size of the matrix; see [7, 8, 9, 19, 20, 28] and the references within for more details—this investigation goes beyond the scope of the paper and contrary to our intended objective of keeping implementation simplistic.

3. Numerical examples. All computations were carried out using MATLAB version R2022b on an iMac with a 3.7Ghz Intel Core i5 processor and 32GB (2667 MHz) of memory using the operating system macOS Monterey. Machine epsilon is $\epsilon = 2.2 \cdot 10^{-16}$. The numerical experiments were performed on matrices from the SuiteSparse Matrix Collection [12] with varied sizes, norms, and condition numbers (see Table 3.1). The matrices mhd4800b, bibd_20_10, and stormG2_1000 were used in [22], while mhd4800b is also used in the demo routine for `svt`.

TABLE 3.1
Test matrices used for the examples from the SuiteSparse Matrix Collection [12].

Matrix/Properties	size	nonzeros	σ_1 (Largest)	Cond #
illc1033	1,033×320	4,732	≈ 2.144	$1.9 \cdot 10^4$
mhd4800b	4,800×4,800	27,520	≈ 2.196	$8.1 \cdot 10^{13}$
JP	87,616×67,320	13,734,559	$\approx 4{,}223$	not full rank
bibd_20_10	190×184,756	8,314,020	$\approx 1{,}403.2$	$1.2 \cdot 10^1$
stormG2_1000	528,185×1,377,306	3,459,881	$\approx 3{,}288$	not full rank

For our example, we compared between MATLAB's `svds`, `irlba_def` (Appendix A), and the MATLAB software wrapper `svt` [22]. The methods `M1` through `M4` are as labeled in Figure 2.1. Associated with the computation of a k_{i+1}-PSVD of A, we report the error $\mathcal{A}_{tot}^{err}$ from (2.17) computed exactly for all methods and the error bound $\mathcal{A}_{bnd}^{err}$ from (2.18) only for `M4` with the starting error of a $\overline{k}_0$-PSVD being computed exactly. For all methods we also report

$$\mathcal{UV}^{err} := \sqrt{\|U_{k_{i+1}}^T U_{k_{i+1}} - I\|^2 + \|V_{k_{i+1}}^T V_{k_{i+1}} - I\|^2}, \tag{3.1}$$

and for Example 3.2 we record the CPU times in seconds using MATLAB's `tic-toc` command. We used the same random vector for all restart runs with a fixed RandomStream chosen to coincide with the demo for `svt` (`svt` did not permit an input starting vector). We set the tolerance for convergence to $\sqrt{\text{eps}} \approx 10^{-8}$.

EXAMPLE 3.1. This example illustrates the error growth for methods `M1` through `M4` for the matrices mhd4800b, JP, and bibd_20_10. All methods/runs start with the same k_0-PSVD

ETNA
Kent State University and
Johann Radon Institute (RICAM)

($k_0 = 10$) and compute subsequent sets of triplets in increments of $\overline{k}_1 = 5$, $\overline{k}_1 = 10$, and $\overline{k}_1 = 20$ until reaching $k_{i+1} = 110$. Table 3.2 reports the errors $\mathcal{A}_{tot}^{err}$ and $\mathcal{UV}^{err}$ after one restart and when $k_{i+1} = 110$. The error growth is modest for well-conditioned matrices and both the two-sided (`M1`, `M2`) and one-sided (`M3`, `M4`) methods behave as expected. Table 3.2 shows that the error bound $\mathcal{A}_{bnd}^{err}$ (2.18) from `irlba_def` is quite good and does not tend to dramatically overestimate the exact error $\mathcal{A}_{tot}^{err}$. It is worth noting that if $\mathcal{UV}^{err}$ becomes too large, then it can be refined through a process similar to Remark 2.1, e.g., for the matrix mhd4800b with $\overline{k}_1 = 5$ and $k_{20} = 110$, a single refinement at the end of method `M3` results in the reduced error $\mathcal{UV}^{err} = 5.5 \cdot 10^{-15}$ with a new $\mathcal{A}_{tot}^{err} = 2.7 \cdot 10^{-10}$. This can be done at any stage, though its frequent use is not recommended as it can be computationally expensive.

TABLE 3.2
Example 3.1: Displays of the errors $\mathcal{A}_{tot}^{err}$ (2.17) and $\mathcal{UV}^{err}$ (3.1) for the four methods from Figure 2.1. All runs start with $k_0 = 10$, and reported are the errors after one restart k_1 and when $k_{i+1} = 110$. For the matrix bibd_20_10, a random vector is introduced and a failure to converge for the one-sided method `M3` *is denoted by (−).*

		mhd4800b			JP			bibd_20_10		
k_0-PSVD		$\mathcal{A}_{tot}^{err} = 1.2\cdot10^{-9}$ $\mathcal{UV}^{err} = 3.3\cdot10^{-15}$			$\mathcal{A}_{tot}^{err} = 4.7\cdot10^{-9}$ $\mathcal{UV}^{err} = 4.9\cdot10^{-15}$			$\mathcal{A}_{tot}^{err} = 2.1\cdot10^{-10}$ $\mathcal{UV}^{err} = 1.7\cdot10^{-14}$		
Initial $\overline{k}_1$		$\overline{k}_1 = 5$	$\overline{k}_1 = 10$	$\overline{k}_1 = 20$	$\overline{k}_1 = 5$	$\overline{k}_1 = 10$	$\overline{k}_1 = 20$	$\overline{k}_1 = 5$	$\overline{k}_1 = 10$	$\overline{k}_1 = 20$
1 Restart		$k_1 = 15$	$k_1 = 20$	$k_1 = 30$	$k_1 = 15$	$k_1 = 20$	$k_1 = 30$	$k_1 = 15$	$k_1 = 20$	$k_1 = 30$
`M1`	$\mathcal{A}_{tot}^{err}$	$1.4\cdot10^{-10}$	$1.4\cdot10^{-10}$	$1.5\cdot10^{-10}$	$2.0\cdot10^{-5}$	$2.0\cdot10^{-6}$	$2.2\cdot10^{-6}$	$2.1\cdot10^{-10}$	$2.2\cdot10^{-10}$	$2.2\cdot10^{-10}$
	$\mathcal{UV}^{err}$	$3.6\cdot10^{-15}$	$3.9\cdot10^{-15}$	$3.9\cdot10^{-15}$	$6.5\cdot10^{-15}$	$5.3\cdot10^{-15}$	$4.9\cdot10^{-15}$	$2.4\cdot10^{-14}$	$5.2\cdot10^{-14}$	$5.4\cdot10^{-14}$
`M2`	$\mathcal{A}_{tot}^{err}$	$1.4\cdot10^{-10}$	$1.4\cdot10^{-10}$	$1.5\cdot10^{-10}$	$2.0\cdot10^{-5}$	$2.0\cdot10^{-6}$	$2.2\cdot10^{-6}$	$2.1\cdot10^{-10}$	$2.2\cdot10^{-10}$	$2.2\cdot10^{-10}$
	$\mathcal{UV}^{err}$	$8.9\cdot10^{-15}$	$7.0\cdot10^{-15}$	$7.2\cdot10^{-15}$	$7.8\cdot10^{-15}$	$5.6\cdot10^{-15}$	$5.0\cdot10^{-15}$	$3.0\cdot10^{-14}$	$5.9\cdot10^{-14}$	$6.3\cdot10^{-14}$
`M3`	$\mathcal{A}_{tot}^{err}$	$1.4\cdot10^{-10}$	$1.4\cdot10^{-10}$	$1.5\cdot10^{-10}$	$2.0\cdot10^{-5}$	$2.0\cdot10^{-6}$	$2.2\cdot10^{-6}$	−	−	−
	$\mathcal{UV}^{err}$	$7.1\cdot10^{-15}$	$7.5\cdot10^{-15}$	$7.8\cdot10^{-15}$	$5.8\cdot10^{-15}$	$5.7\cdot10^{-15}$	$5.1\cdot10^{-15}$	−	−	−
`M4`	$\mathcal{A}_{tot}^{err}$	$1.6\cdot10^{-10}$	$1.4\cdot10^{-10}$	$1.5\cdot10^{-10}$	$1.8\cdot10^{-6}$	$6.5\cdot10^{-8}$	$9.2\cdot10^{-7}$	$2.1\cdot10^{-10}$	$2.1\cdot10^{-10}$	$2.1\cdot10^{-10}$
	$\mathcal{UV}^{err}$	$6.6\cdot10^{-15}$	$7.2\cdot10^{-15}$	$7.0\cdot10^{-15}$	$7.2\cdot10^{-15}$	$7.3\cdot10^{-15}$	$5.3\cdot10^{-15}$	$2.1\cdot10^{-14}$	$1.2\cdot10^{-14}$	$2.2\cdot10^{-14}$
	$\mathcal{A}_{bnd}^{err}$	$3.1\cdot10^{-10}$	$1.4\cdot10^{-10}$	$1.5\cdot10^{-10}$	$1.8\cdot10^{-6}$	$7.0\cdot10^{-8}$	$9.3\cdot10^{-7}$	$2.2\cdot10^{-10}$	$2.3\cdot10^{-10}$	$2.3\cdot10^{-10}$
n Restart		$k_{20} = 110$	$k_{10} = 110$	$k_5 = 110$	$k_{20} = 110$	$k_{10} = 110$	$k_5 = 110$	$k_{20} = 110$	$k_{10} = 110$	$k_5 = 110$
`M1`	$\mathcal{A}_{tot}^{err}$	$5.5\cdot10^{-10}$	$1.0\cdot10^{-9}$	$2.0\cdot10^{-10}$	$4.1\cdot10^{-5}$	$2.6\cdot10^{-5}$	$2.2\cdot10^{-6}$	$5.4\cdot10^{-9}$	$6.3\cdot10^{-9}$	$3.3\cdot10^{-9}$
	$\mathcal{UV}^{err}$	$2.2\cdot10^{-10}$	$7.2\cdot10^{-15}$	$5.7\cdot10^{-15}$	$1.7\cdot10^{-14}$	$9.0\cdot10^{-15}$	$7.5\cdot10^{-15}$	$4.6\cdot10^{-11}$	$4.9\cdot10^{-11}$	$2.7\cdot10^{-11}$
`M2`	$\mathcal{A}_{tot}^{err}$	$6.6\cdot10^{-10}$	$1.3\cdot10^{-9}$	$2.0\cdot10^{-10}$	$4.1\cdot10^{-5}$	$2.6\cdot10^{-5}$	$2.8\cdot10^{-6}$	$1.9\cdot10^{-7}$	$1.4\cdot10^{-8}$	$2.9\cdot10^{-9}$
	$\mathcal{UV}^{err}$	$2.2\cdot10^{-9}$	$4.3\cdot10^{-13}$	$1.2\cdot10^{-14}$	$2.7\cdot10^{-14}$	$9.3\cdot10^{-15}$	$8.2\cdot10^{-15}$	$9.0\cdot10^{-10}$	$3.2\cdot10^{-11}$	$1.1\cdot10^{-11}$
`M3`	$\mathcal{A}_{tot}^{err}$	$3.8\cdot10^{-10}$	$4.0\cdot10^{-10}$	$2.0\cdot10^{-10}$	$4.1\cdot10^{-5}$	$2.6\cdot10^{-5}$	$2.8\cdot10^{-6}$	−	−	−
	$\mathcal{UV}^{err}$	$1.1\cdot10^{-9}$	$1.4\cdot10^{-14}$	$1.2\cdot10^{-14}$	$2.4\cdot10^{-14}$	$1.1\cdot10^{-14}$	$7.9\cdot10^{-15}$	−	−	−
`M4`	$\mathcal{A}_{tot}^{err}$	$7.6\cdot10^{-10}$	$2.0\cdot10^{-10}$	$2.0\cdot10^{-10}$	$3.8\cdot10^{-5}$	$1.5\cdot10^{-5}$	$1.1\cdot10^{-6}$	$2.2\cdot10^{-10}$	$2.2\cdot10^{-10}$	$2.2\cdot10^{-10}$
	$\mathcal{UV}^{err}$	$1.8\cdot10^{-14}$	$1.5\cdot10^{-14}$	$1.1\cdot10^{-14}$	$1.8\cdot10^{-14}$	$9.5\cdot10^{-15}$	$1.3\cdot10^{-14}$	$4.5\cdot10^{-14}$	$3.9\cdot10^{-14}$	$1.1\cdot10^{-13}$
	$\mathcal{A}_{bnd}^{err}$	$4.1\cdot10^{-9}$	$5.9\cdot10^{-10}$	$3.4\cdot10^{-10}$	$3.2\cdot10^{-4}$	$2.7\cdot10^{-5}$	$1.5\cdot10^{-6}$	$7.2\cdot10^{-10}$	$3.6\cdot10^{-10}$	$3.1\cdot10^{-10}$

Table 3.2 also shows that for the matrix bibd_20_10, the one-sided `M3` method fails to converge—it appears that the singular value distribution of bibd_20_10 (one largest singular value 1403.2, the next 19 largest singular values clustered around 467.7, and the final 170 singular values clustered around 113.4) caused the `svds` to introduce random vector(s) in order to continue to build the subspace. This is not detected by the FLAG output for `svds` but can be determined by analyzing $\mathcal{UV}^{err}$, as the orthogonality with the converged singular vectors is lost; the one-sided method `M4` remedies this by using `irlba_def` (lines 36, 49 in Appendix A). Thus, we recommend $\mathcal{UV}^{err}$ to be checked if one-sided deflation is used with `svds`.

EXAMPLE 3.2. In this example, we show how the problem of computing the largest singular triplets above a predetermined threshold (for applications, see for example [4, 5, 11]) can be solved using our proposed methods `M1`–`M4`, and we compare our results with an external method `svt` [22]. For `M1`–`M4` we implemented a simplistic algorithm based on Figure 2.1, while the `svt` implementation came from the authors' GitHub account.[4] Both `svt` and `M1`–`M4` approach the problem of thresholding as sequentially computing a predetermined number, denoted here by $'incre'$, of the largest singular triplets until a singular value that falls below the threshold is identified. For a fair comparison, all methods used the default settings for deflation from `svt`, i.e., an initial k_0-PSVD with $k_0 = 6$ was computed, and then at each restart the number of the newly computed singular triplets were doubled ($'incre' = 2 \cdot 'incre'$, where initially, $'incre'$ is set to 5) until a desired threshold was reached.

From Table 3.3, we see that the error $\mathcal{A}_{tot}^{err}$ (respectively, $\mathcal{UV}^{err}$) corresponding to *all* four methods `M1`–`M4` is no worse (respectively, several orders of magnitude lower) than the error produced by `svt`. Moreover, the CPU time for `M1`–`M4` is roughly half of the time used by `svt`. This suggests that our proposed methods, despite our simplistic implementations, are at least very competitive when it comes to thresholding. We do note, however, that the comparison between `M1`–`M4` and `svt` is not as straightforward, and in fact leads to many open questions, e.g., different choices of $'incre'$ (or different threshold) can cause the algorithm to succeed or fail thus placing similar limitations on our methods as `svt`. More concretely, `svt` and `M1` both failed to converge for illc1033 with a threshold set to 0.2, i.e., neither methods captured all 222 singular values, where `svt` and `M1` only computed 79 and 21 out of 222, respectively. However, changing $'incre'$ to 6, enabled `M1` to compute all 222 singular triplets above 0.2, but `svt` still failed. Other reasonable choices for $'incre'$ also led to `svt` not being able to compute all 222 singular triplets for illc1033. The threshold 0.1 chosen for mhd4800b was used in the demo for `svt`, and the threshold for stormG2_100 was set to capture the 50 largest singular triplets as reported in [22, Sec. 4.6] for `svt`. While it is beyond the scope of this paper and an ongoing area of research, our preliminary results show that convergence of the methods based on `svds` can be improved by utilizing more sophisticated techniques, e.g., using $'FailureTreatment','keep'$, where the matrices V and U are updated with only the converged singular vectors and then `svds` is restarted with an appropriate linear combination of the approximate unconverged vectors [29].

TABLE 3.3
Example 3.2: Displays of the error $\mathcal{A}_{tot}^{err}$ (2.17), the error bound $\mathcal{A}_{bnd}^{err}$ (2.18) for `M4`*, $\mathcal{UV}^{err}$ (3.1), and the CPU times in seconds for the threshold routine based on the four methods from Figure 2.1 and* `svt` *[22]. Failure to capture* all *singular values above the given threshold is denoted by (−).*

	mhd4800b			stormG2_100			illc1033		
Threshold	0.1 ($k_1 = 48$)			632.4603 ($k_1 = 50$)			0.2 ($k_1 = 222$)		
	$\mathcal{A}_{tot}^{err}$ / $\mathcal{A}_{bnd}^{err}$	$\mathcal{UV}^{err}$	CPU time	$\mathcal{A}_{tot}^{err}$ / $\mathcal{A}_{bnd}^{err}$	$\mathcal{UV}^{err}$	CPU time	$\mathcal{A}_{tot}^{err}$ / $\mathcal{A}_{bnd}^{err}$	$\mathcal{UV}^{err}$	CPU time
`M1`	$9.9\cdot10^{-10}$	$4.1\cdot10^{-12}$	0.30 s	$3.0\cdot10^{-5}$	$1.7\cdot10^{-14}$	$5.4\cdot10^{1}$ s	–	–	–
`M2`	$9.0\cdot10^{-10}$	$3.8\cdot10^{-12}$	0.20 s	$2.4\cdot10^{-5}$	$8.7\cdot10^{-9}$	$4.6\cdot10^{1}$ s	$4.2\cdot10^{-8}$	$1.3\cdot10^{-12}$	2.7 s
`M3`	$4.8\cdot10^{-11}$	$1.7\cdot10^{-13}$	0.18 s	$2.4\cdot10^{-5}$	$8.7\cdot10^{-9}$	$4.2\cdot10^{1}$ s	$4.2\cdot10^{-8}$	$1.3\cdot10^{-12}$	2.0 s
`M4`	$8.2\cdot10^{-12}$ / $1.1\cdot10^{-11}$	$2.2\cdot10^{-14}$	0.32 s	$6.8\cdot10^{-6}$ / $1.0\cdot10^{-5}$	$1.8\cdot10^{-14}$	$1.1\cdot10^{2}$ s	$2.8\cdot10^{-8}$ / $5.3\cdot10^{-8}$	$2.4\cdot10^{-13}$	0.81 s
`svt`	$1.5\cdot10^{-9}$	$2.7\cdot10^{-9}$	0.52 s	$3.0\cdot10^{-5}$	$1.2\cdot10^{-8}$	$2.9\cdot10^{2}$ s	–	–	–

4. Conclusion.

In this short note we described and briefly analyzed a powerful method for enlarging an already computed PSVD. The simplicity of our proposed approach, the ease in which it can be implemented, the reasonable error growth, and its direct connection to commonly used SVD solvers makes this work particularly attractive to a broader audience.

[4] Code available at: https://github.com/Hua-Zhou/svt. Retrived on November 30, 2022.

As part of the ongoing work, the authors are currently developing a public domain threshold software and an updated explicit deflation irlba public domain code.

Appendix A. Simplistic implementation of thick-restarted IRLBA with deflation.

```
function[U,S,V,flag,AVerr_itr,ATUerr_itr,Aerr_itr] = irlba_def(A,P,U,V,k)
%IRLBA_DEF - thick-restarted w/ Ritz vectors and explicit one-sided deflation
%   References: Primarily [2] with more robust theory/implementation in [7,8]
%   This MATLAB code is for illustrated purposes only and is not optimized.
% Initialization
m = max(3*k,15); Smax = 0; J = 1; iter = 1; UTAP = 0; tol = sqrt(eps);
ranvec = 0; flag = 0; if ~isempty(U), UTAP = zeros(size(U,2),m); end
% d-GKLB Alg. 1 step 3 Comment out if input V'*P(:,J) = 0
if ~isempty(V), P(:,J) = P(:,J) - V*(V'*P(:,J)); end
P(:,J) = P(:,J)/norm(P(:,J));

% d-GKLB and thick-restarted (maximum iteration fixed at 1000 steps)
while iter <= 1000

  % d-GKLB Alg. 1 steps 4 and 5
  Q(:,J) = A*P(:,J);
  if ~isempty(U)  % deflation step - (2.12) and Remark 2.2
      UTAP(:,J) = U'*Q(:,J); Q(:,J) = Q(:,J) - U*UTAP(:,J);
  end
  if J > 1, Q(:,J) = Q(:,J) - Q(:,1:J-1)*B(1:J-1,J);  end % Thick-Restart (1)
  Qnorm = norm(Q(:,J));
  if  Qnorm < sqrt(eps) % if needed rand vec, deflation, orthogonalize
      Q(:,J) = genrand(Q(:,1:J-1),U); B(J,J) = 0; ranvec = 1; % Remark 2.2
  else
      B(J,J) = norm(Q(:,J)); Q(:,J) = Q(:,J)/B(J,J);
  end

  % d-GKLB Alg. 1 step 6
  for i = J:m
    f = A'*Q(:,i) - B(i,i)*P(:,i);  % d-GKLB Alg. 1 step 7
    f = f - P(:,1:i)*(P(:,1:i)'*f); % one-side reorth step
    if i < m % d-GKLB Alg. 1 step 8
       % d-GKLB Alg. 1 steps 9 and 12
       fnorm = norm(f);
       if fnorm < sqrt(eps) % if needed rand vec, deflation, orthogonalize
          P(:,i+1) = genrand(P(:,1:i),V); B(i,i+1) = 0; ranvec = 1; % Remark 2.2
       else
          B(i,i+1) = fnorm; P(:,i+1) = f/B(i,i+1);
       end

       % d-GKLB Alg. 1 steps 10, 11, and 13
       Q(:,i+1) = A*P(:,i+1);
       if ~isempty(U) % deflation step - (2.12) and Remark 2.2
          UTAP(:,i+1) = U'*Q(:,i+1); Q(:,i+1) = Q(:,i+1) - U*UTAP(:,i+1);
       end
       Q(:,i+1) = Q(:,i+1) - B(i,i+1)*Q(:,i);
       Qnorm = norm(Q(:,i+1));
       if  Qnorm < sqrt(eps) % if needed rand vec, deflation, orthogonalize
           Q(:,i+1) = genrand(Q(:,1:i),U); B(i+1,i+1) = 0; ranvec = 1; % Remark 2.2
       else
           B(i+1,i+1) = Qnorm; Q(:,i+1) = Q(:,i+1)/B(i+1,i+1);
       end
    end
  end % end for
```

```
  % Thick-Restarted (2)
  [ub,sb,vb] = svd(B); Smax = max(Smax,sb(1,1)); fnorm = norm(f);
  J = k+4; P(:,1:J-1) = P*vb(:,1:J-1); Q(:,1:J-1) = Q*ub(:,1:J-1);
  sb = sb(1:J-1,1:J-1);  rho = fnorm*ub(m,1:J-1);

  % Error Computations
  AVerr_itr  = norm(UTAP*vb(:,1:k)); % (2.14)
  ATUerr_itr = norm(rho(1:k));       % (2.15)
  Aerr_itr   = sqrt(AVerr_itr^2+ATUerr_itr^2); %(2.16)

  % test convergence and exit
  if Aerr_itr < Smax*tol || iter == 1000
     V =  P(:,1:k); U = Q(:,1:k); S = sb(1:k,1:k);
     if ranvec % Remark 2.2 - random vector compute exact residuals
       AVerr_itr  = norm(A*V-U*S);      % (2.14)
       ATUerr_itr = norm(A'*U-V*S);     % (2.15)
       Aerr_itr   = sqrt(AVerr_itr^2+ATUerr_itr^2); %(2.16)
     end
     if iter == 1000, flag = 1; end; return;
  end

  % Thick-Restarted (3)
  B = [sb, rho']; P(:,J) = f/fnorm; iter = iter + 1;
  if ~isempty(U) % update deflation based on thick-restarted strategy
      UTAP(:,1:J-1) = UTAP*vb(:,1:J-1); % (2.12) and Remark 2.2
  end

end % end while

% Simple function to generate random vector and orthogonalize against
% basis vectors and converged singular vectors.
function y = genrand(X,Z) % Remark 2.2
  y = randn(size(X,1),1); if ~isempty(Z), y = y - Z*(Z'*y); end
  y = y - X*(X'*y); y = y/norm(y);
end % end genrand

end % end irlba_def
```

REFERENCES

[1] O. ALTER, P. O. BROWN, AND D. BOTSTEIN, *Singular value decomposition for genome-wide expression data processing and modeling*, Proc. Nat. Acad. Sci., 97 (2000), pp. 10101–10106.

[2] J. BAGLAMA, *IRLBA: fast partial singular value decomposition method*, in Handbook of Big Data, P. Bühlmann, P. Drineas, M. Kane, and M. van der Laan, eds., Chapman & Hall/CRC Handb. Mod. Stat. Methods, CRC Press, Boca Raton, 2016, pp. 125–136.

[3] J. BAGLAMA, D. CALVETTI, AND L. REICHEL, *Iterative methods for the computation of a few eigenvalues of a large symmetric matrix*, BIT Numer. Math., 36 (1996), pp. 400–421.

[4] J. BAGLAMA, C. FENU, L. REICHEL, AND G. RODRIGUEZ, *Analysis of directed networks via partial singular value decomposition and Gauss quadrature*, Linear Algebra Appl., 456 (2014), pp. 93–121.

[5] J. BAGLAMA, M. KANE, B. LEWIS, AND A. POLIAKOV, *Efficient thresholded correlation using truncated singular value decomposition*, Preprint on arXiv, 2015. https://arxiv.org/abs/1512.07246

[6] J. BAGLAMA, V. PEROVIĆ, AND J. PICUCCI, *Hybrid iterative refined restarted Lanczos bidiagonalization methods*, Numer. Algorithms, 93 (2023), pp. 1183–1212.

[7] J. BAGLAMA AND L. REICHEL, *Augmented implicitly restarted Lanczos bidiagonalization methods*, SIAM J. Sci. Comput., 27 (2005), pp. 19–42.

[8] ———, *Restarted block Lanczos bidiagonalization methods*, Numer. Algorithms, 43 (2006), pp. 251–272.

[9] Z. BAI, J. DEMMEL, J. DONGARRA, A. RUHE, AND H. VAN DER VORST, *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*, SIAM, Philadelphia, 2000.

[10] J. L. BARLOW, *Reorthogonalization for the Golub-Kahan-Lanczos bidiagonal reduction*, Numer. Math., 124 (2013), pp. 237–278.

[11] J.-F. CAI, E. J. CANDÈS, AND Z. SHEN, *A singular value thresholding algorithm for matrix completion*, SIAM J. Optim., 20 (2010), pp. 1956–1982.
[12] T. A. DAVIS AND Y. HU, *The University of Florida sparse matrix collection*, ACM Trans. Math. Software, 38 (2011), Art. 1, 25 pages.
[13] L. ELDÉN, *Matrix Methods in Data Mining and Pattern Recognition*, SIAM, Philadelphia, 2007.
[14] G. GOLUB AND W. KAHAN, *Calculating the singular values and pseudo-inverse of a matrix*, J. Soc. Indust. Appl. Math. Ser. B Numer. Anal., 2 (1965), pp. 205–224.
[15] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, 4th ed., Johns Hopkins University Press, Baltimore, 2013.
[16] N. HALKO, P. G. MARTINSSON, AND J. A. TROPP, *Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions*, SIAM Rev., 53 (2011), pp. 217–288.
[17] Z. JIA AND D. NIU, *A refined harmonic Lanczos bidiagonalization method and an implicitly restarted algorithm for computing the smallest singular triplets of large matrices*, SIAM J. Sci. Comput., 32 (2010), pp. 714–744.
[18] I. T. JOLLIFFE, *Principal Component Analysis*, 2nd. ed. Springer, New York, 2002.
[19] E. KOKIOPOULOU, C. BEKAS, AND E. GALLOPOULOS, *Computing smallest singular triplets with implicitly restarted Lanczos bidiagonalization*, Appl. Numer. Math., 49 (2004), pp. 39–61.
[20] R. M. LARSEN, *Lanczos bidiagonalization with partial reorthogonalization*, Tech. Peport, DAIMI Report Series 27, #537, Aarhus University, Aarhus, 1998.
[21] B. W. LEWIS, J. BAGLAMA, AND L. REICHEL, *The* `irlba` *package*, Software package, 2021. https://cran.r-project.org/web/packages/irlba
[22] C. LI AND H. ZHOU, *svt: Singular value thresholding in MATLAB*, J. Statist. Softw., 81 (2017), pp. 1–12.
[23] R. LI, Y. XI, E. VECHARYNSKI, C. YANG, AND Y. SAAD, *A thick-restart Lanczos algorithm with polynomial filtering for Hermitian eigenvalue problems*, SIAM J. Sci. Comput., 38 (2016), pp. A2512–A2534.
[24] Q. LIANG AND Q. YE, *Computing singular values of large matrices with an inverse-free preconditioned Krylov subspace method*, Electron. Trans. Numer. Anal., 42 (2014), pp. 197–221. http://etna.ricam.oeaw.ac.at/vol.42.2014/pp197-221.dir/pp197-221.pdf
[25] P.-G. MARTINSSON, V. ROKHLIN, AND M. TYGERT, *A randomized algorithm for the decomposition of matrices*, Appl. Comput. Harmon. Anal., 30 (2011), pp. 47–68.
[26] Y. NAKATSUKASA, *Accuracy of singular vectors obtained by projection-based SVD methods*, BIT Numer. Math., 57 (2017), pp. 1137–1152.
[27] E. ONUNWOR AND L. REICHEL, *On the computation of a truncated SVD of a large linear discrete ill-posed problem*, Numer. Algorithms, 75 (2017), pp. 359–380.
[28] B. N. PARLETT, *The Symmetric Eigenvalue Problem*, SIAM, Philadelphia, 1998.
[29] Y. SAAD, *Variations on Arnoldi's method for computing eigenelements of large unsymmetric matrices*, Linear Algebra Appl., 34 (1980), pp. 269–295.
[30] A. STATHOPOULOS, *Locking issues for finding a large number of eigenvectors of Hermitian matrices*, Tech. Rep. WM-CS-2005-09, College of William and Mary, Williamsburg, 2005.
[31] THE MATHWORKS, *MATLAB (R2022b)* `svds`, Natick, Massachusetts.
[32] K. TSUYUZAKI, H. SATO, K. SATO, AND I. NIKAIDO, *Benchmarking principal component analysis for large-scale single-cell RNA-sequencing*, Genome Biology, 21 (2020), 17 pages. DOI: https://doi.org/10.1186/s13059-019-1900-3
[33] L. WU, E. ROMERO, AND A. STATHOPOULOS, *PRIMME_SVDS: a high-performance preconditioned svd solver for accurate large-scale computations*, SIAM J. Sci. Comput., 39 (2017), pp. S248–S271.