SIAM J. Sci. Comput.
Vol. 41, No. 4, pp. A2355–A2383

PASS-EFFICIENT RANDOMIZED ALGORITHMS FOR LOW-RANK MATRIX APPROXIMATION USING ANY NUMBER OF VIEWS[*]

ELVAR K. BJARKASON[†]

Abstract. This paper describes practical randomized algorithms for low-rank matrix approximation that accommodate any budget for the number of views of the matrix. The presented algorithms, which are intended to be as pass efficient as needed, expand and improve on popular randomized algorithms targeting efficient low-rank reconstructions. First, a more flexible subspace iteration algorithm is presented that works for any views $v \geq 2$, instead of only allowing an even v. Second, we propose more general and more accurate single-pass algorithms. In particular, we propose a more accurate memory efficient single-pass method and a more general single-pass algorithm which, unlike previous methods, does not require prior information to ensure near peak performance. Third, combining ideas from subspace and single-pass algorithms, we present a more pass-efficient randomized block Krylov algorithm, which can achieve a desired accuracy using considerably fewer views than that needed by a subspace or previously studied block Krylov methods. However, the proposed accuracy enhanced block Krylov method is restricted to large matrices that are accessed either a few columns or a few rows at a time. Recommendations are also given on how to apply the subspace and block Krylov algorithms either when estimating the dominant left or right singular subspace of a matrix or when estimating a normal matrix, such as those appearing in inverse problems. Computational experiments are carried out that demonstrate the applicability and effectiveness of the presented algorithms.

Key words. singular value decomposition, random, low rank, subspace iteration, single pass, block Krylov

AMS subject classifications. 65F30, 68W20, 15A18, 35R30, 86A22

DOI. 10.1137/18M118966X

1. Introduction.

Low-rank matrix approximation methods are important tools for improving computational performance when dealing with large data sets and highly parameterized models. Recent years have seen a surge in randomized methods for low-rank matrix approximation which are designed for high-performance computing and have provable accuracy bounds [20, 24, 25, 23, 27, 28, 37, 46, 54, 55]. The main computational tasks of modern randomized matrix algorithms involve multiplying the data matrix under consideration with a small number of tall, thin matrices. Since accessing a large matrix is expensive, limiting the number of times it is multiplied with another matrix is key to efficiency. With this in mind, randomized algorithms have been devised that need only view the data matrix once [20, 27, 46, 54, 55]. This is unlike classical matrix factorization algorithms, which typically involve a serial process where the data matrix is multiplied numerous times with a vector. Therefore, randomized methods rather than classical algorithms are more suitable for high-performance computing.

Randomized algorithms have gained popularity because of their ability to accelerate many expensive linear algebra tasks and because many randomized algorithms

[*]Submitted to the journal's Methods and Algorithms for Scientific Computing section May 24, 2018; accepted for publication (in revised form) April 16, 2019; published electronically July 25, 2019.
https://doi.org/10.1137/18M118966X

Funding: This work was supported by the NZ Ministry of Business, Innovation and Employment through grant C05X1306, by a scholarship from Landsbankinn, and by an AUEA Braithwaite-Thompson Graduate Research Award.

[†]Department of Engineering Science, University of Auckland, Auckland 1010, New Zealand (ebja558@aucklanduni.ac.nz).

are easy to implement, often requiring only a few lines of code in their most basic form. This study presents improved randomized algorithms for estimating a low-rank factorization of a matrix using an arbitrary number of views or passes over the information contained in the matrix. The algorithms presented here follow the trend of being simple and easy to implement. The focus is on factorizing matrices based on truncated singular value decompositions. However, the ideas presented can also be extended to other factorizations. Although the present study focuses on matrices with real value entries, the algorithms can be extended to the complex case like other related algorithms [20, 46].

First, we consider generalizing a popular randomized subspace iteration algorithm which uses an even number of matrix views. The more practical subspace iteration algorithm presented here works for any number of views $v \geq 2$. Next, for applications that can only afford a single view, we extend and improve state-of-the-art randomized single-pass methods. Finally, using ideas from our generalized subspace iteration method and single-pass methods, we present modified and pass-efficient randomized block Krylov methods. We discuss how the block Krylov approach can be made more efficient when dealing with large matrices that are accessed a few rows at a time. These are all practical methods aimed at generating approximate, but high-quality, low-rank factorizations of large matrices. We also discuss differences between applying randomized algorithms to a given matrix and applying them to its transpose. This can be of interest when trying to estimate dominant singular subspaces or approximating so-called normal matrices, such as those appearing in inverse problems.

1.1. Motivation.

The primary motivation for this study was to develop algorithms to speed up inverse methods used to estimate parameters in models describing subsurface flow in geothermal reservoirs [34, 2]. Inverting models describing complex geophysical processes, such as subsurface flow, often involves matching a large data set using highly parameterized models. Running the model commonly involves solving an expensive and nonlinear forward problem. Despite the possible nonlinearity of the forward problem, the link between the model parameters and simulated observations is often described in terms of a Jacobian matrix $\boldsymbol{J} \in \mathbb{R}^{N_d \times N_m}$, which locally linearizes the relationship between the parameters and observations. The size of $\boldsymbol{J}$ is determined by the (large) parameter and observation spaces. In this case, explicitly forming $\boldsymbol{J}$ is out of the question since at best it involves solving N_m direct problems (linearized forward simulations) or N_d adjoint problems (linearized backward simulations) [6, 21, 33, 36]. Nevertheless, the information contained in $\boldsymbol{J}$ can be helpful for inverting the model using nonlinear inversion methods, such as a Gauss–Newton or Levenberg–Marquardt (LM) approach, and for quantifying uncertainty.

Using adjoint simulation, direct simulation, and randomized algorithms, the necessary information can be extracted efficiently from $\boldsymbol{J}$ without ever explicitly forming the large matrix $\boldsymbol{J}$. Bjarkason et al. [2] showed that inversion of a nonlinear geothermal reservoir model can be accelerated by using randomized low-rank methods coupled with adjoint and direct methods. In [2], randomized 1-view and 2-view methods were chosen because they use the fewest matrix accesses possible. However, in some cases it may be necessary to use more accurate low-rank approximation methods. This may be the case if the singular spectrum of $\boldsymbol{J}$ does not decay rapidly enough for the 1-view or 2-view method to be suitably accurate. Then a more accurate randomized power or subspace iteration method [14, 18, 20, 37] can be used. Another option is to consider pass-efficient randomized block Krylov methods [11, 19, 29, 32, 37]. Standard randomized subspace iteration and block Krylov methods use $2(q+1)$ views to form a low-rank approximation, where q is the number of iterations. However, this leaves out the option of choosing an odd number of views. A more desirable algorithm would allow the user to specify a budget of v views, which could be either odd or even. This is especially important for nonlinear inverse or uncertainty quantification problems, as the matrix views dominate the computational cost, and four or six views could be considered too costly.

1.2. Contributions and outline of the paper. Inspired by similarities between a subspace iteration method presented in the 1990s by [50], which uses an odd number of views $v \geq 3$, and modern randomized subspace iteration methods, we have developed a more general subspace iteration algorithm (see Algorithm 1) which works for any views $v \geq 2$. Algorithm 1 returns an estimated rank-p truncated singular value decomposition (TSVD) of the input matrix $\boldsymbol{A}$, given a target rank p and views $v \geq 2$. The basic idea behind Algorithm 1 is that each extra application of the matrix $\boldsymbol{A}$ improves the approximation. The expected gains in accuracy for each extra view are described by Theorems 3.1 and 4.1. These theorems show that the absolute gain in accuracy achieved by using an extra view is expected to decline exponentially with the number of views. Therefore, stopping at an odd v may suffice. For an even v, Algorithm 1 is a standard subspace iteration method. For an odd v, it is similar to a modified randomized power iteration scheme proposed by [37, sect. 4.3], the difference being that Algorithm 1 applies a subspace iteration approach, and this gives flexibility in terms of the number of matrix views.

Section 3 briefly reviews the standard randomized subspace iteration method. Section 4 presents Algorithm 1 and gives theoretical bounds for the accuracy of the method. Section 4.2 discusses the difference between applying Algorithm 1 to a matrix $\boldsymbol{J}$ and applying it to $\boldsymbol{J}^*$, from the perspective of both cost and accuracy. By considering the approximation of a normal matrix $\boldsymbol{J}^*\boldsymbol{J}$, section 4.2.3 also gives insight into the relationship between Algorithm 1 and algorithms designed for low-rank approximation of positive-semidefinite (psd) matrices. Applying Algorithm 1 one way can be shown to be algebraically equivalent to applying a Nyström-type method [15, 20] to $\boldsymbol{J}^*\boldsymbol{J}$; see section 4.2.3. Another way of using Algorithm 1 is algebraically equivalent to applying a pinched method [15, 20] to $\boldsymbol{J}^*\boldsymbol{J}$; see also section 4.2.3. Additionally, section 4.2.3 gives new insight into the difference between the Nyström and pinched sketching methods. Section 5 outlines how low-rank approximations can be used to solve the type of nonlinear inverse problem motivating this study. Three variants for approximating LM updates are discussed as well as the consequences of choosing certain combinations of model update strategies and low-rank approximation schemes.

To address applications that may only admit one view, section 6 presents modified and improved randomized 1-view algorithms. The 1-view algorithms are mainly based on state-of-the-art algorithms proposed and discussed by [46] but also draw upon algorithms in [55, 20, 25, 3, 48, 57]. Randomized 1-view algorithms use simultaneous sketching of the column and row spaces of a matrix to form an approximation. The general 1-view algorithm proposed and recommended by [46, Alg. 7] has the drawback that the sketches for the column and row spaces cannot be chosen to have the same size without sacrificing robustness. Section 6.4 presents modified 1-view algorithms that enable choosing the same sketch size for the column and row sampling, without sacrificing robustness. We also discuss how these modified 1-view methods allow postprocessing of the resulting sketches to improve accuracy. Additionally, based on [3, 46, 48], section 6.6 provides an improved extended 1-view algorithm.

Section 7 outlines improved and more pass-efficient randomized block Krylov algorithms. The algorithms are aimed at problems where multiplications with the data matrix and its transpose are expensive. This can be the case when dealing with large matrices stored out-of-core or Jacobian matrices appearing in large-scale inverse problems. Like the generalized subspace iteration method, our proposed Krylov algorithm works for $v \geq 2$. We also suggest improvements for problems dealing with large matrices which are accessed a few rows at a time from out-of-core memory. In this case the Krylov approach can be up to twice as pass-efficient as previously proposed block Krylov methods [12, 20, 29, 32, 37].

Section 8 gives experimental results that demonstrate the accuracy of the presented randomized algorithms and support claims made in previous sections. The randomized algorithms in this study have been coded using Python. The experimental code is available at https://github.com/ebjarkason/RandomSVDanyViews and includes code for the experiments considered in section 8. Some of the Python algorithms were designed with adjoint and direct methods in mind. That is, they allow the user to specify functions for evaluating the action of the matrix of interest and its transpose on other matrices.

2. Preliminaries.

This study focuses on approximation methods by way of singular value decomposition (SVD) of thin, low-rank matrices. The SVD of $\boldsymbol{A} \in \mathbb{R}^{n_r \times n_c}$ is

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{V}^* = \sum_{i=1}^{N} \lambda_i \boldsymbol{u}_i \boldsymbol{v}_i^*, \tag{2.1}$$

where $N = \min(n_r, n_c)$. The matrices $\boldsymbol{U} = [\boldsymbol{u}_1\, \boldsymbol{u}_2\, \dots\, \boldsymbol{u}_N]$ and $\boldsymbol{V} = [\boldsymbol{v}_1\, \boldsymbol{v}_2\, \dots\, \boldsymbol{v}_N]$ have orthonormal columns, where $\boldsymbol{u}_i$ and $\boldsymbol{v}_i$ are the left and right singular vectors of the ith singular value λ_i. The singular values $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_N \geq 0$ are contained in the diagonal matrix $\boldsymbol{\Lambda} = \operatorname{diag}[\lambda_1, \lambda_2, \dots, \lambda_N]$. We let $\boldsymbol{X}^*$ denote the conjugate transpose of the matrix $\boldsymbol{X}$.

For a matrix $\hat{\boldsymbol{A}}$ that is a rank-p approximate TSVD of $\boldsymbol{A}$, we use

$$\hat{\boldsymbol{A}} = \sum_{i=1}^{p} \lambda_i \boldsymbol{u}_i \boldsymbol{v}_i^* = \boldsymbol{U}_p \boldsymbol{\Lambda}_p \boldsymbol{V}_p^*. \tag{2.2}$$

Here λ_i, $\boldsymbol{u}_i$, and $\boldsymbol{v}_i$ can denote either the approximate or the exact ith singular value and vectors of $\boldsymbol{A}$. For exact values, $\hat{\boldsymbol{A}}$ is the optimal rank-p approximation (in terms of the spectral and Frobenius norms), which is denoted by $[\![\boldsymbol{A}]\!]_p$. $[\boldsymbol{U}_p, \boldsymbol{\Lambda}_p, \boldsymbol{V}_p] = \operatorname{tsvd}(\boldsymbol{A}, p)$ denotes a function returning the SVD of $[\![\boldsymbol{A}]\!]_p$. Similarly, when $n_r = n_c$ and $\boldsymbol{A}$ is Hermitian psd, $[\boldsymbol{\Lambda}_p, \boldsymbol{V}_p] = \operatorname{tevd}(\boldsymbol{A}, p)$ denotes finding the eigenvalue decomposition of $[\![\boldsymbol{A}]\!]_p = \boldsymbol{V}_p \boldsymbol{\Lambda}_p \boldsymbol{V}_p^*$. We use the spectral norm, denoted by $\|\cdot\|$, and the Frobenius norm, denoted by $\|\cdot\|_{\mathrm{F}}$, to quantify the accuracy of the low-rank approximations.

In what follows, the QR decomposition of $\boldsymbol{A} \in \mathbb{R}^{n_r \times n_c}$, with $n_r \geq n_c$, is defined as $\boldsymbol{Q}\boldsymbol{R} := \boldsymbol{A}$, where $\boldsymbol{Q}$ is an $n_r \times n_c$ matrix with orthonormal columns and $\boldsymbol{R}$ is an $n_c \times n_c$ upper-triangular matrix. Finding such a thin or economic QR-factorization is expressed by $[\boldsymbol{Q}, \boldsymbol{R}] = \operatorname{qr}(\boldsymbol{A})$. Similarly, an $n_r \times r$ matrix $\boldsymbol{Q}$ that has orthonormal columns which form a basis for the range of a matrix $\boldsymbol{A}$ of rank r is denoted by $\boldsymbol{Q} = \operatorname{orth}(\boldsymbol{A})$. In the presented algorithms, we use $\operatorname{randn}(m, n)$ to denote an $m \times n$ Gaussian random matrix. For a Hermitian positive-definite matrix $\boldsymbol{A}$, we use $\boldsymbol{C}_L = \operatorname{chol}(\boldsymbol{A}, \mathit{LOWER})$ to express finding a lower-triangular Cholesky matrix $\boldsymbol{C}_L$ such that $\boldsymbol{A} = \boldsymbol{C}_L \boldsymbol{C}_L^*$.

3. Standard randomized subspace iteration.

A low-rank approximation of an $n_r \times n_c$ matrix $\boldsymbol{A}$ can be found by applying a simple two-stage randomized procedure [14, 20, 25]. The first stage is a randomized range finder, and the second stage uses the approximate range of $\boldsymbol{A}$ to construct the desired low-rank approximation.

1. (*Randomized stage*) Find an $n_r \times k$ matrix $\boldsymbol{Q}_c$ whose columns are an approximate orthonormal basis for the range (column space) of $\boldsymbol{A}$, such that $\boldsymbol{A} \approx \boldsymbol{Q}_c \boldsymbol{Q}_c^* \boldsymbol{A}$.
2. (*Deterministic stage*) Evaluate $\boldsymbol{B} = \boldsymbol{Q}_c^* \boldsymbol{A}$. Then $\boldsymbol{Q}_c \boldsymbol{B}$, is a rank-$k$ approximation of $\boldsymbol{A}$. This approximate QB-decomposition can be used to construct other approximate low-rank factorizations of $\boldsymbol{A}$, such as the TSVD.

The matrix $\boldsymbol{Q}_c$ can be generated cheaply by accessing or viewing the matrix $\boldsymbol{A}$ only once. If we draw a random matrix $\boldsymbol{\Omega}_r \in \mathbb{R}^{n_c \times k}$, then $\boldsymbol{Y}_c = \boldsymbol{A}\boldsymbol{\Omega}_r$ sketches the range of $\boldsymbol{A}$. In this study each element of $\boldsymbol{\Omega}_r$ is drawn independently at random from a standard Gaussian distribution. However, other types of sampling matrices can also be considered [14, 20, 22, 46]. For robust outcomes, k should be larger than the target rank p. That is, choose $k = p + l$, where l is an oversampling factor required for increased accuracy (e.g., $l = 10$ [18, 20, 25]). Subsequently, QR-decomposition can, for instance, be used to find the orthonormal matrix $\boldsymbol{Q}_c$, i.e., $\boldsymbol{Y}_c = \boldsymbol{Q}_c \boldsymbol{R}_c$. With this basis, we can form the thin matrix $\boldsymbol{B}$ in stage 2 by one extra view of $\boldsymbol{A}$. Taking $\boldsymbol{A} \approx \boldsymbol{Q}_c [\![\boldsymbol{B}]\!]_p = \boldsymbol{U}_p \boldsymbol{\Lambda}_p \boldsymbol{V}_p^*$ gives the basic randomized SVD algorithm [18, 25], which is a 2-view method.

The basic 2-view method works well when the singular values decay rapidly. However, it may lack accuracy for some applications. In that case, power or subspace iteration can be used to improve the randomized approach [14, 18, 20, 37]. Using q power iterations, an improved orthonormal matrix $\boldsymbol{Q}_c$ is found using $(\boldsymbol{A}\boldsymbol{A}^*)^q \boldsymbol{A}\boldsymbol{\Omega}_r$ instead of $\boldsymbol{A}\boldsymbol{\Omega}_r$. A drawback is that for each iteration, the matrix $\boldsymbol{A}$ is accessed two additional times. The spectral norm error of the randomized approximation $\boldsymbol{A} \approx \boldsymbol{Q}_c \boldsymbol{Q}_c^* \boldsymbol{A}$ is expected to improve exponentially with q [20, 37]; see the following theorem from [20].

THEOREM 3.1 (average spectral error for the power scheme [20]). *Let $\boldsymbol{A} \in \mathbb{R}^{n_r \times n_c}$ with nonnegative singular values $\lambda_1 \geq \lambda_2 \geq \cdots$, let $p \geq 2$ be the target rank, and let $l \geq 2$ be an oversampling parameter, with $p + l \leq \min(n_r, n_c)$. Draw a Gaussian random matrix $\boldsymbol{\Omega}_r \in \mathbb{R}^{n_c \times (p+l)}$, and set $\boldsymbol{Y}_c = (\boldsymbol{A}\boldsymbol{A}^*)^q \boldsymbol{A}\boldsymbol{\Omega}_r$ for an integer $q \geq 0$. Let $\boldsymbol{Q}_c \in \mathbb{R}^{n_r \times (p+l)}$ be an orthonormal matrix which forms a basis for the range of $\boldsymbol{Y}_c$. Then*

$$\mathbb{E}\left[\|\boldsymbol{A} - \boldsymbol{Q}_c \boldsymbol{Q}_c^* \boldsymbol{A}\|\right] \leq \left[\left(1 + \sqrt{\frac{p}{l-1}}\right) \lambda_{p+1}^{2q+1} + \frac{\mathrm{e}\sqrt{p+l}}{l} \left(\sum_{j>p} \lambda_j^{2(2q+1)}\right)^{1/2}\right]^{1/(2q+1)}.$$

In floating point arithmetic, subspace iteration is a numerically more robust extension of the power iteration scheme. The standard subspace iteration method views the input matrix $\boldsymbol{A}$ an even $2(q+1)$ times. To suit any budget of views, a low-rank approximation algorithm that works for any number of views is appealing. The following section presents a more general subspace iteration algorithm, which gives an approximate TSVD of a matrix for any number of views greater than one. Section 6 discusses algorithms that use one view.

4. Generalized randomized subspace iteration.

The basic idea of the generalized subspace iteration method presented here is that the standard subspace iteration process can be halted halfway through an iteration to give an orthonormal matrix $\boldsymbol{Q}_r$ which approximates the corange (row space) of $\boldsymbol{A}$. This equates to evaluating the corange sketch $\boldsymbol{Y}_r = (\boldsymbol{A}^*\boldsymbol{A})^q\boldsymbol{\Omega}_r$ for $q \geq 1$, which is like performing $q - 1/2$ power iterations. Then, an orthonormal basis $\boldsymbol{Q}_r$ for the range of $\boldsymbol{Y}_r$ gives $\boldsymbol{A} \approx \boldsymbol{A}\boldsymbol{Q}_r\boldsymbol{Q}_r^*$ using $2q + 1$ views.

The generalized subspace iteration algorithm is given by Algorithm 1. For $v \geq 2$ views, Algorithm 1 returns an approximate TSVD $\boldsymbol{U}_p\boldsymbol{\Lambda}_p\boldsymbol{V}_p^*$ of $\boldsymbol{A}$ by one of the following approaches:

1. (*If v is even*) Find an orthonormal matrix $\boldsymbol{Q}_c$ whose columns form a basis for the range of $(\boldsymbol{A}\boldsymbol{A}^*)^{(v-2)/2}\boldsymbol{A}\boldsymbol{\Omega}_r$. Then find the rank-$p$ TSVD $\boldsymbol{V}_p\boldsymbol{\Lambda}_p\hat{\boldsymbol{U}}_p^*$ of $\boldsymbol{A}^*\boldsymbol{Q}_c$, and set $\boldsymbol{U}_p = \boldsymbol{Q}_c\hat{\boldsymbol{U}}_p$.
2. (*If v is odd*) Find an orthonormal matrix $\boldsymbol{Q}_r$ whose columns form a basis for the range of $(\boldsymbol{A}^*\boldsymbol{A})^{(v-1)/2}\boldsymbol{\Omega}_r$. Then find the rank-$p$ TSVD $\boldsymbol{U}_p\boldsymbol{\Lambda}_p\hat{\boldsymbol{V}}_p^*$ of $\boldsymbol{A}\boldsymbol{Q}_r$, and set $\boldsymbol{V}_p = \boldsymbol{Q}_r\hat{\boldsymbol{V}}_p$.

For an even number of matrix views, Algorithm 1 proceeds by using the standard subspace iteration method. For odd v, Algorithm 1 is similar to and uses the same number of views as a modified power iteration method proposed by [37, sect. 4.3]. The main difference is that Algorithm 1 uses the subspace iteration framework to give a practical algorithm that works for any budget of views $v \geq 2$.

The generalized subspace iteration method (Algorithm 1) can be varied in the same ways as the standard subspace iteration method. For instance, Algorithm 1 can be modified to use cheaper LU-factorizations instead of QR-factorizations to normalize the algorithm after each of the first $v - 2$ views. Another option may be to skip some of the renormalization steps. These types of alternative renormalization schemes are presented for the standard subspace or power iteration approaches in [14, 22, 51].

The present study focuses on methods for generating approximate TSVD factorizations. Nevertheless, the general half-iteration ideas presented here can also be applied to other low-rank methods that use an approximate range or corange to form a factorization. The half-iteration approach, for instance, can be extended to form a partial pivoted QR-factorization [30], a nonnegative matrix factorization [13], randomized CUR and interpolative decompositions [25], and a UTV decomposition [26].

As presented, Algorithm 1 uses a predefined number of views v given by the user. However, to avoid using an unnecessarily large v, it is possible to use a convergence scheme, such as the one used in [50], to halt the algorithm when the estimated singular values contained in the matrices $\boldsymbol{R}_c$ and $\boldsymbol{R}_r$ appear to have converged sufficiently. However, to guarantee a low-rank approximation which satisfies a predefined accuracy, the randomized blocked algorithm presented in [30] can be used instead. Still, the generalized subspace method presented here can be used, instead of the standard subspace method, within the adaptive and accuracy enhanced blocked algorithm given by [30].

4.1. Accuracy of Algorithm 1.

For an even number of views, Algorithm 1 equates to using standard subspace iteration, and the expected error is given by Theorem 3.1. For odd v, the expected error of Algorithm 1 is given by the following theorem. A proof for Theorem 4.1 is given in section SM1 in the supplementary materials. The proof is based on the approach used in [20] to prove Theorem 3.1.

THEOREM 4.1 (average spectral error for half-power scheme). *Let $\boldsymbol{A} \in \mathbb{R}^{n_r \times n_c}$ with nonnegative singular values $\lambda_1 \geq \lambda_2 \geq \cdots$, let $p \geq 2$ be the target rank, and let $l \geq 2$ be an oversampling parameter, with $p + l \leq \min(n_r, n_c)$. Draw a Gaussian*

Algorithm 1. Randomized SVD using generalized subspace iteration.

```
INPUT: Matrix $\boldsymbol{A} \in \mathbb{R}^{n_r \times n_c}$, integers $p > 0$, $l \geq 0$, and $v \geq 2$.
RETURNS: Approximate rank-$p$ SVD, $\boldsymbol{U}_p \boldsymbol{\Lambda}_p \boldsymbol{V}_p^*$, of $\boldsymbol{A}$.
 1: $\boldsymbol{Q}_r = \mathrm{randn}(n_c,\, p + l)$.
 2: for $j = 1$ to $v$ do
 3:     if $j$ is odd then
 4:         $[\boldsymbol{Q}_c,\, \boldsymbol{R}_c] = \mathrm{qr}(\boldsymbol{A}\boldsymbol{Q}_r)$.
 5:     else
 6:         $[\boldsymbol{Q}_r,\, \boldsymbol{R}_r] = \mathrm{qr}(\boldsymbol{A}^*\boldsymbol{Q}_c)$.
 7: if $v$ is even then
 8:     $[\hat{\boldsymbol{V}}_p,\, \boldsymbol{\Lambda}_p,\, \hat{\boldsymbol{U}}_p] = \mathrm{tsvd}(\boldsymbol{R}_r,\, p)$.
 9: else
10:     $[\hat{\boldsymbol{U}}_p,\, \boldsymbol{\Lambda}_p,\, \hat{\boldsymbol{V}}_p] = \mathrm{tsvd}(\boldsymbol{R}_c,\, p)$.
11: $\boldsymbol{U}_p = \boldsymbol{Q}_c \hat{\boldsymbol{U}}_p$ and $\boldsymbol{V}_p = \boldsymbol{Q}_r \hat{\boldsymbol{V}}_p$.
```

random matrix $\boldsymbol{\Omega}_r \in \mathbb{R}^{n_c \times (p+l)}$*, and set* $\boldsymbol{Y}_r = (\boldsymbol{A}^*\boldsymbol{A})^q \boldsymbol{\Omega}_r$ *for an integer* $q \geq 1$*. Let* $\boldsymbol{Q}_r \in \mathbb{R}^{n_c \times (p+l)}$ *be an orthonormal matrix which forms a basis for the range of* $\boldsymbol{Y}_r$*. Then*

$$\mathbb{E}\left[\|\boldsymbol{A} - \boldsymbol{A}\boldsymbol{Q}_r\boldsymbol{Q}_r^*\|\right] \leq \left[\left(1 + \sqrt{\frac{p}{l-1}}\right)\lambda_{p+1}^{2q} + \frac{\mathrm{e}\sqrt{p+l}}{l}\left(\sum_{j>p}\lambda_j^{4q}\right)^{1/2}\right]^{1/(2q)}.$$

The bounds in Theorem 4.1 are what we could expect based on Theorem 3.1. That is, the expected error decays exponentially with the number of views. Theorem 3.1 states that the expected error for $\boldsymbol{A} \approx \boldsymbol{Q}_c\boldsymbol{Q}_c^*\boldsymbol{A}$ depends on $2q+1$ when using $2q+1$ views to form $\boldsymbol{Q}_c$. Theorem 4.1 states similarly that the expected error for $\boldsymbol{A} \approx \boldsymbol{A}\boldsymbol{Q}_r\boldsymbol{Q}_r^*$ depends on $2q$ when using $2q$ views to form $\boldsymbol{Q}_r$. Theorems 3.1 and 4.1 show that it is important for the accuracy of Algorithm 1 that the spectrum decays rapidly.

4.2. Apply Algorithm 1 to a matrix or its transpose?

An approximate TSVD of a matrix $\boldsymbol{J}$ can be found using Algorithm 1 with either $\boldsymbol{J}$ or $\boldsymbol{J}^*$ as input. However, it may be worth considering that these two approaches can differ in cost. Also, depending on the application, these two approaches can have different effects on the accuracy of downstream tasks, which use the low-rank approximation. The latter may seem surprising considering that the expected quality for an approximate TSVD of $\boldsymbol{J}$ is the same whether we apply Algorithm 1 to $\boldsymbol{J}$ or $\boldsymbol{J}^*$.

4.2.1. Computational cost.

Considering computational cost, the choice between applying Algorithm 1 to $\boldsymbol{J}$ and applying it to $\boldsymbol{J}^*$ depends on a few factors. For even v, the two options only differ when it comes to generating the Gaussian sampling matrix. The cost of generating $\boldsymbol{\Omega}_r$ is $(p+l)n_c T_{\mathrm{rand}}$, where T_{rand} is the cost of generating a Gaussian random number. In that case, to reduce cost, $\boldsymbol{J}^*$ should be used as input when $\boldsymbol{J}$ has more columns than rows.

However, for odd v, the cost to consider when choosing between applying Algorithm 1 to $\boldsymbol{J}$ or to $\boldsymbol{J}^*$ is $\mathscr{O}([p+l]n_c T_{\mathrm{rand}} + [p+l]T_{\mathrm{mult}} + [p+l]^2 n_r)$, where T_{mult} indicates the cost of multiplying the input matrix by a vector, and the last term comes from the QR-factorization on line 4 in Algorithm 1. For the problem motivating this study, the dominant cost is from the matrix multiplication, and in some cases there may be a noticeable difference between the cost of evaluating $\boldsymbol{J}$ or $\boldsymbol{J}^*$ times a matrix. For physics-based simulations, $\boldsymbol{J}$ can represent a Jacobian matrix, which gives a local linear mapping between the model inputs (parameters) and outputs (e.g., observations or predictions). Then $\boldsymbol{J}$ times a matrix can be evaluated efficiently using a direct method, and $\boldsymbol{J}^*$ times a matrix can be evaluated using an adjoint method. The adjoint runs require more memory and can therefore be more costly. In that case it might be best to apply Algorithm 1 to $\boldsymbol{J}$, when v is odd, to lower the cost. This is likewise the case when using automatic differentiation [4, 17], where the forward and backward modes correspond to the direct and adjoint methods, respectively.

4.2.2. Computational accuracy. To evaluate accuracy we can look at the bounds in Theorems 3.1 and 4.1. For even v Algorithm 1 gives matrices $\boldsymbol{Q}_c$ and $\boldsymbol{Q}_r$, which form approximate bases for the range and corange of the input matrix $\boldsymbol{A}$, such that

(4.1)
$$\mathbb{E}\left[\|\boldsymbol{A} - \boldsymbol{Q}_c\boldsymbol{Q}_c^*\boldsymbol{A}\|\right] \leq \left[\left(1 + \sqrt{\frac{p}{l-1}}\right)\lambda_{p+1}^{v-1} + \frac{\mathrm{e}\sqrt{p+l}}{l}\left(\sum_{j>p}\lambda_j^{2(v-1)}\right)^{1/2}\right]^{1/(v-1)};$$

(4.2)
$$\mathbb{E}\left[\|\boldsymbol{A} - \boldsymbol{A}\boldsymbol{Q}_r\boldsymbol{Q}_r^*\|\right] \leq \left[\left(1 + \sqrt{\frac{p}{l-1}}\right)\lambda_{p+1}^{v} + \frac{\mathrm{e}\sqrt{p+l}}{l}\left(\sum_{j>p}\lambda_j^{2v}\right)^{1/2}\right]^{1/v}.$$

However, for odd v the error bounds for the final matrices $\boldsymbol{Q}_c$ and $\boldsymbol{Q}_r$ in Algorithm 1 are

(4.3)
$$\mathbb{E}\left[\|\boldsymbol{A} - \boldsymbol{Q}_c\boldsymbol{Q}_c^*\boldsymbol{A}\|\right] \leq \left[\left(1 + \sqrt{\frac{p}{l-1}}\right)\lambda_{p+1}^{v} + \frac{\mathrm{e}\sqrt{p+l}}{l}\left(\sum_{j>p}\lambda_j^{2v}\right)^{1/2}\right]^{1/v};$$

(4.4)
$$\mathbb{E}\left[\|\boldsymbol{A} - \boldsymbol{A}\boldsymbol{Q}_r\boldsymbol{Q}_r^*\|\right] \leq \left[\left(1 + \sqrt{\frac{p}{l-1}}\right)\lambda_{p+1}^{v-1} + \frac{\mathrm{e}\sqrt{p+l}}{l}\left(\sum_{j>p}\lambda_j^{2(v-1)}\right)^{1/2}\right]^{1/(v-1)}.$$

Noting that the left and right singular vectors are estimated as a linear combination of the columns of $\boldsymbol{Q}_r$ and $\boldsymbol{Q}_c$ according to $\boldsymbol{U}_p = \boldsymbol{Q}_c\hat{\boldsymbol{U}}_p$ and $\boldsymbol{V}_p = \boldsymbol{Q}_r\hat{\boldsymbol{V}}_p$, we see that the above error bounds suggest that the accuracy of the left-singular vectors may differ from that of the right-singular vectors. When v is even, $\boldsymbol{V}_p$ should be more accurate than, or as accurate as, $\boldsymbol{U}_p$, but when v is odd, $\boldsymbol{U}_p$ should be more accurate than, or as accurate as, $\boldsymbol{V}_p$.

This distinction may be important when the goal is not only the low-rank approximation of a matrix $\boldsymbol{J}$ but for instance if we are mainly interested in its right-singular vectors and do not care (or care less) about the left-singular vectors. Section 5 discusses applications where this is of interest. One such application is estimating a truncated eigenvalue-decomposition (TEVD) of the normal matrix $\boldsymbol{J}^*\boldsymbol{J}$. In that case we need the leading right-singular vectors and singular values of $\boldsymbol{J}$. Using even v, it

Algorithm 2. Nyström approach for a truncated eigendecomposition of $\boldsymbol{J}^*\boldsymbol{J}$.

```
INPUT: Matrix $\boldsymbol{J} \in \mathbb{R}^{n_r \times n_c}$, integers $p > 0$, $l \geq 0$, and $v \geq 2$.
RETURNS: Approximate rank-$p$ EVD, $\boldsymbol{V}_p \boldsymbol{\Lambda}_p^2 \boldsymbol{V}_p^*$, of $\boldsymbol{J}^*\boldsymbol{J}$.
1: if $v$ is even then
2:     $\boldsymbol{\Omega}_r = \text{randn}(n_c,\, p + l)$.
3:     Use subspace iteration to evaluate $\boldsymbol{Q}_r = \text{orth}([\boldsymbol{J}^*\boldsymbol{J}]^{(v-2)/2}\boldsymbol{\Omega}_r)$.
4: else
5:     $\boldsymbol{\Omega}_c = \text{randn}(n_r,\, p + l)$.
6:     Use subspace iteration to evaluate $\boldsymbol{Q}_r = \text{orth}([\boldsymbol{J}^*\boldsymbol{J}]^{(v-3)/2}\boldsymbol{J}^*\boldsymbol{\Omega}_c)$.
7: $\boldsymbol{Y}_r = \boldsymbol{J}^*[\boldsymbol{J}\boldsymbol{Q}_r]$.
8: $\boldsymbol{Y}_r \leftarrow \boldsymbol{Y}_r + \nu \boldsymbol{Q}_r$, where $\nu = 2.2 \cdot 10^{-16}\, \|\boldsymbol{Y}_r\|$.
9: $\boldsymbol{B} = \boldsymbol{Q}_r^* \boldsymbol{Y}_r$.
10: $\boldsymbol{C}_L = \text{chol}([\boldsymbol{B} + \boldsymbol{B}^*]/2,\, LOWER)$.
11: $[\sim,\, \hat{\boldsymbol{\Lambda}}_p,\, \boldsymbol{V}_p,] = \text{tsvd}(\boldsymbol{C}_L^{-1}\boldsymbol{Y}_r^*,\, p)$.
12: $\boldsymbol{\Lambda}_p^2 = \hat{\boldsymbol{\Lambda}}_p^2 - \nu \boldsymbol{I}$, and set all negative elements of $\boldsymbol{\Lambda}_p^2$ to 0.
```

can be more accurate to apply Algorithm 1 to $\boldsymbol{J}$. However, for odd v it is better to input $\boldsymbol{J}^*$. The following subsection shows that this strategy equates to applying a popular low-rank approximation method, meant for psd matrices, to the normal matrix $\boldsymbol{J}^*\boldsymbol{J}$. Recent studies in [11, 39] consider the accuracy of approximate principal subspaces generated by the standard randomized subspace [39] and block Krylov [11] methods.

4.2.3. Link to standard methods for approximating a normal matrix.

The so-called prolonged, or Nyström type, and pinched sketching methods have been discussed extensively in the literature for approximating a psd matrix $\boldsymbol{A}_{\text{psd}}$; see, e.g., [15, 20]. Given an approximate range $\boldsymbol{Q}_r$ for $\boldsymbol{A}_{\text{psd}}$, the pinched method gives the following approximation:

$$\boldsymbol{A}_{\text{psd}} \approx \boldsymbol{Q}_r(\boldsymbol{Q}_r^*\boldsymbol{A}_{\text{psd}}\boldsymbol{Q}_r)\boldsymbol{Q}_r^*. \tag{4.5}$$

The prolonged method, on the other hand, uses

$$\boldsymbol{A}_{\text{psd}} \approx (\boldsymbol{A}_{\text{psd}}\boldsymbol{Q}_r)[\boldsymbol{Q}_r^*\boldsymbol{A}_{\text{psd}}\boldsymbol{Q}_r]^{-1}(\boldsymbol{A}_{\text{psd}}\boldsymbol{Q}_r)^*. \tag{4.6}$$

Assuming $\boldsymbol{Q}_r$ and a straightforward implementation, the pinched and prolonged approaches have the same cost when it comes to multiplication with $\boldsymbol{A}_{\text{psd}}$. Although in this case they have similar cost, (4.6) is more accurate than (4.5) [15, 20].

A psd matrix can generally be written as a normal matrix $\boldsymbol{A}_{\text{psd}} = \boldsymbol{J}^*\boldsymbol{J}$. Then the pinched sketch can be written as

$$\boldsymbol{J}^*\boldsymbol{J} \approx \boldsymbol{Q}_r(\boldsymbol{Q}_r^*\boldsymbol{J}^*\boldsymbol{J}\boldsymbol{Q}_r)\boldsymbol{Q}_r^* = (\boldsymbol{Q}_r\boldsymbol{Q}_r^*\boldsymbol{J}^*)(\boldsymbol{J}\boldsymbol{Q}_r\boldsymbol{Q}_r^*). \tag{4.7}$$

Assume for now that $\boldsymbol{Q}_r$ is formed in a standard fashion by applying q power iterations to $\boldsymbol{A}_{\text{psd}}$, with $\boldsymbol{Q}_r = \text{orth}(\boldsymbol{A}_{\text{psd}}^q\boldsymbol{\Omega}_r)$. Then (4.7) is algebraically equivalent to applying Algorithm 1 to $\boldsymbol{J}$ with $2q+1$ views to find $\boldsymbol{J} \approx \boldsymbol{J}\boldsymbol{Q}_r\boldsymbol{Q}_r^*$ and then forming the low-rank TEVD of $\boldsymbol{J}^*\boldsymbol{J}$. For the prolonged sketch, we can write

$$\boldsymbol{J}^*\boldsymbol{J} \approx \boldsymbol{J}^*(\boldsymbol{J}\boldsymbol{Q}_r)[(\boldsymbol{J}\boldsymbol{Q}_r)^*(\boldsymbol{J}\boldsymbol{Q}_r)]^{-1}(\boldsymbol{J}\boldsymbol{Q}_r)^*\boldsymbol{J} = \boldsymbol{J}^*\boldsymbol{Q}_c\boldsymbol{Q}_c^*\boldsymbol{J} = (\boldsymbol{J}^*\boldsymbol{Q}_c\boldsymbol{Q}_c^*)(\boldsymbol{Q}_c\boldsymbol{Q}_c^*\boldsymbol{J}), \tag{4.8}$$

where $\boldsymbol{Q}_c$ is an orthonormal basis for $\boldsymbol{J}\boldsymbol{Q}_r$. Therefore, using the prolonged method is like using Algorithm 1 to find $\boldsymbol{J} \approx \boldsymbol{Q}_c\boldsymbol{Q}_c^*\boldsymbol{J}$. Assuming again that $\boldsymbol{Q}_r = \text{orth}(\boldsymbol{A}_{\text{psd}}^q\boldsymbol{\Omega}_r)$,

Algorithm 3. Using pinched sketch for a truncated eigendecomposition of $\boldsymbol{J}^*\boldsymbol{J}$.

INPUT: Matrix $\boldsymbol{J} \in \mathbb{R}^{n_r \times n_c}$, integers $p > 0$, $l \geq 0$, and $v \geq 2$.
RETURNS: Approximate rank-p EVD, $\boldsymbol{V}_p\boldsymbol{\Lambda}_p^2\boldsymbol{V}_p^*$, of $\boldsymbol{J}^*\boldsymbol{J}$.

```
1: if $v$ is even then
2:     $\boldsymbol{\Omega}_c = \text{randn}(n_r,\, p+l)$.
3:     Use subspace iteration to evaluate $\boldsymbol{Q}_r = \text{orth}([\boldsymbol{J}^*\boldsymbol{J}]^{(v-2)/2}\boldsymbol{J}^*\boldsymbol{\Omega}_c)$.
4: else
5:     $\boldsymbol{\Omega}_r = \text{randn}(n_c,\, p+l)$.
6:     Use subspace iteration to evaluate $\boldsymbol{Q}_r = \text{orth}([\boldsymbol{J}^*\boldsymbol{J}]^{(v-1)/2}\boldsymbol{\Omega}_r)$.
7: $\boldsymbol{B}_c = \boldsymbol{J}\boldsymbol{Q}_r$.
8: $[\boldsymbol{\Lambda}_p^2,\, \hat{\boldsymbol{V}}_p] = \text{tevd}(\boldsymbol{B}_c^*\boldsymbol{B}_c,\, p)$.
9: $\boldsymbol{V}_p = \boldsymbol{Q}_r\hat{\boldsymbol{V}}_p$.
```

then the prolonged scheme is algebraically equivalent to applying Algorithm 1 to $\boldsymbol{J}$ using $2q+2$ views. This is a generalization of the observation made in [15] that for $\boldsymbol{Q}_r = \text{orth}(\boldsymbol{A}_{\text{psd}}\boldsymbol{\Omega}_r)$, using the prolonged method is like applying a 4-view method to $\boldsymbol{A}_{\text{psd}}^{1/2}$, and using the pinched method is like viewing $\boldsymbol{A}_{\text{psd}}^{1/2}$ three times. Therefore, it is clear that using the standard prolonged method is more accurate than using the pinched one because it is like applying one extra view. Furthermore, (4.6) corresponds to using Algorithm 1 to estimate $\boldsymbol{J}^*\boldsymbol{J}$ in the more accurate way using $2q+2$ views (see section 4.2.2). However, (4.5) corresponds to using Algorithm 1 to estimate $\boldsymbol{J}^*\boldsymbol{J}$ in the less accurate way with $2q+1$ views.

For the above discussion the factorization $\boldsymbol{A}_{\text{psd}} = \boldsymbol{J}^*\boldsymbol{J}$ was mainly used to demonstrate differences between (4.5) and (4.6), and it was not assumed that $\boldsymbol{J}$ could be applied separately. However, if we can multiply $\boldsymbol{J}^*$ and $\boldsymbol{J}$ separately with matrices, then the pinched and prolonged methods can be implemented more generally given a budget of views v for $\boldsymbol{J}$. The modified prolonged method is given in Algorithm 2, which is based on a single-pass Nyström algorithm in [45]. Algorithm 2 uses $v-2$ views of $\boldsymbol{J}$ to generate an approximate basis $\boldsymbol{Q}_r$. Then, a TEVD is formed based on (4.8), with the last two views. Considering these steps and (4.8), we see that this is like the recommended accurate way of using Algorithm 1 to approximate $\boldsymbol{J}^*\boldsymbol{J}$; see section 4.2.2.

Similarly, Algorithm 3 gives a pinched approach to approximate $\boldsymbol{J}^*\boldsymbol{J}$. Algorithm 3 uses $v-1$ views to form $\boldsymbol{Q}_r$ instead of $v-2$ views. Using the final view, an approximation is formed based on (4.7). This equates to not following the recommendations given in section 4.2.2 for using Algorithm 1 to approximate $\boldsymbol{J}^*\boldsymbol{J}$. Section 8.3.2 demonstrates that Algorithm 2 is more accurate than Algorithm 3. As mentioned, we can use these methods or Algorithm 1 to estimate $\boldsymbol{J}^*\boldsymbol{J}$. We have preferred using Algorithm 1 since it allows the flexibility of generating an approximate TSVD of $\boldsymbol{J}$ or $\boldsymbol{J}^*\boldsymbol{J}$ as needed.

5. Nonlinear least-squares inversion.

Following from the above discussion, this section briefly considers the application motivating this study as an example where the recommendations given in sections 4.2.2 and 4.2.3 are useful.

Solving linear matrix equations is a routine task in scientific computing but becomes expensive for large matrices. Various randomized methods have been presented for solving linear equations or linear regression [1, 7, 12, 16, 31, 38, 40, 56]. The algorithms in this study were motivated by a study in [2] which considered randomized methods for updating parameters of a nonlinear reservoir model using a modified LM approach. Another topic of interest is using randomized methods to estimate the posterior probability distribution for the parameters of a nonlinear model using a Laplace approximation [5, 10, 9], which involves approximating a normal matrix $\boldsymbol{J}^*\boldsymbol{J}$.

5.1. Levenberg–Marquardt update. An assumption commonly used for inversion of reservoir models is that the distributions for the observations and the prior of the model parameters can be described by Gaussian statistics. That is, the statistics for the observations and prior can be described by mean values and a covariance matrix. Here we apply a linear whitening transformation such that both the model parameters $\boldsymbol{x} \in \mathbb{R}^{N_m}$ and observation error $\boldsymbol{d}(\boldsymbol{x}) \in \mathbb{R}^{N_d}$ are associated with a zero mean and a covariance matrix that is an appropriately sized identity matrix. The inversion task is to solve

$$\underset{\boldsymbol{x}}{\operatorname{argmin}} \; \|\boldsymbol{d}(\boldsymbol{x})\|^2 + \mu \|\boldsymbol{x}\|^2 , \tag{5.1}$$

where μ is a regularization weight. Using the LM approach, (5.1) is tackled by an iterative procedure where model updates are found by

$$[\boldsymbol{J}^*\boldsymbol{J} + (\mu+\gamma)\boldsymbol{I}]\,\delta\boldsymbol{x} = -\boldsymbol{J}^*\boldsymbol{d} - \mu\boldsymbol{x}, \tag{5.2}$$

and models are updated by $\boldsymbol{x} \leftarrow \boldsymbol{x} + \delta\boldsymbol{x}$. Here $\gamma > 0$ is the adjustable LM damping factor and $\boldsymbol{J} \in \mathbb{R}^{N_d \times N_m}$ is the Jacobian matrix which defines an approximate linear mapping from $\boldsymbol{x}$ to $\boldsymbol{d}(\boldsymbol{x})$. In [2, 41, 42, 43] $\boldsymbol{J}$ is referred to as the dimensionless sensitivity matrix. For large N_d and N_m, generating the often dense Jacobian and solving (5.2) becomes costly. However, for a nonlinear problem, an exact solution to (5.2) is not necessarily needed, and approximate solutions are often used to save time. After applying a randomized method to find $\boldsymbol{J} \approx \boldsymbol{U}_p\boldsymbol{\Lambda}_p\boldsymbol{V}_p^*$, that factorization can be used to approximately solve (5.2) [2].

5.2. Approximate truncated updates. Here we discuss three approximate LM updates based on a TSVD of $\boldsymbol{J}$. The work of [2] discussed advantages of using randomized 2-view or 1-view methods for speeding up the LM approach for reservoir models. After forming an approximate TSVD of $\boldsymbol{J}$, an update can be found according to [2, 41, 42, 43]

$$\delta\boldsymbol{x}_1 = -\boldsymbol{V}_p[\boldsymbol{\Lambda}_p^2 + (\mu+\gamma)\boldsymbol{I}_p]^{-1}(\boldsymbol{\Lambda}_p\boldsymbol{U}_p^*\boldsymbol{d} + \mu\boldsymbol{V}_p^*\boldsymbol{x}). \tag{5.3}$$

The study in [2] demonstrated that using randomized low-rank approximations can work well and can be considerably faster than using a standard iterative Lanczos method.

The LM method is better suited to the randomized paradigm than using the corresponding Gauss-Newton method ($\gamma = 0$), since the LM damping factor acts as a fail-safe mechanism. That is, when an LM update fails because of an inaccurate low-rank approximation, a new update is attempted using a larger γ. Increasing γ reduces contributions from the least accurately estimated singular vectors, which are associated with small singular values. A benefit of using (5.3) is that no extra views of $\boldsymbol{J}$ are needed for additional damping factors γ within an LM iteration, since the TSVD of $\boldsymbol{J}$ is independent of γ.

A second option, similar to (5.3), is to use [44]

$$\delta\boldsymbol{x}_2 = -\boldsymbol{V}_p[\boldsymbol{\Lambda}_p^2 + (\mu+\gamma)\boldsymbol{I}_p]^{-1}\boldsymbol{V}_p^*(\boldsymbol{J}^*\boldsymbol{d} + \mu\boldsymbol{x}). \tag{5.4}$$

Generating $\boldsymbol{J}^*\boldsymbol{d}$ does not add much cost since, with minor modifications, it can be evaluated along with the first pass for $\boldsymbol{J}^*$ within the randomized scheme. Equation (5.4) needs estimates for the principal right-singular vectors and singular values of $\boldsymbol{J}$. These can be found by approximating a TSVD of $\boldsymbol{J}$ or a TEVD of $\boldsymbol{J}^*\boldsymbol{J}$; see the discussion in section 4.2.

When using the exact TSVD, we have $\delta\boldsymbol{x}_1 = \delta\boldsymbol{x}_2$. Likewise, when using a randomized approximation according to $\boldsymbol{J} \approx [\![\boldsymbol{J}\boldsymbol{Q}_r]\!]_p\boldsymbol{Q}_r^* = \boldsymbol{U}_p\boldsymbol{\Lambda}_p\boldsymbol{V}_p^*$, with $\boldsymbol{V}_p = \boldsymbol{Q}_r\hat{\boldsymbol{V}}_p$, we have

$$\delta\boldsymbol{x}_2 = -\boldsymbol{V}_p[\boldsymbol{\Lambda}_p^2 + (\mu+\gamma)\boldsymbol{I}_p]^{-1}(\hat{\boldsymbol{V}}_p^*\boldsymbol{Q}_r^*\boldsymbol{J}^*\boldsymbol{d} + \mu\boldsymbol{V}_p^*\boldsymbol{x}) \tag{5.5}$$

$$= -\boldsymbol{V}_p[\boldsymbol{\Lambda}_p^2 + (\mu+\gamma)\boldsymbol{I}_p]^{-1}(\hat{\boldsymbol{V}}_p^*\hat{\boldsymbol{V}}_p\boldsymbol{\Lambda}_p\boldsymbol{U}_p^*\boldsymbol{d} + \mu\boldsymbol{V}_p^*\boldsymbol{x}) = \delta\boldsymbol{x}_1. \tag{5.6}$$

In this case it is not worth forming $\boldsymbol{J}^*\boldsymbol{d}$, and using $\delta\boldsymbol{x}_1$ should be preferred. However, when using $\boldsymbol{J} \approx \boldsymbol{Q}_c[\![\boldsymbol{Q}_c^*\boldsymbol{J}]\!]_p = \boldsymbol{Q}_c\hat{\boldsymbol{U}}_p\boldsymbol{\Lambda}_p\boldsymbol{V}_p^*$, we see that $\delta\boldsymbol{x}_1$ can differ from $\delta\boldsymbol{x}_2$. Finding $\delta\boldsymbol{x}_2$ with $\boldsymbol{J} \approx \boldsymbol{Q}_c[\![\boldsymbol{Q}_c^*\boldsymbol{J}]\!]_p$ uses the recommended way (when considering accuracy) of applying Algorithm 1 or Algorithm 2 to approximate $\boldsymbol{J}^*\boldsymbol{J}$; see section 4.2. Using $\delta\boldsymbol{x}_2$ with Algorithm 1 to generate $\boldsymbol{J} \approx [\![\boldsymbol{J}\boldsymbol{Q}_r]\!]_p\boldsymbol{Q}_r^*$ is like using $\delta\boldsymbol{x}_2$ with Algorithm 3 to approximate $\boldsymbol{J}^*\boldsymbol{J}$.

A third approximate solution to (5.2) can be found by using $\boldsymbol{J} \approx \hat{\boldsymbol{J}} = \boldsymbol{U}_p\boldsymbol{\Lambda}_p\boldsymbol{V}_p^*$ and

$$[\boldsymbol{J}^*\boldsymbol{J} + (\mu+\gamma)\boldsymbol{I}]^{-1} \approx \left[\hat{\boldsymbol{J}}^*\hat{\boldsymbol{J}} + (\mu+\gamma)\boldsymbol{I}\right]^{-1} = \frac{1}{\mu+\gamma}\left[\boldsymbol{I} - \boldsymbol{V}_p\boldsymbol{D}_p\boldsymbol{V}_p^*\right], \tag{5.7}$$

where $\boldsymbol{D}_p \in \mathbb{R}^{p\times p}$ is a diagonal matrix with $[\boldsymbol{D}_p]_{i,i} = \lambda_i^2/(\mu+\gamma+\lambda_i^2)$. Taking $\gamma = 0$, we have that (5.7) can be used to estimate the posterior parameter covariance matrix and can therefore be used to estimate parameter uncertainty [5, 35]. Using (5.7) for the LM update gives

$$\delta\boldsymbol{x}_3 = -\frac{1}{\mu+\gamma}\left[\boldsymbol{I} - \boldsymbol{V}_p\boldsymbol{D}_p\boldsymbol{V}_p^*\right](\boldsymbol{J}^*\boldsymbol{d} + \mu\boldsymbol{x}). \tag{5.8}$$

When using subspace iteration methods to form this approximation, it is worth keeping in mind and following the recommendations in sections 4.2.2 and 4.2.3, since the accuracy of (5.7) depends on the estimated right-singular vectors of $\boldsymbol{J}$ but not the left-singular vectors.

Equation (5.8) was proposed by [49] with $\boldsymbol{J}^*\boldsymbol{J}$ approximated using a subspace iteration procedure applied to $\boldsymbol{J}^*\boldsymbol{J}$. Similarly, [53] proposed this type of randomized approximation to solve linear equations arising in Gaussian process regression and classification in the machine learning context. The approximation error of (5.7) is $\mathscr{O}\left(\sum_{p+1}^{N_m}\frac{\lambda_i^2}{\mu+\gamma+\lambda_i^2}\right)$ [5, 35]. Therefore, a good approximation can be maintained when only truncating away singular values that are negligible compared to $\sqrt{\mu+\gamma}$. This matches our experience with using (5.8), where updates typically fail unless an LM damping factor is used such that $\mu+\gamma$ is similar to or greater than λ_{p+1}^2.

Section SM2 compares the properties of the three approximate LM solution methods presented here, based on an analysis given by [52]. A comparison is given for the length of the model updates, and some insight is given for the expected approximation error. The end of section SM2 also gives a correction for a proposition given by [52, Prop. 5.6]. As shown in section SM2, for a given low-rank approximation the lengths of $\delta\boldsymbol{x}_1$ and $\delta\boldsymbol{x}_2$ increase monotonically with the truncation p. Therefore, when using $\delta\boldsymbol{x}_1$ or $\delta\boldsymbol{x}_2$, the truncation point p can be used to regularize each model update. However, the third scheme $\delta\boldsymbol{x}_3$ does not have this property, since lowering p is expected to result in a longer model update $\delta\boldsymbol{x}_3$. Because of the regularizing properties of $\delta\boldsymbol{x}_1$ and $\delta\boldsymbol{x}_2$, we have found that those methods work well at early iterations. However, $\delta\boldsymbol{x}_3$ can work better at later iterations, since $\delta\boldsymbol{x}_3$ is in the full parameter space, unlike $\delta\boldsymbol{x}_1$ and $\delta\boldsymbol{x}_2$.

6. Single-pass methods for rectangular matrices.

The work of [2] also considered using a 1-view method to generate a TSVD and then solving the LM update equations approximately using (5.3). This approach speeds up each LM iteration, since in this case all the adjoint and direct equations can be solved in parallel. One-view methods are of broad interest for problems where a standard 2-view approach is considered too costly or where the information in the data matrix is only accessible a single time. Besides the inverse application discussed in [2], 1-view methods have been used for a storage optimal approach to solving convex low-rank matrix optimization [59]. Other possible useful applications may include on-the-fly processing and compression of large data sets, such as temporal weather data or outputs from numerical simulations [47]. The following subsections consider advancing state-of-the-art 1-view methods.

6.1. The baseline 1-view method.

Recently, Tropp et al. [46] presented a 1-view algorithm for fixed-rank approximation of rectangular matrices and compared it with other randomized 1-view methods. They compared their 1-view method [46, Alg. 7] with two similar methods based on [8, 54] and with an extended 1-view method [3, 48]. The approaches based on [8] and [54] performed the worst. They concluded that their 1-view algorithm [46, Alg. 7] is the preferred method for matrices with rapidly decaying spectra because they can be approximated accurately with a low-rank approximation. However, for small oversampling or matrices with flat spectra, the extended method performed the best for a given memory budget. However, the extended method [46] can be improved so that it performs as well as the 1-view method suggested by [46] for matrices with rapidly decaying spectra; see sections 6.6 and 8.4.

Here we consider the algorithm recommended by [46] as the baseline 1-view method. The following subsections look at possible improvements of the baseline algorithm. The baseline 1-view algorithm [46, Alg. 7] is as follows:

1. Given oversampling parameters l_1 and l_2 with $0 \leq l_1 \leq l_2$, form random sampling matrices $\boldsymbol{\Omega}_r \in \mathbb{R}^{n_c \times (p+l_1)}$ and $\boldsymbol{\Omega}_c \in \mathbb{R}^{n_r \times (p+l_2)}$ for the range and corange.
2. Find the range and corange sketches $\boldsymbol{Y}_c = \boldsymbol{A}\boldsymbol{\Omega}_r$ and $\boldsymbol{Y}_r = \boldsymbol{A}^*\boldsymbol{\Omega}_c$.
3. Find an orthonormal basis for the range $\boldsymbol{Q}_c = \operatorname{orth}(\boldsymbol{Y}_c)$.
4. Find the least-squares solution $\boldsymbol{X}$ to $(\boldsymbol{\Omega}_c^*\boldsymbol{Q}_c)\boldsymbol{X} = \boldsymbol{Y}_r^*$, i.e., $\boldsymbol{X} = (\boldsymbol{\Omega}_c^*\boldsymbol{Q}_c)^\dagger \boldsymbol{Y}_r^*$.
5. Then $\boldsymbol{A} \approx \boldsymbol{Q}_c[\![\boldsymbol{X}]\!]_p$.

The above 1-view method is based on the same elements as the basic 2-view method. The main difference is that in the 1-view method the corange sketch $\boldsymbol{Y}_r$ is constructed independently of the range sketch $\boldsymbol{Y}_c$. Therefore, the range and corange sketches, which are used to form the low-rank approximation, can be found using one pass of the data matrix $\boldsymbol{A}$. However, the convenience of constructing an approximation with only one view reduces accuracy. Like Tropp et al. [46], we generally recommend more accurate methods, such as Algorithm 1, for applications that can afford more than one view.

6.2. Baseline oversampling.

In practice we need reliable oversampling schemes. For a sketch budget $T = 2p + l_1 + l_2$, Tropp et al. [46] suggested three oversampling schemes based on empirical evidence and theoretical bounds for the accuracy of their 1-view approach. They presented oversampling schemes to suit both real valued matrices and complex matrices. However, we assume that we are dealing with a real matrix and $T \geq 2p+6$.

First, for a flat singular spectrum, [46] recommended using

$$l_1 = \max\left\{2, \left\lfloor (T-1)\frac{\sqrt{p(T-p-2)(1-2/(T-1))}-(p-1)}{T-2p-1} \right\rfloor - p\right\} \text{ and } l_2 = T-2p-l_1. \tag{6.1}$$

Second, for a moderately decaying spectrum, [46] recommended using

$$l_1 = \max\{2, \lfloor (T-1)/3 \rfloor - p\} \quad \text{and} \quad l_2 = T-2p-l_1. \tag{6.2}$$

Third, for the case of a rapidly decaying spectrum, [46] recommended choosing

$$l_1 = \lfloor (T-2)/2 \rfloor - p \quad \text{and} \quad l_2 = T-2p-l_1. \tag{6.3}$$

When using the baseline 1-view method, it is generally not advisable to choose l_1 too close to l_2 ($l_1 \approx l_2$), e.g., by following (6.3). Using $l_1 \approx l_2$, and especially $l_1 = l_2$, is a bad idea, unless the spectral decay is rapid. For peak performance, some prior knowledge about the matrix is needed to determine which of the above oversampling schemes should be chosen. This can be the case for certain applications. However, when reliable prior ideas are unavailable, then (6.2) is the all-around favored choice [46].

6.3. Disadvantages of the baseline method. It would be beneficial to remedy the drawbacks of needing prior insight to achieve peak performance and being unable to use $l_1 \approx l_2$ without risking loss of accuracy. Furthermore, the versatile oversampling scheme (6.2) is suboptimal for matrices with rapidly decaying spectra. For the best performance of the 1-view method, we should choose l_1 close to l_2 in cases where $\boldsymbol{A}$ has sharp spectral decay, but we need an escape mechanism for cases where the spectral decay is not sharp. Another reason to fix the drawback of choosing $l_1 = l_2$ is that for some applications, l_2 can be a computational bottleneck. In that case it is tempting to set $l_1 = l_2$ to try to get as much information as possible given the computational constraints. The following subsections discuss modifications to the baseline method to enable good performance for a range of problems with the choice $l_1 = l_2$.

6.4. A more flexible 1-view method. Choosing $l_1 \approx l_2$ commonly results in solving a badly posed or badly conditioned least-squares problem $(\boldsymbol{\Omega}_c^* \boldsymbol{Q}_c)\boldsymbol{X} = \boldsymbol{Y}_r^*$, and as a result the approximation can suffer. The coefficient matrix $\boldsymbol{\Omega}_c^* \boldsymbol{Q}_c$ has $p+l_2$ rows and $p+l_1$ columns. Therefore, a way to avoid ill-conditioning is to sample more for the corange ($l_2 > l_1$) to give an overdetermined problem. Another option is to select $\boldsymbol{Q}_c$ based on a TSVD of $\boldsymbol{Y}_c$ [55, 20, 25]. Instead of choosing $\boldsymbol{Q}_c$ as an orthonormal basis for the full range of $\boldsymbol{Y}_c$, we can choose $[\boldsymbol{Q}_c, \sim, \sim] = \text{tsvd}(\boldsymbol{Y}_c, p+l_c)$, where $0 \leq l_c \leq l_1$. That is, $\boldsymbol{Q}_c \in \mathbb{R}^{n_r \times (p+l_c)}$ contains the $p+l_c$ leading left-singular vectors of $\boldsymbol{Y}_c$. If l_c is smaller than l_1, then using $(\boldsymbol{\Omega}_c^* \boldsymbol{Q}_c) \in \mathbb{R}^{(p+l_2)\times(p+l_c)}$ can be expected to give a better conditioned problem, even for $l_1 = l_2$. Finding the left-singular vectors of $\boldsymbol{Y}_c$ instead of just finding an orthonormal basis does not add much to the cost—especially considering that applying $\boldsymbol{A}$ typically contributes the most to the cost and time for applications using the 1-view approach.

Incorporating this modification into the baseline 1-view method gives Algorithm 4. Choosing $l_c = l_1$ gives the baseline algorithm as proposed by [46]. In Algorithm 4 we use

Algorithm 4. Randomized 1-view method for TSVD: Tropp variant.

```
INPUT: Matrix $\boldsymbol{A}\in\mathbb{R}^{n_r\times n_c}$, integers $p>0$ and $l_2\geq l_1\geq l_{\mathrm{c}}\geq 0$.
RETURNS: Approximate rank-$p$ SVD, $\boldsymbol{U}_p\boldsymbol{\Lambda}_p\boldsymbol{V}_p^*$, of $\boldsymbol{A}$.
1: (a) $\boldsymbol{\Omega}_r=\mathrm{randn}(n_c,p+l_1)$ and (b) $\boldsymbol{\Omega}_c=\mathrm{randn}(n_r,p+l_2)$.
2: (a) $\boldsymbol{Y}_c=\boldsymbol{A}\boldsymbol{\Omega}_r$ and (b) $\boldsymbol{Y}_r=\boldsymbol{A}^*\boldsymbol{\Omega}_c$.
3: if $l_{\mathrm{c}}<l_1$ then $[\boldsymbol{Q}_c,\sim,\sim]=\mathrm{tsvd}(\boldsymbol{Y}_c,p+l_{\mathrm{c}})$; else $[\boldsymbol{Q}_c,\sim]=\mathrm{qr}(\boldsymbol{Y}_c)$.
4: $[\hat{\boldsymbol{Q}},\hat{\boldsymbol{R}}]=\mathrm{qr}(\boldsymbol{\Omega}_c^*\boldsymbol{Q}_c)$.
5: $\boldsymbol{X}=\hat{\boldsymbol{R}}^{-1}\hat{\boldsymbol{Q}}^*\boldsymbol{Y}_r^*$.
6: $[\hat{\boldsymbol{U}}_p,\boldsymbol{\Lambda}_p,\boldsymbol{V}_p]=\mathrm{tsvd}(\boldsymbol{X},p)$.
7: $\boldsymbol{U}_p=\boldsymbol{Q}_c\hat{\boldsymbol{U}}_p$.
```

(**a**) and (**b**) to denote and group together steps that can be performed independently in parallel. Using l_{c} and $l_1=l_2$ can work just as well as, and in some cases better than, the schemes suggested by [46]; see section 8.4. For the test matrices we have considered, we have found that using $l_1=l_2$ and $l_{\mathrm{c}}\approx l_1/2$ is a good all-round parameter choice and is similar to using (6.2) and $l_{\mathrm{c}}=l_1$.

Similar algorithms were presented by [55, 20, 25]; however, in the notation used here the methods presented by [55, 46, 25] used $l_{\mathrm{c}}=0$ and $l_1=l_2$. The algorithms presented here are more general in terms of the oversampling parameters and although the focus here is on choosing $l_1=l_2$, the presented algorithms can use oversampling parameters fulfilling $0\leq l_{\mathrm{c}}\leq l_1\leq l_2$. The algorithm proposed by [55, sect. 5.2] finds a low-rank approximation by solving a similar least-squares problem,

$$(\boldsymbol{\Omega}_c^*\boldsymbol{Q}_c)\hat{\boldsymbol{X}}=\boldsymbol{Y}_r^*\boldsymbol{Q}_r, \tag{6.4}$$

where $\boldsymbol{Q}_c$ and $\boldsymbol{Q}_r$ are formed, respectively, by the leading left-singular vectors of $\boldsymbol{Y}_c$ and $\boldsymbol{Y}_r$. Then the low-rank approximation is taken as $\boldsymbol{A}\approx\boldsymbol{Q}_c[\![\hat{\boldsymbol{X}}]\!]_p\boldsymbol{Q}_r^*$. Algorithm 4 can be modified easily to use (6.4). Dealing with the additional matrix $\boldsymbol{Q}_r$, which does not appear in Algorithm 4, does not necessarily increase computational time since it can be dealt with in parallel with other core tasks. For the matrices we considered, Algorithm 4 and (6.4) performed similarly in terms of accuracy. For the tests in section 8.4, we present results using Algorithm 4 but omit showing results for (6.4) as they are comparable.

Following [55], an approximation was proposed in [20, 25] which can also be written as $\boldsymbol{A}\approx\boldsymbol{Q}_c[\![\hat{\boldsymbol{X}}]\!]_p\boldsymbol{Q}_r^*$. The distinction is that [20, 25] suggested finding $\hat{\boldsymbol{X}}$ as the least-squares solution to (6.4) and a related equation $(\boldsymbol{\Omega}_r^*\boldsymbol{Q}_r)\hat{\boldsymbol{X}}^*=\boldsymbol{Y}_c^*\boldsymbol{Q}_c$. We are not aware of any research using the variant proposed by [20, 25]. However, for the matrices tested here, the variant in [20, 25] mostly gave accuracy similar to the simpler algorithms presented above. Considering that result, and the additional computational complexity and expense of dealing with the additional least-squares equations, we do not further consider the variant in [20, 25].

6.5. Oversampling when using l_{c}.

6.5.1. Basic oversampling scheme. Like the baseline algorithm, Algorithm 4 requires practical ways of choosing l_1, l_2, and l_{c}. The oversampling parameter l_{c} can be chosen as

$$l_{\mathrm{c}}=\lfloor\alpha l_1\rfloor,\quad\text{with}\quad 0\leq\alpha\leq 1. \tag{6.5}$$

Here we are concerned with schemes when $l_1 \simeq l_2$ or more precisely $l_1 = \lfloor (l_1 + l_2)/2 \rfloor = \lfloor T/2 \rfloor - p$. In that case, l_c functions similarly in relation to l_1 as l_1 functions in relation to l_2 when using the baseline 1-view method. That is, choosing l_c close to l_1 works well when the spectrum decays rapidly. Choosing $l_1 \simeq l_2$ and $\alpha = 1/2$ can work quite well for various matrices in a similar way to using (6.2) and $l_\mathrm{c} = l_1$.

6.5.2. A minimum variance approach. After forming the sketches $\boldsymbol{Y}_c$ and $\boldsymbol{Y}_r$ we can try to apply some postprocessing to determine a good l_c parameter given the information in the sketches and sampling matrices. An approach we have found to work well looks at how the spectrum of $\boldsymbol{X}$ varies for all possible values of l_c given $\boldsymbol{Y}_c$ and $\boldsymbol{Y}_r$. The approach is based on the observation that the singular values of $\boldsymbol{X}$ tend to have large variance when choosing l_c close to l_1 for $l_1 \simeq l_2$. Furthermore, the spectrum of $\boldsymbol{X}$ can also exhibit large variance when choosing l_c close to 0 and often results in inferior approximations. We therefore consider finding l_c that locally gives a matrix $\boldsymbol{X}(l_\mathrm{c})$ with a singular spectrum that has minimum variance in some sense.

Defining the vector of locally normalized singular values $\boldsymbol{s}(l_\mathrm{c})$, we have for $0 < l_\mathrm{c} < l_1$

$$\boldsymbol{s}(l_\mathrm{c}) := \left[\frac{\lambda_1(l_\mathrm{c}-1)}{\lambda_1(l_\mathrm{c})}, \dots, \frac{\lambda_p(l_\mathrm{c}-1)}{\lambda_p(l_\mathrm{c})}, \frac{\lambda_1(l_\mathrm{c})}{\lambda_1(l_\mathrm{c})}, \dots, \frac{\lambda_p(l_\mathrm{c})}{\lambda_p(l_\mathrm{c})}, \frac{\lambda_1(l_\mathrm{c}+1)}{\lambda_1(l_\mathrm{c})}, \dots, \frac{\lambda_p(l_\mathrm{c}+1)}{\lambda_p(l_\mathrm{c})} \right], \tag{6.6}$$

and for $l_\mathrm{c} = 0$

$$\boldsymbol{s}(l_\mathrm{c}) := \left[\frac{\lambda_1(l_\mathrm{c})}{\lambda_1(l_\mathrm{c})}, \dots, \frac{\lambda_p(l_\mathrm{c})}{\lambda_p(l_\mathrm{c})}, \frac{\lambda_1(l_\mathrm{c}+1)}{\lambda_1(l_\mathrm{c})}, \dots, \frac{\lambda_p(l_\mathrm{c}+1)}{\lambda_p(l_\mathrm{c})} \right]; \tag{6.7}$$

the *minimum variance* l_c is found according to

$$l_\mathrm{c}^{\mathrm{MinVar}} := \underset{l_\mathrm{c}}{\operatorname{argmin}} \operatorname{Var}[\boldsymbol{s}(l_\mathrm{c})], \quad \text{with} \quad 0 \le l_\mathrm{c} < l_1. \tag{6.8}$$

Note that this postprocessing does not require re-evaluating the sketches. We just need to generate the left-singular vectors of $\boldsymbol{Y}_c$ and truncate them to $p + l_\mathrm{c}$ for each trial l_c. For efficiency we evaluate $\boldsymbol{\Omega}_c^* \boldsymbol{Q}_c$ for $l_\mathrm{c} = l_1$, where in this context $\boldsymbol{Q}_c$ denotes the left-singular vectors of $\boldsymbol{Y}_c$. Then for each trial l_c we only need the first $p + l_\mathrm{c}$ columns of $\boldsymbol{\Omega}_c^* \boldsymbol{Q}_c$.

The additional computational operations are $\mathscr{O}(l_1 [p + l_1]^2 n_r)$ when using Algorithm 4. To improve efficiency it is possible to instead apply a variant using (6.4) at a cost of $\mathscr{O}([p + l_1]^2 n_r + l_1 [p + l_1]^3)$. The $\mathscr{O}([p + l_1]^2 n_r)$ term comes from evaluating $\boldsymbol{Y}_r^* \boldsymbol{Q}_r$ once, where $\boldsymbol{Q}_r$ is an orthonormal basis for $\boldsymbol{Y}_r$. The remainder of the cost comes from solving the smaller least-squares problem $l_1 + 1$ times. This additional postprocessing cost, for finding a better l_c, is potentially worth it for applications using the 1-view approach, where the dominant cost comes from forming the sketches. Furthermore, the computations for each trial l_c can be carried out independently in parallel to the other trial variables.

Choosing l_c by (6.8) worked well for the problems we considered, and this minimum variance approach can perform quite closely to the peak performance for a range of matrices; see section 8.4. Nevertheless, the basic postprocessing approach presented here to find a good l_c parameter can be improved. For instance, when the rank p is small (e.g., $p = 1$), the vector $\boldsymbol{s}(l_\mathrm{c})$ presents a small sample size for estimating a variance. We have noticed that the minimum variance approach is less reliable for $p = 1$. In that case it may be better to temporarily use a larger rank as input into the 1-view algorithm of choice and truncate afterwards, although this was not done for the test cases reported here.

6.6. Extended sketching scheme. In [46], an extended 1-view variant based on [3, 48] had the best performance for small memory budgets and matrices with flat spectra. However, for greater memory budgets and rapidly decaying spectra, [46] found the baseline 1-view method to outperform the extended approach.

For the extended approach, we assume that $l_1{=}l_2$ and introduce additional subsampled randomized Fourier transform (SRFT) matrices $\boldsymbol{\Phi}_c{\in}\mathbb{R}^{n_c\times s}$ and $\boldsymbol{\Phi}_r{\in}\mathbb{R}^{n_r\times s}$, where $s{\geq}p{+}l_1$. Here, for $i{\in}\{c,r\}$, $\boldsymbol{\Phi}_i{=}\boldsymbol{D}_i\boldsymbol{F}_i\boldsymbol{P}_i$ is a random SRFT matrix, where $\boldsymbol{D}_i{\in}\mathbb{R}^{n_i\times n_i}$ is a diagonal matrix with elements drawn independently from a Rademacher distribution (diagonal entries are ± 1), $\boldsymbol{F}_i{\in}\mathbb{R}^{n_i\times n_i}$ is a discrete cosine transform matrix when $\boldsymbol{A}$ only has real entries or is a discrete Fourier transform matrix when $\boldsymbol{A}$ has complex entries, and $\boldsymbol{P}_i{\in}\mathbb{R}^{n_i\times s}$ has columns sampled from the $n_i\times n_i$ identity matrix without replacement. Advantages of an SRFT are its low storage cost (needing only $n_i{+}s$ entries for $\boldsymbol{D}_i$ and $\boldsymbol{P}_i$) and that for a dense $\boldsymbol{A}$, $\boldsymbol{A\Phi}_c$ can be evaluated using $\mathscr{O}(n_r n_c \log s)$ operations [20, 23, 55] instead of the usual $\mathscr{O}(n_r n_c s)$ operations. However, we should note that we do not consider the extended approach discussed here for the Jacobian matrices motivating the present study. The reason is that, when using direct and adjoint methods to find $\boldsymbol{A}$ or its transpose times a matrix $\boldsymbol{\Phi}_i$, the cost scales with the number of columns s. The SRFT matrices need more columns than the Gaussian random matrices, and using SRFT matrices is therefore not advantageous in the context of adjoint and direct methods.

With the extended approach we again find $\boldsymbol{Y}_c{=}\boldsymbol{A\Omega}_r$ and $\boldsymbol{Y}_r{=}\boldsymbol{A}^*\boldsymbol{\Omega}_c$, but we also find an extended sketch $\boldsymbol{Z}{=}\boldsymbol{\Phi}_r^*\boldsymbol{A\Phi}_c$. Subsequently, we evaluate the following QR-factorizations:

$$\boldsymbol{Q}_c\boldsymbol{R}_c{:=}\boldsymbol{Y}_c; \quad \boldsymbol{Q}_r\boldsymbol{R}_r{:=}\boldsymbol{Y}_r; \quad \boldsymbol{U}_c\boldsymbol{T}_c{:=}\boldsymbol{\Phi}_r^*\boldsymbol{Q}_c; \quad \boldsymbol{U}_r\boldsymbol{T}_r{:=}\boldsymbol{\Phi}_c^*\boldsymbol{Q}_r. \tag{6.9}$$

Finally, an approximation can be found for the extended approach according to [46],

$$\hat{\boldsymbol{A}}_{\text{bwz}}{:=}\boldsymbol{Q}_c\boldsymbol{T}_c^\dagger[\![\boldsymbol{U}_c^*\boldsymbol{Z}\boldsymbol{U}_r]\!]_p(\boldsymbol{T}_r^*)^\dagger\boldsymbol{Q}_r^*. \tag{6.10}$$

However, we have noticed that a more accurate approximation can be found by using

$$\hat{\boldsymbol{A}}_{\text{bwz2}}{:=}\boldsymbol{Q}_c\left[\!\left[\boldsymbol{T}_c^\dagger\boldsymbol{U}_c^*\boldsymbol{Z}\boldsymbol{U}_r(\boldsymbol{T}_r^*)^\dagger\right]\!\right]_p\boldsymbol{Q}_r^*. \tag{6.11}$$

Defining the QR-factorization $\hat{\boldsymbol{Q}}\hat{\boldsymbol{R}}{:=}\boldsymbol{\Omega}_c^*\boldsymbol{Q}_c$ used in Algorithm 4, we see that the difference between (6.10) and (6.11) is analogous to the difference between using $\hat{\boldsymbol{A}}_{\text{woo}}{:=}\boldsymbol{Q}_c\hat{\boldsymbol{R}}^\dagger[\![\hat{\boldsymbol{Q}}^*\boldsymbol{Y}_r^*]\!]_p$ and the baseline 1-view method $\hat{\boldsymbol{A}}_{\text{tropp}}{:=}\boldsymbol{Q}_c[\![\hat{\boldsymbol{R}}^\dagger\hat{\boldsymbol{Q}}^*\boldsymbol{Y}_r^*]\!]_p$. The approximation $\hat{\boldsymbol{A}}_{\text{woo}}$ is an adaptation made by [46] of an approach given by [54]. As previously mentioned, [46] found that the $\hat{\boldsymbol{A}}_{\text{woo}}$ variant did not perform nearly as well as the baseline $\hat{\boldsymbol{A}}_{\text{tropp}}$.

From [46] it is unclear[1] how to choose the oversampling factors for the above extended approach. Storing the sampling matrices and sketches requires $(2p{+}l_1{+}l_2{+}1)(n_r{+}n_c){+}s(s{+}2)$ numbers [46]. Tropp et al. [46] compared the extended approach to the baseline method by considering oversampling parameters such that $(2p{+}l_1{+}l_2{+}1)(n_r{+}n_c){+}s(s{+}2)$ approximately equals the memory allocated to the simpler 1-view methods $T(n_r{+}n_c)$. We do the same when comparing the extended approach with Algorithm 4. For the matrices tested here, we found that using (6.11) with

$$l_1{=}l_2{=}0.8\lfloor T/2{-}p\rfloor \tag{6.12}$$

[1]Very recently, [47] presented a theoretical analysis of the extended 1-view method proposed here and gave theoretical guidance on how to choose oversampling parameters for the extended method.

and choosing s as the largest value fulfilling $(2p+l_1+l_2+1)(n_r+n_c)+s(s+2)\approx T(n_r+n_c)$ resulted in errors close to the best performance. In terms of memory, this approach performed the best out of the 1-view schemes mentioned above; see section 8.4.

6.7. Matrices streamed row wise. For the special case where the elements of a large matrix are read into random-access memory a few rows at a time, it is possible to attain the accuracy of a 2-view method while only accessing the elements of the matrix once [57, 58]. Looping over all the rows in $\boldsymbol{A}$ only once, the matrices $\boldsymbol{Y}_c=\boldsymbol{A}\boldsymbol{\Omega}_r$ and $\hat{\boldsymbol{Y}}_r=\boldsymbol{A}^*\boldsymbol{Y}_c$ can be found according to $\boldsymbol{Y}_c[i,:]=\boldsymbol{A}[i,:]\boldsymbol{\Omega}_r$ and $\hat{\boldsymbol{Y}}_r=\boldsymbol{A}^*\boldsymbol{Y}_c=\sum_i(\boldsymbol{A}[i,:])^*\boldsymbol{Y}_c[i,:]$, where $\boldsymbol{A}[i,:]$ denotes the ith row of $\boldsymbol{A}$. Finding $[\boldsymbol{Q}_c,\boldsymbol{R}_c]=\text{qr}(\boldsymbol{Y}_c)$ and the least-squares solution $\boldsymbol{B}$ to $\boldsymbol{B}^*\boldsymbol{R}_c=\hat{\boldsymbol{Y}}_r$, we then have $\boldsymbol{Q}_c\boldsymbol{B}\approx\boldsymbol{A}$. In [57], this QB-factorization was found by an iterative blocked algorithm instead of by forming $\boldsymbol{Q}_c\boldsymbol{R}_c:=\boldsymbol{Y}_c$ and solving for $\boldsymbol{B}$ directly. If $\boldsymbol{R}_c$ is invertible, then $\boldsymbol{B}^*=\hat{\boldsymbol{Y}}_r\boldsymbol{R}_c^{-1}=\boldsymbol{A}^*\boldsymbol{Q}_c$ and we get the basic randomized 2-view approximation in a single pass. However, it is worth noting that the least-squares problem $\boldsymbol{B}^*\boldsymbol{R}_c=\hat{\boldsymbol{Y}}_r$ can be badly conditioned. The conditioning issue can, for instance, be dealt with by finding a TSVD of $\boldsymbol{R}_c$ and estimating a solution using a pseudoinverse.

In [58], the above single-pass ideas were extended to the power iteration approach to obtain higher quality approximations of large matrices accessed row wise. The following section discusses generalizing a pass-efficient randomized block Krylov algorithm and combining randomized block Krylov methods with the ideas of [57, 58].

7. Pass-efficient randomized block Krylov methods.

As mentioned previously, if the 1-view approach is not suitably accurate, then more accurate methods, such as Algorithm 1, should be considered. While Algorithm 1 is more pass efficient than classical iterative methods, which involve matrix-vector multiplication, its pass efficiency can be improved by using a randomized block Krylov approach [11, 19, 29, 32, 37]. The next subsection gives a brief overview of the current state of the art for randomized block Krylov methods. The subsections that follow outline two extensions of the block Krylov approach. The first approach, like Algorithm 1, enables the user to specify any budget of views $v\geq 2$. The second approach incorporates the ideas discussed in section 6.7 to give a more pass-efficient method for large matrices stored in row-major format.

7.1. A prototype block Krylov method. Following [11, 19, 29, 32, 37], an approximate basis $\boldsymbol{Q}_c$ for the range of $\boldsymbol{A}$ can be found as the basis of a Krylov subspace given by

$$\boldsymbol{K}_c=\text{range}(\boldsymbol{A}\boldsymbol{\Omega}_r\ [\boldsymbol{A}\boldsymbol{A}^*]\boldsymbol{A}\boldsymbol{\Omega}_r\ \cdots\ [\boldsymbol{A}\boldsymbol{A}^*]^q\boldsymbol{A}\boldsymbol{\Omega}_r), \tag{7.1}$$

where $\boldsymbol{\Omega}_r\in\mathbb{R}^{n_c\times(p+l)}$ is a random sampling matrix. Possible improvements over the simpler subspace iteration method come from using a larger and more complete basis $\boldsymbol{Q}_c$. For numerical stability it is best to evaluate $[\boldsymbol{A}\boldsymbol{A}^*]^j\boldsymbol{A}\boldsymbol{\Omega}_r$ using subspace iteration [29, 32].

7.2. A more general block Krylov method for $v\geq 2$. Like the standard subspace iteration method, the above prototype block Krylov method uses an even $2(q+1)$ views. Again, we can generalize the above prototype approach to work for any views $v\geq 2$. For odd v, we simply form a basis $\boldsymbol{Q}_r$ for the corange using a Krylov subspace defined as

$$\boldsymbol{K}_r=\text{range}([\boldsymbol{A}^*\boldsymbol{A}]\boldsymbol{\Omega}_r\ [\boldsymbol{A}^*\boldsymbol{A}]^2\boldsymbol{\Omega}_r\ \cdots\ [\boldsymbol{A}^*\boldsymbol{A}]^{(v-1)/2}\boldsymbol{\Omega}_r). \tag{7.2}$$

Algorithm 5. Randomized SVD using generalized block Krylov method.

```
INPUT: Matrix $A \in \mathbb{R}^{n_r \times n_c}$, integers $p>0$, $l \geq 0$, and $v \geq 2$.
RETURNS: Approximate rank-$p$ SVD, $U_p \Lambda_p V_p^*$, of $A$.
1: if $v$ is even, then
2:     Generate $K_c$ by (7.1) using subspace iteration with $v-1$ views ($q=[v-2]/2$).
3:     $[Q_c, \sim]=\text{qr}(K_c)$.
4:     $[V_p, \Lambda_p, \hat{U}_p]=\text{tsvd}(A^* Q_c, p)$. Then find $U_p = Q_c \hat{U}_p$.
5: else
6:     Generate $K_r$ by (7.2) using subspace iteration with $v-1$ views.
7:     $[Q_r, \sim]=\text{qr}(K_r)$.
8:     $[U_p, \Lambda_p, \hat{V}_p]=\text{tsvd}(A Q_r, p)$. Then find $V_p = Q_r \hat{V}_p$.
```

Combining the above, we get Algorithm 5, which is a more flexible block Krylov method than that used by [11, 19, 29, 32, 37] and allows $v \geq 2$. For even v, Algorithm 5 uses $A \approx Q_c Q_c^* A$; otherwise, it uses $A \approx A Q_r Q_r^*$. When the goal is to approximate the left-singular vectors of a matrix J, the same rule of thumb regarding accuracy applies to Algorithm 5 and Algorithm 1: apply Algorithm 5 to J^* for even v but to J for odd v. This is the opposite of how the block Krylov approach was used by [11, 32].

Notice that Algorithm 5 is the same as Algorithm 1 for $v<4$. The main cost of Algorithm 5 is multiplying A with $(v+\lfloor v/2 \rfloor -1)(p+l)$ vectors, and the cost of the QR factorizations is $\mathscr{O}(v^2 n_c [p+l]^2)$, assuming $n_r = n_c$. This is a higher cost for a given v than that in Algorithm 1, which requires multiplying A with $v(p+l)$ vectors, and the cost of the QR-factorizations is $\mathscr{O}(v n_c [p+l]^2)$. However, Algorithm 5 may require fewer views than Algorithm 1 to achieve a desired accuracy when Algorithm 1 needs $v>4$.

For $l=0$ and even v, Musco and Musco [32] showed that Algorithm 5 requires $\mathscr{O}(1/\sqrt{\epsilon})$ views to achieve $\|A - Q_c Q_c^* A\| \leq (1+\epsilon) \|A - [\![A]\!]_p\|$; however, Algorithm 1 requires $\mathscr{O}(1/\epsilon)$ views [32]. Because their analysis assumed no oversampling, it is unclear how the methods compare in practice when oversampling is applied. By oversampling more, the accuracy of Algorithm 1 can be increased, and l can be chosen such that the cost of Algorithm 1 is similar to the cost of Algorithm 5. Which approach should be chosen in practice? The results in section 8.3.3 suggest that Algorithm 1 with more sampling can be competitive with Algorithm 5 when the matrix has reasonably rapid spectral decay. Algorithm 5, however, is more advantageous for problems where the tail of the singular spectrum is flat. This can be the case when dealing with noisy data matrices, which was the problem motivating [32].

7.3. A block Krylov method for matrices stored row wise. The ideas in section 6.7 can be applied in the block Krylov framework to improve pass efficiency when dealing with large matrices accessed row by row. In that case, $A^* A$ times a matrix can be evaluated in a single pass. Therefore, the Krylov subspace for the corange of A (7.2) can be evaluated using half the number of views needed by Algorithm 5. Using this approach could be advantageous when dealing with large data matrices stored out-of-core.

8. Experimental results.

8.1. Test matrices. To test the methods discussed in this study we consider seven test matrices. The first is a 6,135×24,000 Jacobian matrix S_D arising in a synthetic geothermal inverse problem like the one examined in [2]. The leading,

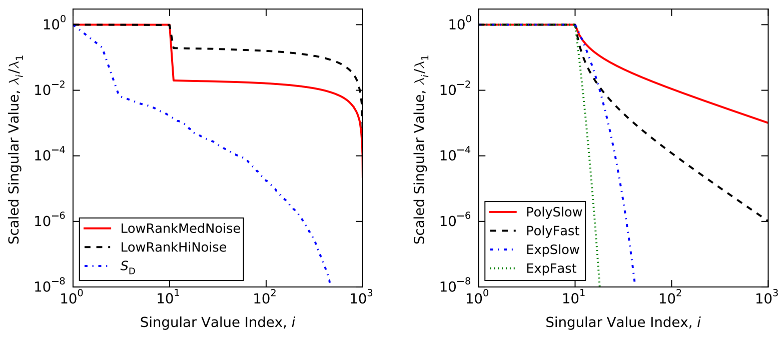

FIG. 1. *Scaled singular values for the test matrices.*

normalized singular values of $\boldsymbol{S}_{\mathrm{D}}$ are depicted in Figure 1. The other test matrices are the same as some of those used by [46]: two matrices that have a singular spectrum with a flat tail, two where the trailing singular values decay polynomially, and two where the decay of the tail is exponential.

Defining $R{=}10$ and $n{=}1{,}000$, each of the flat matrices is a realization of a matrix which combines a rank-R matrix and a random noise matrix. That is, these flat or low-rank plus noise matrices are $n \times n$ matrices given by

$$\text{(8.1)} \qquad \operatorname{diag}(\underbrace{1,\ldots,1}_{\mathrm{R}},0,\ldots,0)+\sqrt{\frac{\eta R}{2n^2}}(\boldsymbol{G}+\boldsymbol{G}^*), \quad \text{where} \quad \boldsymbol{G}{=}\operatorname{randn}(n,n).$$

We use a medium noise matrix (`LowRankMedNoise`) with $\eta{=}10^{-2}$ and a high noise matrix (`LowRankHiNoise`) with $\eta{=}1$. The polynomial decay matrices are diagonal matrices of the following type:

$$\text{(8.2)} \qquad \operatorname{diag}(\underbrace{1,\ldots,1}_{\mathrm{R}},2^{-\rho},3^{-\rho},\ldots,[n-R+1]^{-\rho}).$$

The first polynomially decaying matrix has a slow decay $\rho{=}1$ (`PolySlow`); the other decays faster with $\rho{=}2$ (`PolyFast`). Similarly, the exponentially decaying matrices are

$$\text{(8.3)} \qquad \operatorname{diag}(\underbrace{1,\ldots,1}_{\mathrm{R}},10^{-\theta},10^{-2\theta},\ldots,10^{-(n-R)\theta}).$$

We use one matrix with a rapid decay $\theta{=}1$ (`ExpFast`) and another with a slower decay $\theta{=}0.25$ (`ExpSlow`). The singular spectra of the above test matrices are shown in Figure 1.

8.2. Quantifying algorithmic accuracy. To quantify the accuracy of the algorithms, the generated rank-p approximations $\hat{\boldsymbol{A}}_{\mathrm{out}}$ are compared with the optimal rank-p factorization of the matrix of interest $\boldsymbol{A}$. We compare the relative Frobenius norm and relative spectral norm errors, defined as

$$\text{(8.4)} \qquad \text{relative Frobenius error} := \frac{\left\|\boldsymbol{A}-\hat{\boldsymbol{A}}_{\mathrm{out}}\right\|_{\mathrm{F}}}{\|\boldsymbol{A}-[\![\boldsymbol{A}]\!]_p\|_{\mathrm{F}}}-1;$$

$$\text{(8.5)} \qquad \text{relative spectral error} := \frac{\left\|\boldsymbol{A}-\hat{\boldsymbol{A}}_{\mathrm{out}}\right\|}{\|\boldsymbol{A}-[\![\boldsymbol{A}]\!]_p\|}-1.$$

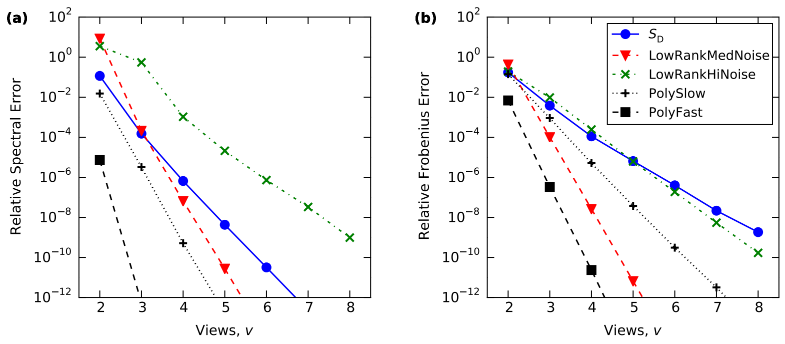

FIG. 2. *Errors when using the generalized subspace iteration method (Algorithm* 1*), with a target rank* $p=10$ *and oversampling parameter* $l=10$. (a) *Relative spectral norm error* (8.5). (b) *Relative Frobenius norm error* (8.4).

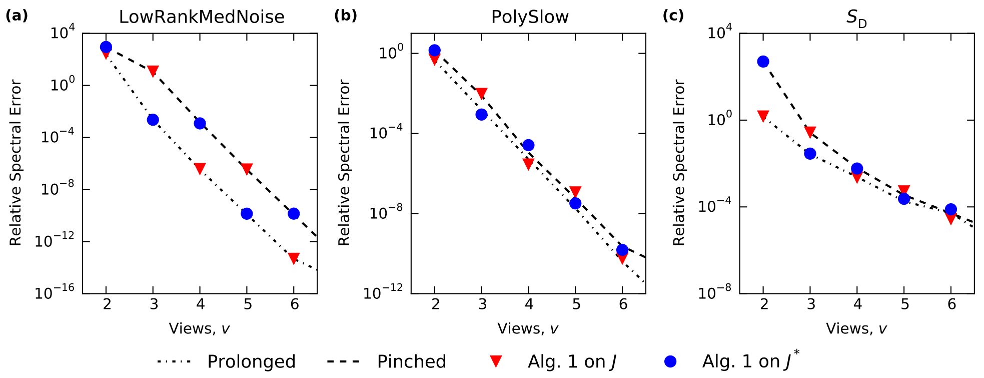

FIG. 3. *For selected test matrices* $\boldsymbol{J}$*, shown are spectral norm approximation errors* (8.5) *of the normal matrix* $\boldsymbol{J}^*\boldsymbol{J}$ *when using the prolonged (Nyström) sketch Algorithm* 2*, the pinched sketch Algorithm* 3*,* $\boldsymbol{J}$ *as input into Algorithm* 1*, or* $\boldsymbol{J}^*$ *as input into Algorithm* 1*. For all plots the target rank was* $p=10$ *and* $l=5$.

8.3. Effectiveness of the generalized subspace iteration method.

8.3.1. Standard application. Figure 2 shows the decay of the spectral and Frobenius norm errors as functions of views when using the generalized subspace iteration method (Algorithm 1). To generate Figure 2 we considered a target rank $p=10$ and $l=10$. The errors in Figure 2 are averages over 50 calls of Algorithm 1. As we would expect, based on Theorems 3.1 and 4.1, the relative errors decay exponentially with v. The advantage of using Algorithm 1 over the standard subspace approach (which is the same as Algorithm 1 for even views) is apparent from Figure 2: an odd number of views can suffice to achieve a desired accuracy. In that case, an additional view is excessive.

8.3.2. Normal matrices. Here we consider using Algorithm 1 to form a low-rank approximation of $\boldsymbol{J}^*\boldsymbol{J}$. The approximations are compared against $[\![\boldsymbol{J}^*\boldsymbol{J}]\!]_p$, and expected errors were estimated by 500 runs. Figure 3 showcases the difference between using $\boldsymbol{J}$ and $\boldsymbol{J}^*$ as input into Algorithm 1. As suggested in section 4.2, using $\boldsymbol{J}$ as input for even v and $\boldsymbol{J}^*$ as input for odd v is most accurate. This is like using the Nyström-type approach (Algorithm 2). The alternative choices for Algorithm 1,

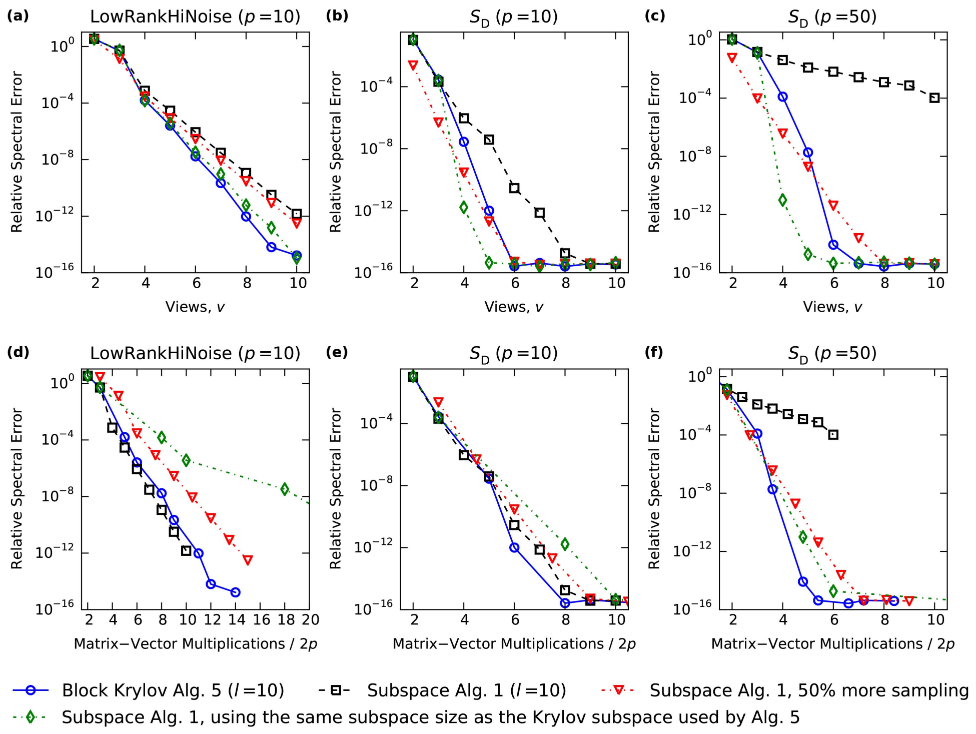

FIG. 4. (a)–(c) *Improvement in matrix approximation errors when increasing the number of matrix passes or views, when using the generalized subspace iteration method (Algorithm* 1*) and the generalized block Krylov algorithm (Algorithm* 5*).* (d)–(f) *Errors with respect to the number of matrix-vector multiplications.*

which equate to using the pinched method (Algorithm 3), perform worse. Notice that for some problems, a poor choice of input into Algorithm 1 can result in little gain in accuracy over using the same input matrix and one less view. However, for some problems the difference can be small.

8.3.3. Subspace iteration compared with block Krylov approach. We also consider differences between applying the block Krylov Algorithm 5 and the simpler subspace iteration Algorithm 1. Figure 4 compares the pass efficiency of Algorithm 5 and Algorithm 1 when averaging over 50 runs. The results show for fixed oversampling and views that the block Krylov method is more accurate. However, the subspace iteration method can be made more accurate by oversampling more. For the matrices with reasonably rapid spectral decay, additional oversampling can make the pass efficiency of Algorithm 1 comparable to Algorithm 5. Figure 4 shows that for reasonably decaying matrices, Algorithm 1 can achieve accuracy akin to Algorithm 5 using a similar number of matrix-vector multiplications and views. This is good to know, since the main computational cost for the Jacobian matrices motivating this study is from the number of matrix-vector multiplications.

Considering these results and the fact that Algorithm 5 has higher computational cost associated with the large Krylov space, Algorithm 1 may be advantageous for some problems. Furthermore, for each view, Algorithm 1 involves multiplying the input matrix with a matrix having a fixed number of columns. This may be more suitable when dealing with adjoint and direct simulations, as the sample size $p+l$

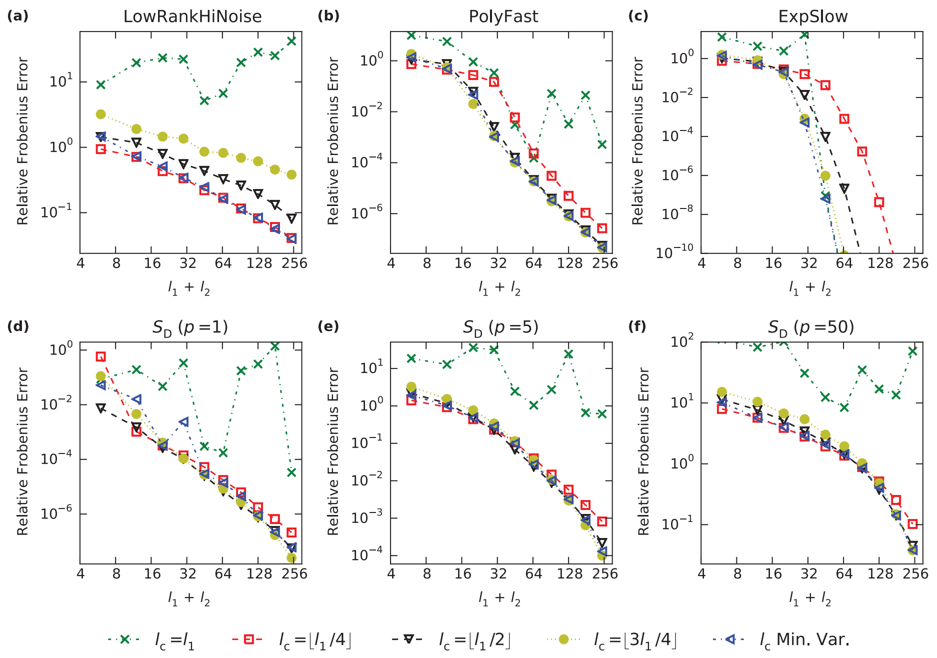

FIG. 5. *Relative Frobenius errors when using Algorithm* 4 *with* $l_1 \simeq l_2$ ($l_1 = \lfloor (l_1+l_2)/2 \rfloor = \lfloor T/2 \rfloor - p$). *The rank of the approximations is* $p=5$, *unless specified otherwise. The plots compare the performance of choosing a fixed* l_c *using* (6.5) *with choosing* l_c *adaptively according to the minimum variance (Min. Var.) approach* (6.8).

could be tailored to the available hardware. Algorithm 5, on the other hand, would involve a variable number of adjoint and direct solves, which could be harder to optimize. However, for matrices with trailing singular values that decay slowly, the block Krylov approach can be better; see Figure 4 and [32]. Figures 4(a) and (d) show that increasing l may not be worth it when the decay is slow, since considerably more oversampling and matrix-vector multiplications may be needed to notably affect accuracy.

8.4. Comparison of 1-view methods. Section 6 gives templates for 1-view variants that potentially improve state-of-the-art 1-view methods given in [46]. Here we are mainly interested in Algorithm 4 when using $l_1 \simeq l_2$. By $l_1 \simeq l_2$ we mean $l_1 = \lfloor T/2 \rfloor - p$ (which gives $l_1 = l_2$ or $l_1 = l_2 - 1$). The relative errors presented in this section are averages over 50 trials.

8.4.1. Choosing l_c. Figure 5 compares, for $l_1 \simeq l_2$, schemes where l_c is chosen a priori as a fixed ratio of l_1 (6.5) or using the minimum variance approach (6.8). The minimum variance approach uses postprocessing to choose l_c every time a low-rank approximation is generated. Therefore, l_c can vary between runs, although other parameters stay fixed. To increase accuracy, l_c should take on larger values when the spectrum decays fast. But notice from Figure 5 how a large l_c, especially $l_c = l_1$, is a bad idea unless the spectral decay is rapid and l_1 is large enough. These results for the relationship between l_c and l_1 are analogous to the results for the relationship between l_1 and l_2 when using the baseline 1-view method [46]. From Figure 5 we draw the conclusion that the simple choice $l_c = \lfloor l_1/2 \rfloor$ works quite well overall. However,

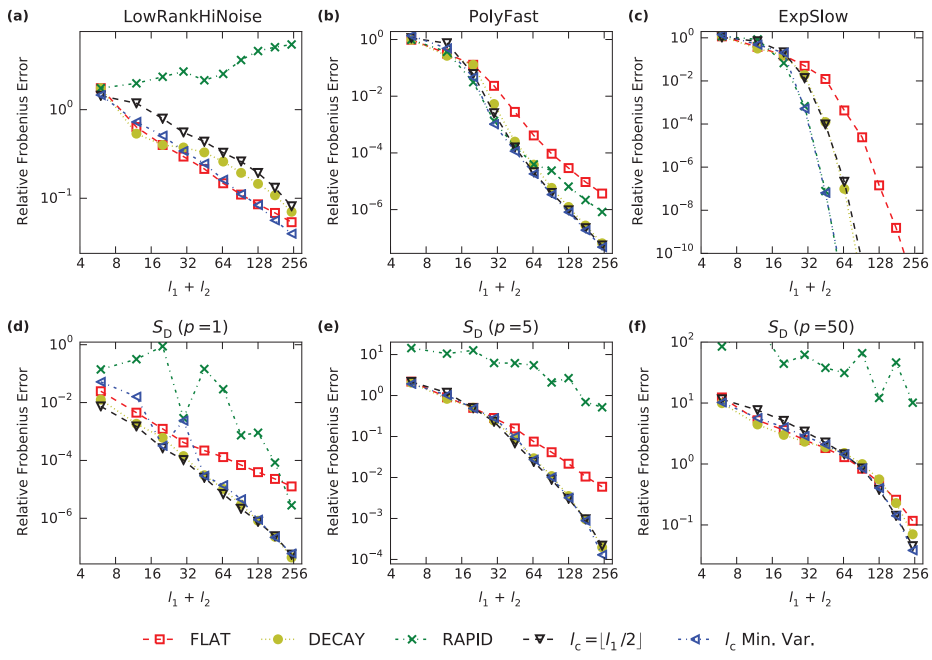

FIG. 6. *Relative Frobenius errors when using Algorithm* 4 *compared against the baseline* 1*-view algorithm (see section* 6.1*). The rank of the approximations is* p=5 *unless specified otherwise. For Algorithm* 4 *we consider* $l_1 \simeq l_2$ *with* l_c *chosen as* $\lfloor l_1/2 \rfloor$ *or chosen adaptively by the minimum variance (Min. Var.) approach* (6.8)*. For the baseline method, we consider oversampling schemes for flat (*`FLAT`*:* (6.1)*), moderately decaying (*`DECAY`*:* (6.2)*), and rapidly decaying (*`RAPID`*:* (6.3)*) singular spectra.*

like the baseline oversampling schemes proposed by [46], the fixed l_c schemes require some prior ideas about the expected spectral decay to get peak performance. The adaptive minimum variance approach, on the other hand, performs close to the best for most of the test problems and oversampling budgets.

8.4.2. Baseline method compared with $l_1 \simeq l_2$ variants. Figure 6 compares the baseline 1-view approach with Algorithm 4 using $l_1 \simeq l_2$, with l_c chosen using the minimum variance approach or $l_c = \lfloor l_1/2 \rfloor$. For the baseline method, we ran the three oversampling schemes proposed by [46]. Figure 6 shows that Algorithm 4 with $l_1 \simeq l_2$ performs similarly to the baseline 1-view approaches. The scheme using $l_c = \lfloor l_1/2 \rfloor$ works similarly to the most widely applicable baseline scheme (6.2). Again, the minimum variance approach worked the best overall.

8.4.3. Optimal performance and the extended scheme. To further compare the 1-view methods, we look at their optimal performance using a comparison similar to [46]. We consider the baseline 1-view method, Algorithm 4 with $l_1 \simeq l_2$, and the two extended sketching variants. Here we try to estimate the *best performance* a method can be expected to achieve for a given sampling budget T. The best performance of a method was found by running each method 50 times for feasible sampling parameters given T and choosing the fixed oversampling scheme that gave the minimum average Frobenius error. This differs slightly from the *oracle performance* measure used by Tropp et al. [46], where they ran each method for feasible parameters and found the minimum error. They repeated this and took the average minimum error over 20 trials. The oracle performance measure gives lower errors than the best performance measure. In our opinion, the best performance measure better reflects the peak performance that we could hope to achieve.

Figure 7 compares the best performance for the baseline method, Algorithm 4 with $l_1 \simeq l_2$, and the extended sketch methods. As claimed in section 6.6 the proposed modification to the extended sketch (6.11) performs substantially better than using (6.10). Using (6.11) also retains the good performance of (6.10) for small sampling and flat spectra. Overall, for a fixed memory budget T the improved extended scheme (6.11) performs the best. The baseline 1-view method and Algorithm 4 ($l_1 \simeq l_2$) have similar peak performance characteristics. But how do practical implementations compare to the best performance?

Figure 8 shows approximation errors that are achievable in practice using the minimum variance approach and shows the best performance for Algorithm 4 with $l_1 \simeq l_2$. The minimum variance scheme does well and gives errors comparable to the best performance. Figure 8 also depicts results of using (6.11) with sampling parameters chosen by (6.12). This scheme, arrived at by simple trial and error, results in errors close to the best performance. The oversampling scheme (6.12) and the minimum variance method worked well for the matrices we tested, but this may not necessarily be the case in general. However, the presented 1-view methods and oversampling schemes are promising.

9. Conclusions.

We have presented practical randomized algorithms which are variations of popular randomized algorithms for a rank-p approximation of a rectangular matrix. The study was aimed at practical algorithms that can be applied to a range of problems which may have an arbitrary budget for the number of times the matrix is accessed or viewed. To this end, we considered a more general randomized subspace iteration algorithm that generates a rank-p factorization using any number of views $v \geq 2$ as a substitute for a widely used subspace algorithm which only uses an even number of views. We also extended this approach to a block Krylov framework, which can be advantageous for matrices that have a heavy tail in their singular spectrum.

For applications that can only afford viewing the data matrix once, we presented improved randomized 1-view methods. Adapting a 1-view algorithm recently recommended by [46], we arrived at a simple algorithm and sampling scheme that can outperform the algorithm recommended by [46] for a range of matrices. The 1-view approach proposed here achieves this by using an adaptive postprocessing step which analyzes in a simple way the information contained in the randomized sketches created by the 1-view scheme. We also improve an extended 1-view method which was tested by [46]. A simple adjustment to the extended 1-view method makes it considerably more accurate, and it can outperform the previously mentioned 1-view method when the goal is to minimize the memory required by the randomized matrix sketches. Furthermore, based on 1-view and power iteration methods proposed by [57, 58], we proposed a highly pass-efficient randomized block Krylov algorithm that is suited to approximating large matrices stored out-of-core in either row-major or column-major format.

We also discussed how randomized algorithms can be applied to the type of nonlinear and large-scale inverse problems motivating this study. In particular, we discussed various randomized strategies for approximating LM model updates at a low computational cost. In relation to the motivating problem, we discussed subtle differences, accuracy wise, between applying the randomized subspace or block Krylov algorithms

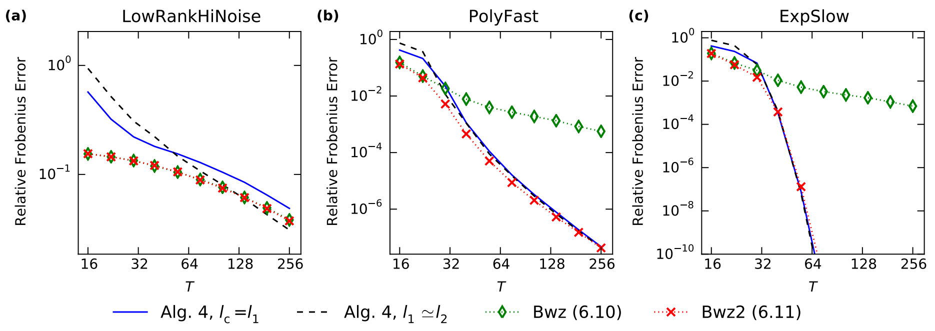

FIG. 7. *Best performance of Algorithm* 4 *(with* $l_1 \simeq l_2$*), the baseline* 1*-view algorithm (see section* 6.1*), an inaccurate extended* 1*-view method* (6.10)*, and an improved extended* 1*-view method* (6.11)*. The methods are compared in terms of their performance with respect to the memory budget* T*. All tests used a fixed rank* $p=5$ *for the approximations.*

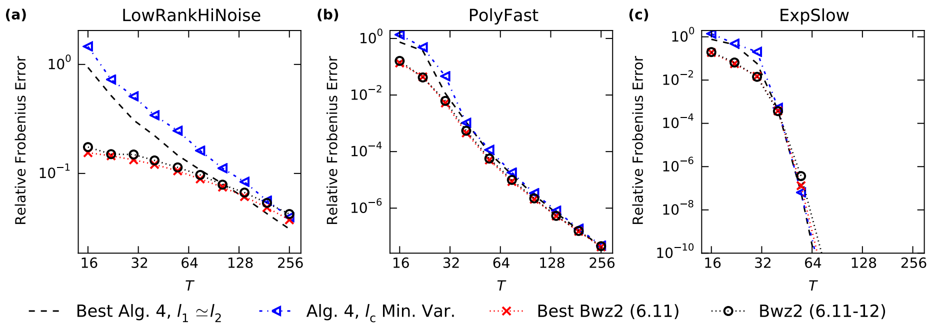

FIG. 8. *Best performance of Algorithm* 4 *(with* $l_1 \simeq l_2$*) and an improved extended* 1*-view method* (6.11) *compared with practical versions of both methods. The methods are compared in terms of their performance with respect to the memory budget* T*. All tests used a fixed rank* $p=5$ *for the approximations.* `Min. Var.` *refers to using Algorithm* 4 *with* (6.8).

to a matrix of interest or its transpose. The subtle differences may be important when estimating normal matrices, such as those appearing in inverse problems, and are important in applications that involve estimating either the left-singular or right-singular subspace of a matrix. Computational experiments supported the claims made on the properties of the presented randomized algorithms.

Acknowledgments. The author thanks Farbod (Fred) Roosta-Khorasani for thoughtful discussions on randomized methods, and Prof. Michael J. O'Sullivan, Oliver J. Maclaren, Ruanui Nicholson, and N. Benjamin Erichson for helpful discussions and feedback on the manuscript. The author also appreciates the constructive comments made by the anonymous reviewers. The author thanks the reviewer who pointed out that the extended 1-view method was recently analyzed further in technical report [47], which was made available online during the review of this manuscript.

REFERENCES

[1] H. AVRON, P. MAYMOUNKOV, AND S. TOLEDO, *Blendenpik: Supercharging LAPACK's least-squares solver*, SIAM J. Sci. Comput., 32 (2010), pp. 1217–1236, https://doi.org/10.1137/090767911.

[2] E. K. Bjarkason, O. J. Maclaren, J. P. O'Sullivan, and M. J. O'Sullivan, *Randomized truncated SVD Levenberg-Marquardt approach to geothermal natural state and history matching*, Water Resources Res., 54 (2018), pp. 2376–2404.

[3] C. Boutsidis, D. P. Woodruff, and P. Zhong, *Optimal principal component analysis in distributed and streaming models*, in Proc. 48th Annual ACM Symposium on Theory of Computing, Cambridge, MA, 2016, pp. 236–249.

[4] H. M. Bücker, G. Corliss, P. Hovland, U. Naumann, and B. Norris, eds., *Automatic Differentiation: Applications, Theory, and Implementations*, Lecture Notes in Comput. Sci. Eng. 50, Springer-Verlag, 2006.

[5] T. Bui-Thanh, C. Burstedde, O. Ghattas, J. Martin, G. Stadler, and L. C. Wilcox, *Extreme-scale UQ for Bayesian inverse problems governed by PDEs*, in Proc. International Conference for High Performance Computing, Networking, Storage and Analysis, Salt Lake City, UT, 2012, pp. 1–11.

[6] J. Carrera, A. Alcolea, A. Medina, J. Hidalgo, and L. J. Slooten, *Inverse problem in hydrogeology*, Hydrogeology J., 13 (2005), pp. 206–222.

[7] K. L. Clarkson and D. P. Woodruff, *Low-rank approximation and regression in input sparsity time*, J. ACM, 63 (2017), 54.

[8] M. B. Cohen, S. Elder, C. Musco, C. Musco, and M. Persu, *Dimensionality reduction for k-means clustering and low rank approximation*, in Proc. 47th Annual ACM Symposium on Theory of Computing, Portland, OR, 2015, pp. 163–172.

[9] T. Cui, K. J. Law, and Y. M. Marzouk, *Dimension-independent likelihood-informed MCMC*, J. Comput. Phys., 304 (2016), pp. 109–137.

[10] T. Cui, J. Martin, Y. M. Marzouk, A. Solonen, and A. Spantini, *Likelihood-informed dimension reduction for nonlinear inverse problems*, Inverse Problems, 30 (2014), 114015.

[11] P. Drineas, I. C. F. Ipsen, E. M. Kontopoulou, and M. Magdon-Ismail, *Structural Convergence Results for Approximation of Dominant Subspaces from Block Krylov Spaces*, preprint, https://arxiv.org/abs/1609.00671v2, 2017.

[12] P. Drineas, M. W. Mahoney, S. Muthukrishnan, and T. Sarlós, *Faster least squares approximation*, Numer. Math., 117 (2010), pp. 219–249.

[13] N. B. Erichson, A. Mendible, S. Wihlborn, and J. N. Kutz, *Randomized nonnegative matrix factorization*, Pattern Recognition Lett., 104 (2018), pp. 1–7.

[14] N. B. Erichson, S. Voronin, S. L. Brunton, and J. N. Kutz, *Randomized matrix decompositions using R*, J. Statist. Softw., 89 (2019), pp. 1–48.

[15] A. Gittens and M. W. Mahoney, *Revisiting the Nyström method for improved large-scale machine learning*, J. Mach. Learn. Res., 17 (2016), pp. 1–65.

[16] R. M. Gower and P. Richtárik, *Randomized iterative methods for linear systems*, SIAM J. Matrix Anal. Appl., 36 (2015), pp. 1660–1690, https://doi.org/10.1137/15M1025487.

[17] A. Griewank and A. Walther, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, 2nd ed., SIAM, 2008, https://doi.org/10.1137/1.9780898717761.

[18] M. Gu, *Subspace iteration randomization and singular value problems*, SIAM J. Sci. Comput., 37 (2015), pp. A1139–A1173, https://doi.org/10.1137/130938700.

[19] N. Halko, P.-G. Martinsson, Y. Shkolnisky, and M. Tygert, *An algorithm for the principal component analysis of large data sets*, SIAM J. Sci. Comput., 33 (2011), pp. 2580–2594, https://doi.org/10.1137/100804139.

[20] 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, https://doi.org/10.1137/090771806.

[21] M. Hinze, R. Pinnau, M. Ulbrich, and S. Ulbrich, *Optimization with PDE Constraints*, Math. Model. Theory Appl. 23, Springer, 2009.

[22] H. Li, G. C. Linderman, A. Szlam, K. P. Stanton, Y. Kluger, and M. Tygert, *Algorithm* 971: *An implementation of a randomized algorithm for principal component analysis*, ACM Trans. Math. Softw., 43 (2017), 28.

[23] E. Liberty, F. Woolfe, P.-G. Martinsson, V. Rokhlin, and M. Tygert, *Randomized algorithms for the low-rank approximation of matrices*, Proc. Natl. Acad. Sci. USA, 104 (2007), pp. 20167–20172.

[24] M. W. Mahoney, *Randomized algorithms for matrices and data*, Found. Trends Mach. Learn., 3 (2011), pp. 123–224.

[25] P.-G. Martinsson, *Randomized Methods for Matrix Computations and Analysis of High Dimensional Data*, preprint, https://arxiv.org/abs/1607.01649v1, 2016.

[26] P.-G. Martinsson, G. Quintana-Ortí, and N. Heavner, *randUTV: A Blocked Randomized Algorithm for Computing a Rank-Revealing UTV Factorization*, preprint, https://arxiv.org/abs/1703.00998, 2017.

[27] P.-G. Martinsson, V. Rokhlin, and M. Tygert, *A Randomized Algorithm for the Approximation of Matrices*, Tech. Report YALEU/DCS/RR-1361, Computer Science Department, Yale University, New Haven, CT, 2006.

[28] 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.

[29] P.-G. Martinsson, A. Szlam, and M. Tygert, *Normalized power iterations for the computation of SVD*, in Proceedings of the Neural and Information Processing Systems (NIPS) Workshop on Low-Rank Methods for Large-Scale Machine Learning, Vancouver, Canada, 2011, http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.683.7657.

[30] P.-G. Martinsson and S. Voronin, *A randomized blocked algorithm for efficiently computing rank-revealing factorizations of matrices*, SIAM J. Sci. Comput., 38 (2016), pp. S485–S507, https://doi.org/10.1137/15M1026080.

[31] X. Meng, M. A. Saunders, and M. W. Mahoney, *LSRN: A parallel iterative solver for strongly over- or underdetermined systems*, SIAM J. Sci. Comput., 36 (2014), pp. C95–C118, https://doi.org/10.1137/120866580.

[32] C. Musco and C. Musco, *Randomized block Krylov methods for stronger and faster approximate singular value decomposition*, in Advances in Neural Information Processing Systems 28, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, eds., Curran Associates, Inc., 2015, pp. 1396–1404.

[33] D. S. Oliver, A. C. Reynolds, and N. Liu, *Inverse Theory for Petroleum Reservoir Characterization and History Matching*, Cambridge University Press, Cambridge, UK, 2008.

[34] M. J. O'Sullivan, K. Pruess, and M. J. Lippmann, *State of the art of geothermal reservoir simulation*, Geothermics, 30 (2001), pp. 395–429.

[35] N. Petra, J. Martin, G. Stadler, and O. Ghattas, *A computational framework for infinite-dimensional Bayesian inverse problems, part* II: *Stochastic Newton MCMC with application to ice sheet flow inverse problems*, SIAM J. Sci. Comput., 36 (2014), pp. A1525–A1555, https://doi.org/10.1137/130934805.

[36] J. R. P. Rodrigues, *Calculating derivatives for automatic history matching*, Comput. Geosci., 10 (2006), pp. 119–136.

[37] V. Rokhlin, A. Szlam, and M. Tygert, *A randomized algorithm for principal component analysis*, SIAM J. Matrix Anal. Appl., 31 (2009), pp. 1100–1124, https://doi.org/10.1137/080736417.

[38] V. Rokhlin and M. Tygert, *A fast randomized algorithm for overdetermined linear least-squares regression*, Proc. Natl. Acad. Sci. USA, 105 (2008), pp. 13212–13217.

[39] A. K. Saibaba, *Analysis of Randomized Subspace Iteration: Canonical Angles and Unitarily Invariant Norms*, preprint, https://arxiv.org/abs/1804.02614, 2018.

[40] T. Sarlós, *Improved approximation algorithms for large matrices via random projections*, in Proc. 47th Annual IEEE Symposium on Foundations of Computer Science, Berkeley, CA, 2006, pp. 143–152.

[41] M. G. Shirangi, *History matching production data and uncertainty assessment with an efficient TSVD parameterization algorithm*, J. Petrol. Sci. Eng., 113 (2014), pp. 54–71.

[42] M. G. Shirangi and A. A. Emerick, *An improved TSVD-based Levenberg–Marquardt algorithm for history matching and comparison with Gauss–Newton*, J. Petrol. Sci. Eng., 143 (2016), pp. 258–271.

[43] R. Tavakoli and A. C. Reynolds, *History matching with parameterization based on the singular value decomposition of a dimensionless sensitivity matrix*, SPE J., 15 (2010), pp. 495–508.

[44] R. Tavakoli and A. C. Reynolds, *Monte Carlo simulation of permeability fields and reservoir performance predictions with SVD parameterization in RML compared with EnKF*, Comput. Geosci., 15 (2011), pp. 99–116.

[45] J. A. Tropp, A. Yurtsever, M. Udell, and V. Cevher, *Fixed-rank approximation of a positive-semidefinite matrix from streaming data*, in Proc. 31st International Conference on Neural Information Processing Systems, Long Beach, CA, 2017, pp. 1225–1234.

[46] J. A. Tropp, A. Yurtsever, M. Udell, and V. Cevher, *Practical sketching algorithms for low-rank matrix approximation*, SIAM J. Matrix Anal. Appl., 38 (2017), pp. 1454–1485, https://doi.org/10.1137/17M1111590.

[47] J. A. Tropp, A. Yurtsever, M. Udell, and V. Cevher, *More Practical Sketching Algorithms for Low-Rank Matrix Approximation*, ACM Report 2018-01, Caltech, Pasadena, CA, 2018.

[48] J. Upadhyay, *Fast and Space-Optimal Low-Rank Factorization in the Streaming Model with Application in Differential Privacy*, preprint, https://arxiv.org/abs/1604.01429v1, 2016.

[49] C. R. Vogel and J. G. Wade, *A modified Levenberg-Marquardt algorithm for large-scale inverse problems*, in Computation and Control III, Birkhäuser Boston, Boston, MA, 1993, pp. 367–378.

[50] C. R. Vogel and J. G. Wade, *Iterative SVD-based methods for ill-posed problems*, SIAM J. Sci. Comput., 15 (1994), pp. 736–754.

[51] S. Voronin and P.-G. Martinsson, *RSVDPACK: An Implementation of Randomized Algorithms for Computing the Singular Value, Interpolative, and CUR Decompositions of Matrices on Multi-core and GPU Architectures*, preprint, https://arxiv.org/abs/1502.05366v3, 2016.

[52] S. Voronin, D. Mikesell, and G. Nolet, *Compression approaches for the regularized solutions of linear systems from large-scale inverse problems*, Int. J. Geomath., 6 (2015), pp. 251–294.

[53] S. Wang, L. Luo, and Z. Zhang, *SPSD matrix approximation vis column selection: Theories, algorithms, and extensions*, J. Mach. Learn. Res., 17 (2016), pp. 1–49.

[54] D. P. Woodruff, *Sketching as a tool for numerical linear algebra*, Found. Trends Theoret. Comput. Sci., 10 (2014), pp. 1–157.

[55] F. Woolfe, E. Liberty, V. Rokhlin, and M. Tygert, *A fast randomized algorithm for the approximation of matrices*, Appl. Comput. Harmon. Anal., 25 (2008), pp. 335–366.

[56] H. Xiang and J. Zou, *Regularization with randomized SVD for large-scale discrete inverse problems*, Inverse Problems, 29 (2013), 085008.

[57] W. Yu, Y. Gu, J. Li, S. Liu, and Y. Li, *Single-pass PCA of large high-dimensional data*, in Proc. 26th International Joint Conference on Artificial Intelligence (IJCAI-17), Melbourne, Australia, 2017, pp. 3350–3356.

[58] W. Yu, Y. Gu, and Y. Li, *Efficient randomized algorithms for the fixed-precision low-rank matrix approximation*, SIAM J. Matrix Anal. Appl., 39 (2018), pp. 1339–1359.

[59] A. Yurtsever, M. Udell, J. A. Tropp, and V. Cevher, *Sketchy decisions: Convex low-rank matrix optimization with optimal storage*, in Proc. 20th Intl. Conf. on Artificial Intelligence and Statistics, 2017.