SIAM Review
Vol. 53, No. 2, pp. 217–288

Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions*

N. Halko†
P. G. Martinsson†
J. A. Tropp‡

Abstract. Low-rank matrix approximations, such as the truncated singular value decomposition and the rank-revealing QR decomposition, play a central role in data analysis and scientific computing. This work surveys and extends recent research which demonstrates that *randomization* offers a powerful tool for performing low-rank matrix approximation. These techniques exploit modern computational architectures more fully than classical methods and open the possibility of dealing with truly massive data sets. This paper presents a modular framework for constructing randomized algorithms that compute partial matrix decompositions. These methods use random sampling to identify a subspace that captures most of the action of a matrix. The input matrix is then compressed—either explicitly or implicitly—to this subspace, and the reduced matrix is manipulated deterministically to obtain the desired low-rank factorization. In many cases, this approach beats its classical competitors in terms of accuracy, robustness, and/or speed. These claims are supported by extensive numerical experiments and a detailed error analysis. The specific benefits of randomized techniques depend on the computational environment. Consider the model problem of finding the k dominant components of the singular value decomposition of an $m \times n$ matrix. (i) For a dense input matrix, randomized algorithms require $O(mn \log(k))$ floating-point operations (flops) in contrast to $O(mnk)$ for classical algorithms. (ii) For a sparse input matrix, the flop count matches classical Krylov subspace methods, but the randomized approach is more robust and can easily be reorganized to exploit multiprocessor architectures. (iii) For a matrix that is too large to fit in fast memory, the randomized techniques require only a constant number of passes over the data, as opposed to $O(k)$ passes for classical algorithms. In fact, it is sometimes possible to perform matrix approximation with a *single pass* over the data.

Key words. dimension reduction, eigenvalue decomposition, interpolative decomposition, Johnson–Lindenstrauss lemma, matrix approximation, parallel algorithm, pass-efficient algorithm, principal component analysis, randomized algorithm, random matrix, rank-revealing QR factorization, singular value decomposition, streaming algorithm

AMS subject classifications. Primary, 65F30; Secondary, 68W20, 60B20

DOI. 10.1137/090771806

*Received by the editors September 21, 2009; accepted for publication (in revised form) December 2, 2010; published electronically May 5, 2011.
http://www.siam.org/journals/sirev/53-2/77180.html

†Department of Applied Mathematics, University of Colorado at Boulder, Boulder, CO 80309-0526 (nathan.halko@colorado.edu, mastinss@colorado.edu). The work of these authors was supported by NSF awards 0748488, 0610097, and 941476.

‡Computing & Mathematical Sciences, California Institute of Technology, MC 305-16, Pasadena, CA 91125-5000 (jtropp@acm.caltech.edu). The work of this author was supported by ONR award N000140810883.

Contents.

Part I: Introduction.

1. Overview. On a well-known list of the "Top 10 Algorithms" that have influenced the practice of science and engineering during the 20th century [40], we find an entry that is not really an algorithm: the *idea* of using matrix factorizations to accomplish basic tasks in numerical linear algebra. In the accompanying article [128], Stewart explains that

> The underlying principle of the decompositional approach to matrix computation is that it is not the business of the matrix algorithmicists to solve particular problems but to construct computational platforms from which a variety of problems can be solved.

Stewart goes on to argue that this point of view has had many fruitful consequences, including the development of robust software for performing these factorizations in a highly accurate and provably correct manner.

The decompositional approach to matrix computation remains fundamental, but developments in computer hardware and the emergence of new applications in the information sciences have rendered the classical algorithms for this task inadequate in many situations:

- A salient feature of modern applications, especially in data mining, is that the matrices are stupendously big. Classical algorithms are not always well adapted to solving the type of large-scale problems that now arise.
- In the information sciences, it is common that data are missing or inaccurate. Classical algorithms are designed to produce highly accurate matrix decompositions, but it seems profligate to spend extra computational resources when the imprecision of the data inherently limits the resolution of the output.
- Data transfer now plays a major role in the computational cost of numerical algorithms. Techniques that require few passes over the data may be substantially faster in practice, even if they require as many—or more—floating-point operations (flops).
- As the structure of computer hardware continues to evolve, it becomes increasingly important for numerical algorithms to adapt to a range of novel architectures, such as graphics processing units.

The purpose of this paper is make the case that *randomized* algorithms provide a powerful tool for constructing approximate matrix factorizations. These techniques are simple and effective, sometimes impressively so. Compared with standard deterministic algorithms, the randomized methods are often faster and—perhaps surprisingly—more robust. Furthermore, they can produce factorizations that are accurate to any specified tolerance above machine precision, which allows the user to trade accuracy for speed if desired. We present numerical evidence that these algorithms succeed for real computational problems.

In short, our goal is to demonstrate how randomized methods interact with classical techniques to yield effective, modern algorithms supported by detailed theoretical guarantees. We have made a special effort to help practitioners identify situations where randomized techniques may outperform established methods.

Throughout this article, we provide detailed citations to previous work on randomized techniques for computing low-rank approximations. The primary sources that inform our presentation include [17, 46, 58, 92, 106, 113, 114, 119, 138].

Remark 1.1. Our experience suggests that many practitioners of scientific computing view randomized algorithms as a desperate and final resort. Let us address this concern immediately. Classical Monte Carlo methods are highly sensitive to the

random number generator and typically produce output with low and uncertain accuracy. In contrast, the algorithms discussed herein are relatively insensitive to the quality of randomness and produce highly accurate results. The probability of failure is a user-specified parameter that can be rendered negligible (say, less than 10^{-15}) with a nominal impact on the computational resources required.

1.1. Approximation by Low-Rank Matrices. The roster of standard matrix decompositions includes the pivoted QR factorization, the eigenvalue decomposition, and the singular value decomposition (SVD), all of which expose the (numerical) range of a matrix. Truncated versions of these factorizations are often used to express a *low-rank approximation* of a given matrix:

$$\underset{m \times n}{\boldsymbol{A}} \approx \underset{m \times k}{\boldsymbol{B}} \underset{k \times n.}{\boldsymbol{C}}, \tag{1.1}$$

The inner dimension k is sometimes called the *numerical rank* of the matrix. When the numerical rank is much smaller than either dimension m or n, a factorization such as (1.1) allows the matrix to be stored inexpensively and to be multiplied rapidly with vectors or other matrices. The factorizations can also be used for data interpretation or to solve computational problems, such as least squares.

Matrices with low numerical rank appear in a wide variety of scientific applications. We list only a few:

- A basic method in statistics and data mining is to compute the directions of maximal variance in vector-valued data by performing *principal component analysis* (PCA) on the data matrix. PCA is nothing other than a low-rank matrix approximation [71, sect. 14.5].
- Another standard technique in data analysis is to perform low-dimensional embedding of data under the assumption that there are fewer degrees of freedom than the ambient dimension would suggest. In many cases, the method reduces to computing a partial SVD of a matrix derived from the data. See [71, sects. 14.8–14.9] or [30].
- The problem of estimating parameters from measured data via least-squares fitting often leads to very large systems of linear equations that are close to linearly dependent. Effective techniques for factoring the coefficient matrix lead to efficient techniques for solving the least-squares problem [114].
- Many fast numerical algorithms for solving PDEs and for rapidly evaluating potential fields such as the fast multipole method [66] and $\mathcal{H}$-matrices [65] rely on low-rank approximations of continuum operators.
- Models of multiscale physical phenomena often involve PDEs with rapidly oscillating coefficients. Techniques for *model reduction* or *coarse graining* in such environments are often based on the observation that the linear transform that maps the input data to the requested output data can be approximated by an operator of low rank [56].

1.2. Matrix Approximation Framework. The task of computing a low-rank approximation to a given matrix can be split naturally into two computational stages. The first is to construct a low-dimensional subspace that captures the action of the matrix. The second is to restrict the matrix to the subspace and then compute a standard factorization (QR, SVD, etc.) of the reduced matrix. To be slightly more formal, we subdivide the computation as follows.

Stage A: Compute an approximate basis for the range of the input matrix $\boldsymbol{A}$. In other words, we require a matrix $\boldsymbol{Q}$ for which

$$\boldsymbol{Q} \text{ has orthonormal columns and } \boldsymbol{A} \approx \boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{A}. \tag{1.2}$$

We would like the basis matrix $\boldsymbol{Q}$ to contain as few columns as possible, but it is even more important to have an accurate approximation of the input matrix.

Stage B: Given a matrix $\boldsymbol{Q}$ that satisfies (1.2), we use $\boldsymbol{Q}$ to help compute a standard factorization (QR, SVD, etc.) of $\boldsymbol{A}$.

The task in Stage A can be executed very efficiently with random sampling methods, and these methods are the primary subject of this work. In the next subsection, we offer an overview of these ideas. The body of the paper provides details of the algorithms (section 4) and a theoretical analysis of their performance (sections 8–11).

Stage B can be completed with well-established deterministic methods. Section 3.3.3 contains an introduction to these techniques, and section 5 shows how we apply them to produce low-rank factorizations.

At this point in the development, it may not be clear why the output from Stage A facilitates our job in Stage B. Let us illustrate by describing how to obtain an approximate SVD of the input matrix $\boldsymbol{A}$ given a matrix $\boldsymbol{Q}$ that satisfies (1.2). More precisely, we wish to compute matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ with orthonormal columns and a nonnegative, diagonal matrix $\boldsymbol{\Sigma}$ such that $\boldsymbol{A} \approx \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^*$. This goal is achieved after three simple steps:

1. Form $\boldsymbol{B} = \boldsymbol{Q}^*\boldsymbol{A}$, which yields the low-rank factorization $\boldsymbol{A} \approx \boldsymbol{Q}\boldsymbol{B}$.
2. Compute an SVD of the small matrix: $\boldsymbol{B} = \widetilde{\boldsymbol{U}}\boldsymbol{\Sigma}\boldsymbol{V}^*$.
3. Set $\boldsymbol{U} = \boldsymbol{Q}\widetilde{\boldsymbol{U}}$.

When $\boldsymbol{Q}$ has few columns, this procedure is efficient because we can easily construct the reduced matrix $\boldsymbol{B}$ and rapidly compute its SVD. In practice, we can often avoid forming $\boldsymbol{B}$ explicitly by means of subtler techniques. In some cases, it is not even necessary to revisit the input matrix $\boldsymbol{A}$ during Stage B. This observation allows us to develop *single-pass algorithms*, which look at each entry of $\boldsymbol{A}$ only once.

Similar manipulations readily yield other standard factorizations, such as the pivoted QR factorization, the eigenvalue decomposition, etc.

1.3. Randomized Algorithms.

This paper describes a class of randomized algorithms for completing Stage A of the matrix approximation framework set forth in section 1.2. We begin with some details about the approximation problem these algorithms target (section 1.3.1). Afterward, we motivate the random sampling technique with a heuristic explanation (section 1.3.2) that leads to a prototype algorithm (section 1.3.3).

1.3.1. Problem Formulations.

The basic challenge in producing low-rank matrix approximations is a primitive question that we call the *fixed-precision approximation problem*. Suppose we are given a matrix $\boldsymbol{A}$ and a positive error tolerance ε. We seek a matrix $\boldsymbol{Q}$ with $k = k(\varepsilon)$ orthonormal columns such that

$$\|\boldsymbol{A} - \boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{A}\| \leq \varepsilon, \tag{1.3}$$

where $\|\cdot\|$ denotes the ℓ_2 operator norm. The range of $\boldsymbol{Q}$ is a k-dimensional subspace that captures most of the action of $\boldsymbol{A}$, and we would like k to be as small as possible.

The SVD furnishes an optimal answer to the fixed-precision problem [98]. Let σ_j denote the jth largest singular value of $\boldsymbol{A}$. For each $j \geq 0$,

$$\min_{\mathrm{rank}(\boldsymbol{X}) \leq j} \|\boldsymbol{A} - \boldsymbol{X}\| = \sigma_{j+1}. \tag{1.4}$$

One way to construct a minimizer is to choose $\boldsymbol{X} = \boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{A}$, where the columns of $\boldsymbol{Q}$ are k dominant left singular vectors of $\boldsymbol{A}$. Consequently, the minimal rank k where (1.3) holds equals the number of singular values of $\boldsymbol{A}$ that exceed the tolerance ε.

To simplify the development of algorithms, it is convenient to assume that the desired rank k is specified in advance. We call the resulting problem the *fixed-rank approximation problem*. Given a matrix $\boldsymbol{A}$, a target rank k, and an oversampling parameter p, we seek to construct a matrix $\boldsymbol{Q}$ with $k + p$ orthonormal columns such that

$$\|\boldsymbol{A} - \boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{A}\| \approx \min_{\mathrm{rank}(\boldsymbol{X}) \leq k} \|\boldsymbol{A} - \boldsymbol{X}\|. \tag{1.5}$$

Although there exists a minimizer $\boldsymbol{Q}$ that solves the fixed-rank problem for $p = 0$, the opportunity to use a small number of additional columns provides a flexibility that is crucial for the effectiveness of the computational methods we discuss.

We will demonstrate that algorithms for the fixed-rank problem can be adapted to solve the fixed-precision problem. The connection is based on the observation that we can build the basis matrix $\boldsymbol{Q}$ incrementally and, at any point in the computation, we can inexpensively estimate the residual error $\|\boldsymbol{A} - \boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{A}\|$. Refer to section 4.4 for the details of this reduction.

1.3.2. Intuition. To understand how randomness helps us solve the fixed-rank problem, it is helpful to consider some motivating examples.

First, suppose that we seek a basis for the range of a matrix $\boldsymbol{A}$ with *exact* rank k. Draw a random vector $\boldsymbol{\omega}$, and form the product $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{\omega}$. For now, the precise distribution of the random vector is unimportant; just think of $\boldsymbol{y}$ as a random sample from the range of $\boldsymbol{A}$. Let us repeat this sampling process k times:

$$\boldsymbol{y}^{(i)} = \boldsymbol{A}\boldsymbol{\omega}^{(i)}, \quad i = 1, 2, \ldots, k. \tag{1.6}$$

Owing to the randomness, the set $\{\boldsymbol{\omega}^{(i)} : i = 1, 2, \ldots, k\}$ of random vectors is likely to be in general linear position. In particular, the random vectors form a linearly independent set and no linear combination falls in the null space of $\boldsymbol{A}$. As a result, the set $\{\boldsymbol{y}^{(i)} : i = 1, 2, \ldots, k\}$ of sample vectors is also linearly independent, so it spans the range of $\boldsymbol{A}$. Therefore, to produce an orthonormal basis for the range of $\boldsymbol{A}$, we just need to orthonormalize the sample vectors.

Now, imagine that $\boldsymbol{A} = \boldsymbol{B} + \boldsymbol{E}$, where $\boldsymbol{B}$ is a rank-k matrix containing the information we seek and $\boldsymbol{E}$ is a small perturbation. Our priority is to obtain a basis that covers as much of the range of $\boldsymbol{B}$ as possible, rather than to minimize the number of basis vectors. Therefore, we fix a small number p, and we generate $k + p$ samples:

$$\boldsymbol{y}^{(i)} = \boldsymbol{A}\boldsymbol{\omega}^{(i)} = \boldsymbol{B}\boldsymbol{\omega}^{(i)} + \boldsymbol{E}\boldsymbol{\omega}^{(i)}, \quad i = 1, 2, \ldots, k + p. \tag{1.7}$$

The perturbation $\boldsymbol{E}$ shifts the direction of each sample vector outside the range of $\boldsymbol{B}$, which can prevent the span of $\{\boldsymbol{y}^{(i)} : i = 1, 2, \ldots, k\}$ from covering the entire range of $\boldsymbol{B}$. In contrast, the enriched set $\{\boldsymbol{y}^{(i)} : i = 1, 2, \ldots, k + p\}$ of samples has a much better chance of spanning the required subspace.

PROTO-ALGORITHM: SOLVING THE FIXED-RANK PROBLEM

Given an $m \times n$ matrix $\boldsymbol{A}$, a target rank k, and an oversampling parameter p, this procedure computes an $m \times (k+p)$ matrix $\boldsymbol{Q}$ whose columns are orthonormal and whose range approximates the range of $\boldsymbol{A}$.

1 Draw a random $n \times (k+p)$ test matrix $\boldsymbol{\Omega}$.
2 Form the matrix product $\boldsymbol{Y} = \boldsymbol{A}\boldsymbol{\Omega}$.
3 Construct a matrix $\boldsymbol{Q}$ whose columns form an orthonormal basis for the range of $\boldsymbol{Y}$.

Just how many extra samples do we need? Remarkably, for certain types of random sampling schemes, the failure probability decreases superexponentially with the oversampling parameter p; see (1.9). As a practical matter, setting $p = 5$ or $p = 10$ often gives superb results. This observation is one of the principal facts supporting the randomized approach to numerical linear algebra.

1.3.3. A Prototype Algorithm. The intuitive approach of section 1.3.2 can be applied to general matrices. Omitting computational details for now, we formalize the procedure in the figure labeled Proto-Algorithm.

This simple algorithm is by no means new. It is essentially the first step of a subspace iteration with a random initial subspace [61, sect. 7.3.2]. The novelty comes from the additional observation that the initial subspace should have a slightly higher dimension than the invariant subspace we are trying to approximate. With this revision, it is often the case that *no further iteration is required* to obtain a high-quality solution to (1.5). We believe this idea can be traced to [92, 106, 119].

In order to invoke the proto-algorithm with confidence, we must address several practical and theoretical issues:

- What random matrix $\boldsymbol{\Omega}$ should we use? How much oversampling do we need?
- The matrix $\boldsymbol{Y}$ is likely to be ill-conditioned. How do we orthonormalize its columns to form the matrix $\boldsymbol{Q}$?
- What are the computational costs?
- How can we solve the fixed-precision problem (1.3) when the numerical rank of the matrix is not known in advance?
- How can we use the basis $\boldsymbol{Q}$ to compute other matrix factorizations?
- Does the randomized method work for problems of practical interest? How does its speed/accuracy/robustness compare with standard techniques?
- What error bounds can we expect? with what probability?

The next few sections provide a summary of the answers to these questions. We describe several problem regimes where the proto-algorithm can be implemented efficiently, and we present a theorem that describes the performance of the most important instantiation. Finally, we elaborate on how these ideas can be applied to approximate the truncated SVD of a large data matrix. The rest of the paper contains a more exhaustive treatment—including pseudocode, numerical experiments, and a detailed theory.

1.4. A Comparison between Randomized and Traditional Techniques. To select an appropriate computational method for finding a low-rank approximation to a matrix, the practitioner must take into account the properties of the matrix. Is it dense or sparse? Does it fit in fast memory or is it stored out of core? Does the singular spectrum decay quickly or slowly? The behavior of a numerical linear algebra

algorithm may depend on all these factors [13, 61, 133]. To facilitate a comparison between classical and randomized techniques, we summarize their relative performance in each of three representative environments. Section 6 contains a more in-depth treatment.

We focus on the task of computing an approximate SVD of an $m \times n$ matrix $\boldsymbol{A}$ with numerical rank k. For randomized schemes, Stage A generally dominates the cost of Stage B in our matrix approximation framework (section 1.2). Within Stage A, the computational bottleneck is usually the matrix–matrix product $\boldsymbol{A\Omega}$ in step 2 of the proto-algorithm (section 1.3.3). The power of randomized algorithms stems from the fact that we can reorganize this matrix multiplication for maximum efficiency in a variety of computational architectures.

1.4.1. A General Dense Matrix That Fits in Fast Memory. A standard deterministic technique for computing an approximate SVD is to perform a rank-revealing QR factorization of the matrix, and then to manipulate the factors to obtain the final decomposition. The cost of this approach is typically $\mathrm{O}(kmn)$ flops, although these methods require slightly longer running times in rare cases [26, 68].

In contrast, randomized schemes can produce an approximate SVD using only $\mathrm{O}(mn \log(k) + (m+n)k^2)$ flops. The gain in asymptotic complexity is achieved by using a random matrix $\boldsymbol{\Omega}$ that has some internal structure, which allows us to evaluate the product $\boldsymbol{A\Omega}$ rapidly. For example, randomizing and subsampling the discrete Fourier transform works well. Sections 4.6, 6.1, and 11 contain more information on this approach.

1.4.2. A Matrix for which Matrix–Vector Products Can Be Evaluated Rapidly. When the matrix $\boldsymbol{A}$ is sparse or structured, we may be able to apply it rapidly to a vector. In this case, the classical prescription for computing a partial SVD is to invoke a Krylov subspace method, such as the Lanczos or Arnoldi algorithm. It is difficult to summarize the computational cost of these methods because their performance depends heavily on properties of the input matrix and on the amount of effort spent to stabilize the algorithm. (Inherently, the Lanczos and Arnoldi methods are numerically unstable.) For the same reasons, the error analysis of such schemes is unsatisfactory in many important environments.

At the risk of being overly simplistic, we claim that the typical cost of a Krylov method for approximating the k leading singular vectors of the input matrix is proportional to $k\,T_{\mathrm{mult}} + (m+n)k^2$, where T_{mult} denotes the cost of a matrix–vector multiplication with the input matrix and the constant of proportionality is small. We can also apply randomized methods using a Gaussian test matrix $\boldsymbol{\Omega}$ to complete the factorization at the same cost, $\mathrm{O}(k\,T_{\mathrm{mult}} + (m+n)k^2)$ flops.

With a given budget of flops, Krylov methods sometimes deliver a more accurate approximation than randomized algorithms. Nevertheless, the methods described in this survey have at least two powerful advantages over Krylov methods. First, the randomized schemes are inherently stable, and they come with very strong performance guarantees that do not depend on subtle spectral properties of the input matrix. Second, the matrix–vector multiplies required to form $\boldsymbol{A\Omega}$ can be performed *in parallel*. This fact allows us to restructure the calculations to take full advantage of the computational platform, which can lead to dramatic accelerations in practice, especially for parallel and distributed machines.

A more detailed comparison of randomized schemes and Krylov subspace methods is given in section 6.2.

1.4.3. A General Dense Matrix Stored in Slow Memory or Streamed. When the input matrix is too large to fit in core memory, the cost of transferring the matrix from slow memory typically dominates the cost of performing the arithmetic. The standard techniques for low-rank approximation described in section 1.4.1 require $\mathrm{O}(k)$ passes over the matrix, which can be prohibitively expensive.

In contrast, the proto-algorithm of section 1.3.3 requires only one pass over the data to produce the approximate basis $\boldsymbol{Q}$ for Stage A of the approximation framework. This straightforward approach, unfortunately, is not accurate enough for matrices whose singular spectrum decays slowly, but we can address this problem using very few (say, 2 to 4) additional passes over the data [113]. See section 1.6 or 4.5 for more discussion.

Typically, Stage B uses one additional pass over the matrix to construct the approximate SVD. With slight modifications, however, the two-stage randomized scheme can be revised so that it only makes a single pass over the data. Refer to section 5.5 for information.

1.5. Performance Analysis. A principal goal of this paper is to provide a detailed analysis of the performance of the proto-algorithm described in section 1.3.3. This investigation produces precise error bounds, expressed in terms of the singular values of the input matrix. Furthermore, we determine how several choices of the random matrix $\boldsymbol{\Omega}$ impact the behavior of the algorithm.

Let us offer a taste of this theory. The following theorem describes the average-case behavior of the proto-algorithm with a Gaussian test matrix, assuming we perform the computation in exact arithmetic. This result is a simplified version of Theorem 10.6.

THEOREM 1.1. *Suppose that $\boldsymbol{A}$ is a real $m \times n$ matrix. Select a target rank $k \geq 2$ and an oversampling parameter $p \geq 2$, where $k + p \leq \min\{m, n\}$. Execute the proto-algorithm with a standard Gaussian test matrix to obtain an $m \times (k+p)$ matrix $\boldsymbol{Q}$ with orthonormal columns. Then*

$$\mathbb{E}\,\|\boldsymbol{A} - \boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{A}\| \leq \left[1 + \frac{4\sqrt{k+p}}{p-1} \cdot \sqrt{\min\{m,n\}}\right] \sigma_{k+1}, \tag{1.8}$$

where $\mathbb{E}$ denotes expectation with respect to the random test matrix and σ_{k+1} is the $(k+1)$th singular value of $\boldsymbol{A}$.

We recall that the term σ_{k+1} appearing in (1.8) is the smallest possible error (1.4) achievable with any basis matrix $\boldsymbol{Q}$. The theorem asserts that, on average, the algorithm produces a basis whose error lies within a small polynomial factor of the theoretical minimum. Moreover, the error bound (1.8) in the randomized algorithm is slightly sharper than comparable bounds for deterministic techniques based on rank-revealing QR algorithms [68].

The reader might be worried about whether the expectation provides a useful account of the approximation error. Fear not: the actual outcome of the algorithm is *almost always* very close to the typical outcome because of measure concentration effects. As we discuss in section 10.3, the probability that the error satisfies

$$\|\boldsymbol{A} - \boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{A}\| \leq \left[1 + 9\sqrt{k+p} \cdot \sqrt{\min\{m,n\}}\right] \sigma_{k+1} \tag{1.9}$$

is at least $1 - 3 \cdot p^{-p}$ under very mild assumptions on p. This fact justifies the use of an oversampling term as small as $p = 5$. This simplified estimate is very similar to the major results in [92].

PROTOTYPE FOR RANDOMIZED SVD

Given an $m \times n$ matrix $\boldsymbol{A}$, a target number k of singular vectors, and an exponent q (say, $q = 1$ or $q = 2$), this procedure computes an approximate rank-$2k$ factorization $\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^*$, where $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthonormal, and $\boldsymbol{\Sigma}$ is nonnegative and diagonal.

Stage A:

1 Generate an $n \times 2k$ Gaussian test matrix $\boldsymbol{\Omega}$.
2 Form $\boldsymbol{Y} = (\boldsymbol{A}\boldsymbol{A}^*)^q \boldsymbol{A}\boldsymbol{\Omega}$ by multiplying alternately with $\boldsymbol{A}$ and $\boldsymbol{A}^*$.
3 Construct a matrix $\boldsymbol{Q}$ whose columns form an orthonormal basis for the range of $\boldsymbol{Y}$.

Stage B:

4 Form $\boldsymbol{B} = \boldsymbol{Q}^*\boldsymbol{A}$.
5 Compute an SVD of the small matrix: $\boldsymbol{B} = \widetilde{\boldsymbol{U}}\boldsymbol{\Sigma}\boldsymbol{V}^*$.
6 Set $\boldsymbol{U} = \boldsymbol{Q}\widetilde{\boldsymbol{U}}$.

Note: The computation of $\boldsymbol{Y}$ in step 2 is vulnerable to round-off errors. When high accuracy is required, we must incorporate an orthonormalization step between each application of $\boldsymbol{A}$ and $\boldsymbol{A}^*$; see Algorithm 4.4.

The theory developed in this paper provides much more detailed information about the performance of the proto-algorithm.

- When the singular values of $\boldsymbol{A}$ decay slightly, the error $\|\boldsymbol{A} - \boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{A}\|$ does not depend on the dimensions of the matrix (sections 10.2–10.3).
- We can reduce the size of the bracket in the error bound (1.8) by combining the proto-algorithm with a power iteration (section 10.4). For an example, see section 1.6 below.
- For the structured random matrices we mentioned in section 1.4.1, related error bounds are in force (section 11).
- We can obtain inexpensive a posteriori error estimates to verify the quality of the approximation (section 4.3).

1.6. Example: Randomized SVD. We conclude this introduction with a short discussion of how these ideas allow us to perform an approximate SVD of a large data matrix, which is a compelling application of randomized matrix approximation [113].

The two-stage randomized method offers a natural approach to SVD computations. Unfortunately, the simplest version of this scheme is inadequate in many applications because the singular spectrum of the input matrix may decay slowly. To address this difficulty, we incorporate q steps of a power iteration, where $q = 1$ or $q = 2$ usually suffices in practice. The complete scheme appears in the box labeled Prototype for Randomized SVD. For most applications, it is important to incorporate additional refinements, as we discuss in sections 4 and 5.

The Randomized SVD procedure requires only $2(q + 1)$ passes over the matrix, so it is efficient even for matrices stored out-of-core. The flop count satisfies

$$T_{\text{randSVD}} = (2q + 2)\, k\, T_{\text{mult}} + \mathrm{O}(k^2(m + n)),$$

where T_{mult} is the flop count of a matrix–vector multiply with $\boldsymbol{A}$ or $\boldsymbol{A}^*$. We have the following theorem on the performance of this method in exact arithmetic, which is a consequence of Corollary 10.10.

THEOREM 1.2. *Suppose that $\boldsymbol{A}$ is a real $m \times n$ matrix. Select an exponent q and a target number k of singular vectors, where $2 \le k \le 0.5 \min\{m, n\}$. Execute the*

Randomized SVD algorithm to obtain a rank-2k factorization $\boldsymbol{U\Sigma V}^*$. *Then*

$$\mathbb{E}\,\|\boldsymbol{A} - \boldsymbol{U\Sigma V}^*\| \le \left[1 + 4\sqrt{\frac{2\min\{m,n\}}{k-1}}\right]^{1/(2q+1)} \sigma_{k+1}, \tag{1.10}$$

where $\mathbb{E}$ *denotes expectation with respect to the random test matrix and* σ_{k+1} *is the* $(k+1)$*th singular value of* $\boldsymbol{A}$.

This result is new. Observe that the bracket in (1.10) is essentially the same as the bracket in the basic error bound (1.8). We find that the power iteration drives the leading constant to one exponentially fast as the power q increases. The rank-k approximation of $\boldsymbol{A}$ can never achieve an error smaller than σ_{k+1}, so the randomized procedure computes $2k$ approximate singular vectors that capture as much of the matrix as the first k actual singular vectors.

In practice, we can truncate the approximate SVD, retaining only the first k singular values and vectors. Equivalently, we replace the diagonal factor $\boldsymbol{\Sigma}$ by the matrix $\boldsymbol{\Sigma}_{(k)}$ formed by zeroing out all but the largest k entries of $\boldsymbol{\Sigma}$. For this truncated SVD, we have the error bound

$$\mathbb{E}\,\left\|\boldsymbol{A} - \boldsymbol{U\Sigma}_{(k)}\boldsymbol{V}^*\right\| \le \sigma_{k+1} + \left[1 + 4\sqrt{\frac{2\min\{m,n\}}{k-1}}\right]^{1/(2q+1)} \sigma_{k+1}. \tag{1.11}$$

In other words, we pay no more than an additive term σ_{k+1} when we perform the truncation step. Our numerical experience suggests that the error bound (1.11) is pessimistic. See Remark 5.1 and section 9.4 for some discussion of truncation.

1.7. Outline of Paper. The paper is organized into three parts: an introduction (sections 1–3), a description of the algorithms (sections 4–7), and a theoretical performance analysis (sections 8–11). The two latter parts commence with a short internal outline. Each part is more or less self-contained, and after a brief review of our notation in sections 3.1–3.2, the reader can proceed to either the algorithms or the theory part.

2. Related Work and Historical Context. Randomness has occasionally surfaced in the numerical linear algebra literature; in particular, it is quite standard to initialize iterative algorithms for constructing invariant subspaces with a randomly chosen point. Nevertheless, we believe that sophisticated ideas from random matrix theory have not been incorporated into classical matrix factorization algorithms until very recently. We can trace this development to earlier work in computer science and—especially—to probabilistic methods in geometric analysis. This section presents an overview of the relevant work. We begin with a survey of randomized methods for matrix approximation; then we attempt to trace some of the ideas backward to their sources.

2.1. Randomized Matrix Approximation. Matrices of low numerical rank contain little information relative to their apparent dimension owing to the linear dependency in their columns (or rows). As a result, it is reasonable to expect that these matrices can be approximated with far fewer degrees of freedom. A less obvious fact is that randomized schemes can be used to produce these approximations efficiently.

Several types of approximation techniques build on this idea. These methods all follow the same basic pattern:

1. Preprocess the matrix, usually to calculate sampling probabilities.

2. Take random samples from the matrix, where the term *sample* refers generically to a linear function of the matrix.
3. Postprocess the samples to compute a final approximation, typically with classical techniques from numerical linear algebra. This step may require another look at the matrix.

We continue with a description of the most common approximation schemes.

2.1.1. Sparsification. The simplest approach to matrix approximation is the method of *sparsification* or the related technique of *quantization*. The goal of sparsification is to replace the matrix by a surrogate that contains far fewer nonzero entries. Quantization produces an approximation whose components are drawn from a (small) discrete set of values. These methods can be used to limit storage requirements or to accelerate computations by reducing the cost of matrix–vector and matrix–matrix multiplies [95, Chap. 6]. The manuscript [33] describes applications in optimization.

Sparsification typically involves very simple elementwise calculations. Each entry in the approximation is drawn independently at random from a distribution determined from the corresponding entry of the input matrix. The expected value of the random approximation equals the original matrix, but the distribution is designed so that a typical realization is much sparser.

The first method of this form was devised by Achlioptas and McSherry [2], who built on earlier work on graph sparsification due to Karger [76, 77]. Arora, Hazan, and Kale presented a different sampling method in [7]. See [60, 124] for some recent work on sparsification.

2.1.2. Column Selection Methods. A second approach to matrix approximation is based on the idea that a small set of columns describes most of the action of a numerically low-rank matrix. Indeed, classical existential results [118] demonstrate that every $m \times n$ matrix $\boldsymbol{A}$ contains a k-column submatrix $\boldsymbol{C}$ for which

$$\left\| \boldsymbol{A} - \boldsymbol{C}\boldsymbol{C}^{\dagger}\boldsymbol{A} \right\| \leq \sqrt{1 + k(n-k)} \cdot \left\| \boldsymbol{A} - \boldsymbol{A}_{(k)} \right\|, \tag{2.1}$$

where k is a parameter, the dagger $\dagger$ denotes the pseudoinverse, and $\boldsymbol{A}_{(k)}$ is a best rank-k approximation of $\boldsymbol{A}$. It is NP-hard to perform column selection by optimizing natural objective functions, such as the condition number of the submatrix [27]. Nevertheless, there are efficient deterministic algorithms, such as the strongly rank-revealing QR method of [68], that can nearly achieve the error bound (2.1).

There is a class of randomized algorithms that approach the fixed-rank approximation problem (1.5) using this intuition. These methods first compute a sampling probability for each column, using either the squared Euclidean norms of the columns or their *leverage scores*. (Leverage scores reflect the relative importance of the columns to the action of the matrix; they can be calculated easily from the dominant k right singular vectors of the matrix.) Columns are then selected randomly according to this distribution. Afterward, a postprocessing step is invoked to produce a more refined approximation of the matrix.

We believe that the earliest method of this form appeared in a 1998 paper of Frieze, Kannan, and Vempala [57, 58]. This work was refined substantially in the papers [43, 44, 46]. The basic algorithm samples columns from a distribution related to the squared ℓ_2 norms of the columns. This sampling step produces a small column submatrix whose range is aligned with the range of the input matrix. The final approximation is obtained from a truncated SVD of the submatrix. Given a target rank k and a parameter $\varepsilon > 0$, this approach samples $\ell = \ell(k, \varepsilon)$ columns of the

matrix to produce a rank-k approximation $\boldsymbol{B}$ that satisfies

$$\|\boldsymbol{A} - \boldsymbol{B}\|_{\mathrm{F}} \leq \left\|\boldsymbol{A} - \boldsymbol{A}_{(k)}\right\|_{\mathrm{F}} + \varepsilon \|\boldsymbol{A}\|_{\mathrm{F}}, \tag{2.2}$$

where $\|\cdot\|_{\mathrm{F}}$ denotes the Frobenius norm. We note that the algorithm of [46] requires only a constant number of passes over the data.

Rudelson and Vershynin later showed that the same type of column sampling method also yields spectral-norm error bounds [117]. The techniques in their paper have been very influential; their work has found other applications in randomized regression [52], sparse approximation [134], and compressive sampling [19].

Deshpande et al. [37, 38] demonstrated that the error in the column sampling approach can be improved by iteration and adaptive volume sampling. They showed that it is possible to produce a rank-k matrix $\boldsymbol{B}$ that satisfies

$$\|\boldsymbol{A} - \boldsymbol{B}\|_{\mathrm{F}} \leq (1 + \varepsilon) \left\|\boldsymbol{A} - \boldsymbol{A}_{(k)}\right\|_{\mathrm{F}} \tag{2.3}$$

using a k-pass algorithm. Around the same time, Har-Peled [70] independently developed a recursive algorithm that offers the same approximation guarantees. Very recently, Deshpande and Rademacher improved the running time of volume-based sampling methods [36].

Drineas et al. and Boutsidis et al. have also developed randomized algorithms for the *column subset selection problem*, which requests a column submatrix $\boldsymbol{C}$ that achieves a bound of the form (2.1). Via the methods of Rudelson and Vershynin [117], they showed that sampling columns according to their leverage scores is likely to produce the required submatrix [50, 51]. Subsequent work [17, 18] showed that postprocessing the sampled columns with a rank-revealing QR algorithm can reduce the number of output columns required (2.1). The argument in [17] explicitly decouples the linear algebraic part of the analysis from the random matrix theory. The theoretical analysis in the present work involves a very similar technique.

2.1.3. Approximation by Dimension Reduction. A third approach to matrix approximation is based on the concept of *dimension reduction*. Since the rows of a low-rank matrix are linearly dependent, they can be embedded into a low-dimensional space without altering their geometric properties substantially. A random linear map provides an efficient, nonadaptive way to perform this embedding. (Column sampling can also be viewed as an adaptive form of dimension reduction.)

The proto-algorithm we set forth in section 1.3.3 is simply a dual description of the dimension reduction approach: collecting random samples from the column space of the matrix is equivalent to reducing the dimension of the rows. No precomputation is required to obtain the sampling distribution, but the sample itself takes some work to collect. Afterward, we orthogonalize the samples as preparation for constructing various matrix approximations.

We believe that the idea of using dimension reduction for algorithmic matrix approximation first appeared in a 1998 paper of Papadimitriou et al. [105, 106], who described an application to latent semantic indexing (LSI). They suggested projecting the input matrix onto a random subspace and compressing the original matrix to (a subspace of) the range of the projected matrix. They established error bounds that echo the result (2.2) of Frieze, Kannan, and Vempala [58]. Although the Euclidean column selection method is a more computationally efficient way to obtain this type of error bound, dimension reduction has other advantages, e.g., in terms of accuracy.

Sarlós argued in [119] that the computational costs of dimension reduction can be reduced substantially by means of the structured random maps proposed by Ailon and

Chazelle [3]. Sarlós used these ideas to develop efficient randomized algorithms for least-squares problems; he also studied approximate matrix multiplication and low-rank matrix approximation. The recent paper [103] analyzes a very similar matrix approximation algorithm using Rudelson and Vershynin's methods [117].

The initial work of Sarlós on structured dimension reduction did not immediately yield algorithms for low-rank matrix approximation that were superior to classical techniques. Woolfe et al. showed how to obtain an improvement in asymptotic computational cost, and they applied these techniques to problems in scientific computing [138]. Related work includes [87, 89].

Martinsson, Rokhlin, and Tygert have studied dimension reduction using a Gaussian transform matrix, and they demonstrated that this approach performs much better than earlier analyses had suggested [92]. Their work highlights the importance of oversampling, and their error bounds are very similar to the estimate (1.9) we presented in the introduction. They also demonstrated that dimension reduction can be used to compute an interpolative decomposition of the input matrix, which is essentially equivalent to performing column subset selection.

Rokhlin, Szlam, and Tygert have shown that combining dimension reduction with a power iteration is an effective way to improve its performance [113]. These ideas lead to very efficient randomized methods for large-scale PCA [69]. An efficient, numerically stable version of the power iteration is discussed in section 4.5, as well as [93]. Related ideas appear in a paper of Roweis [115].

Clarkson and Woodruff [29] developed one-pass algorithms for performing low-rank matrix approximation, and they established lower bounds which prove that many of their algorithms have optimal or near-optimal resource guarantees, modulo constants.

2.1.4. Approximation by Submatrices. The matrix approximation literature contains a subgenre that discusses methods for building an approximation from a submatrix and computed coefficient matrices. For example, we can construct an approximation using a subcollection of columns (the interpolative decomposition), a subcollection of rows and a subcollection of columns (the CUR decomposition), or a square submatrix (the matrix skeleton). This type of decomposition was developed and studied in several papers, including [26, 64, 127]. For data analysis applications, see the recent paper [90].

A number of works develop randomized algorithms for this class of matrix approximations. Drineas et al. have developed techniques for computing CUR decompositions, which express $\boldsymbol{A} \approx \boldsymbol{CUR}$, where $\boldsymbol{C}$ and $\boldsymbol{R}$ denote small column and row submatrices of $\boldsymbol{A}$ and where $\boldsymbol{U}$ is a small linkage matrix. These methods identify columns (rows) that approximate the range (corange) of the matrix; the linkage matrix is then computed by solving a small least-squares problem. A randomized algorithm for CUR approximation with controlled absolute error appears in [47]; a relative error algorithm appears in [51]. We also mention a paper on computing a closely related factorization called the *compact matrix decomposition* [130].

It is also possible to produce interpolative decompositions and matrix skeletons using randomized methods, as discussed in [92, 113] and section 5.2 of the present work.

2.1.5. Other Numerical Problems. The literature contains a variety of other randomized algorithms for solving standard problems in and around numerical linear algebra. We list some of the basic references.

Tensor Skeletons. Randomized column selection methods can be used to produce CUR-type decompositions of higher-order tensors [49].

Matrix Multiplication. Column selection and dimension reduction techniques can be used to accelerate the multiplication of rank-deficient matrices [10, 45, 119].

Overdetermined Linear Systems. The randomized Kaczmarz algorithm is a linearly convergent iterative method that can be used to solve overdetermined linear systems [102, 129].

Overdetermined Least Squares. Fast dimension-reduction maps can sometimes accelerate the solution of overdetermined least-squares problems [52, 119].

Nonnegative Least Squares. Fast dimension-reduction maps can be used to reduce the size of nonnegative least-squares problems [16].

Preconditioned Least Squares. Randomized matrix approximations can be used to precondition conjugate gradient to solve least-squares problems [114].

Other Regression Problems. Randomized algorithms for ℓ_1 regression are described in [28]. Regression in ℓ_p for $p \in [1, \infty)$ has also been considered [31].

Facility Location. The Fermat–Weber facility location problem can be viewed as matrix approximation with respect to a different discrepancy measure. Randomized algorithms for this type of problem appear in [122].

2.1.6. Compressive Sampling. Although randomized matrix approximation and compressive sampling are based on some common intuitions, it is facile to consider either one as a subspecies of the other. We offer a short overview of the field of compressive sampling—especially the part connected with matrices—so we can highlight some of the differences.

The theory of compressive sampling starts with the observation that many types of vector-space data are *compressible*. That is, the data are approximated well using a short linear combination of basis functions drawn from a fixed collection [42]. For example, natural images are well approximated in a wavelet basis; numerically low-rank matrices are well approximated as a sum of rank-one matrices. The idea behind compressive sampling is that suitably chosen random samples from this type of compressible object carry a large amount of information. Furthermore, it is possible to reconstruct the compressible object from a small set of these random samples, often by solving a convex optimization problem. The initial discovery works of Candès, Romberg, and Tao [20] and Donoho [41] were written in 2004.

The earliest work in compressive sampling focused on vector-valued data; soon after, researchers began to study compressive sampling for matrices. In 2007, Recht, Fazel, and Parillo demonstrated that it is possible to reconstruct a rank-deficient matrix from Gaussian measurements [112]. More recently, Candès and Recht [22] and Candès and Tao [23] considered the problem of completing a low-rank matrix from a random sample of its entries.

The usual goals of compressive sampling are (i) to design a method for collecting informative, nonadaptive data about a compressible object and (ii) to reconstruct a compressible object given some measured data. In both cases, there is an implicit assumption that we have limited—if any—access to the underlying data.

In the problem of matrix approximation, we typically have a complete representation of the matrix at our disposal. The point is to compute a simpler representation as efficiently as possible under some operational constraints. In particular, we would like to perform as little computation as we can, but we are usually allowed to revisit the input matrix. Because of the different focus, randomized matrix approximation algo-

rithms require fewer random samples from the matrix and use fewer computational resources than compressive sampling reconstruction algorithms.

2.2. Origins. This section attempts to identify some of the major threads of research that ultimately led to the development of the randomized techniques we discuss in this paper.

2.2.1. Random Embeddings. The field of random embeddings is a major precursor to randomized matrix approximation. In a celebrated 1984 paper [74], Johnson and Lindenstrauss showed that the pairwise distances among a collection of N points in a Euclidean space are approximately maintained when the points are mapped randomly to a Euclidean space of dimension $\mathrm{O}(\log N)$. In other words, random embeddings preserve Euclidean geometry. Shortly afterward, Bourgain showed that appropriate random low-dimensional embeddings preserve the geometry of point sets in finite-dimensional ℓ_1 spaces [15].

These observations suggest that we might be able to solve some computational problems of a geometric nature more efficiently by translating them into a lower-dimensional space and solving them there. This idea was cultivated by the theoretical computer science community beginning in the late 1980s, with research flowering in the late 1990s. In particular, nearest-neighbor search can benefit from dimension-reduction techniques [73, 75, 79, 81]. The papers [57, 105] were apparently the first to apply this approach to linear algebra.

Around the same time, researchers became interested in simplifying the form of dimension reduction maps and improving the computational cost of applying the map. Several researchers developed refined results on the performance of a Gaussian matrix as a linear dimension reduction map [32, 73, 94]. Achlioptas demonstrated that discrete random matrices would serve nearly as well [1]. In 2006, Ailon and Chazelle proposed the *fast Johnson–Lindenstrauss transform* [3], which combines the speed of the FFT with the favorable embedding properties of a Gaussian matrix. Subsequent refinements appear in [4, 88]. Sarlós then imported these techniques to study several problems in numerical linear algebra, which has led to some of the fastest algorithms currently available [89, 138].

2.2.2. Data Streams. Muthukrishnan argues that a distinguishing feature of modern data is the manner in which it is *presented* to us. The sheer volume of information and the speed at which it must be processed tax our ability to *transmit* the data elsewhere, to *compute* complicated functions on the data, or to *store* a substantial part of the data [101, sect. 3]. As a result, computer scientists have started to develop algorithms that can address familiar computational problems under these novel constraints. The data stream phenomenon is one of the primary justifications cited by [45] for developing pass-efficient methods for numerical linear algebra problems, and it is also the focus of the recent treatment [29].

One of the methods for dealing with massive data sets is to maintain *sketches*, which are small summaries that allow functions of interest to be calculated. In the simplest case, a sketch is simply a random projection of the data, but it might be a more sophisticated object [101, sect. 5.1]. The idea of sketching can be traced to the work of Alon et al. [5, 6].

2.2.3. Numerical Linear Algebra. Classically, the field of numerical linear algebra has focused on developing deterministic algorithms that produce highly accurate matrix approximations with provable guarantees. Nevertheless, randomized techniques have appeared in several environments.

One of the original examples is the use of random models for arithmetical errors, which was pioneered by von Neumann and Goldstine. Their papers [136, 137] stand among the first works to study the properties of random matrices. The earliest numerical linear algebra algorithm that depends essentially on randomized techniques is probably Dixon's method for estimating norms and condition numbers [39].

Another situation where randomness commonly arises is the initialization of iterative methods for computing invariant subspaces. For example, most numerical linear algebra texts advocate random selection of the starting vector for the power method because it ensures that the vector has a nonzero component in the direction of a dominant eigenvector. Woźniakowski and coauthors have analyzed the performance of the power method and the Lanczos iteration given a random starting vector [80, 86].

Among other interesting applications of randomness, we mention the work by Parker and Pierce, which applies a randomized FFT to eliminate pivoting in Gaussian elimination [107]; work by Demmel, Dumitriu, and Holtz, who have studied randomization in connection with the stability of fast methods for linear algebra [35]; and work by Le and Parker utilizing randomized methods for stabilizing fast linear algebraic computations based on recursive algorithms, such as Strassen's matrix multiplication [82].

2.2.4. Scientific Computing. One of the first algorithmic applications of randomness is the method of Monte Carlo integration introduced by von Neumann and Ulam, and its extensions, such as the Metropolis algorithm for simulations in statistical physics [96]. (See [9] for an introduction.) The most basic technique is to estimate an integral by sampling m points from the measure and computing an empirical mean of the integrand evaluated at the sample locations:

$$\int f(x)\,\mathrm{d}\mu(x) \approx \frac{1}{m}\sum_{i=1}^{m} f(X_i),$$

where X_i are independent and identically distributed according to the probability measure μ. The law of large numbers (usually) ensures that this approach produces the correct result in the limit as $m \to \infty$. Unfortunately, the approximation error typically has a standard deviation of $m^{-1/2}$, and the method provides no certificate of success.

The disappointing computational profile of Monte Carlo integration seems to have inspired a distaste for randomized approaches within the scientific computing community. Fortunately, there are many other types of randomized algorithms—such as the ones in this paper—that do not suffer from the same shortcomings.

2.2.5. Geometric Functional Analysis. There is one more character that plays a central role in our story: the probabilistic method in geometric analysis. Many of the algorithms and proof techniques ultimately come from work in this beautiful but recondite corner of mathematics.

Dvoretzky's theorem [53] states (roughly) that every infinite-dimensional Banach space contains an n-dimensional subspace whose geometry is essentially the same as an n-dimensional Hilbert space, where n is an arbitrary natural number. In 1971, V. D. Milman developed a striking proof of this result by showing that a *random* n-dimensional subspace of an N-dimensional Banach space has this property with exceedingly high probability, provided that N is large enough [97]. Milman's article debuted the *concentration of measure phenomenon*, which is a geometric interpretation of the classical idea that regular functions of independent random variables rarely

deviate far from their mean. This work opened a new era in geometric analysis where the probabilistic method became a basic instrument.

Another prominent example of measure concentration is Kašin's computation of the Gel'fand widths of the ℓ_1 ball [78], subsequently refined in [59]. This work showed that a *random* $(N-n)$-dimensional projection of the N-dimensional ℓ_1 ball has an astonishingly small Euclidean diameter: approximately $\sqrt{(1+\log(N/n))/n}$. In contrast, a nonzero projection of the ℓ_2 ball always has Euclidean diameter one. This basic geometric fact undergirds recent developments in compressive sampling [21].

We have already described a third class of examples: the randomized embeddings of Johnson–Lindenstrauss [74] and of Bourgain [15].

Finally, we mention Maurey's technique of empirical approximation. The original work was unpublished; one of the earliest applications appears in [24, sect. 1]. Although Maurey's idea has not received as much press as the examples above, it can lead to simple and efficient algorithms for sparse approximation. For some examples in machine learning, consider [8, 85, 111, 120].

The importance of random constructions in the geometric analysis community has led to the development of powerful techniques for studying random matrices. Classical random matrix theory focuses on a detailed asymptotic analysis of the spectral properties of special classes of random matrices. In contrast, geometric analysts know methods for determining the approximate behavior of rather complicated finite-dimensional random matrices; see [34] for a survey. We also mention the works of Rudelson [116] and Rudelson and Vershynin [117], which describe powerful tools for studying random matrices drawn from certain discrete distributions. Their papers are rooted deeply in the field of geometric functional analysis, but they reach out toward computational applications.

3. Linear Algebraic Preliminaries.

This section summarizes the background we need for the detailed description of randomized algorithms in sections 4–6 and the analysis in sections 8–11. We introduce notation in section 3.1, describe some standard matrix decompositions in section 3.2, and briefly review standard techniques for computing matrix factorizations in section 3.3.

3.1. Basic Definitions.

The standard Hermitian geometry for $\mathbb{C}^n$ is induced by the inner product

$$\langle \boldsymbol{x},\ \boldsymbol{y} \rangle = \boldsymbol{x} \cdot \boldsymbol{y} = \sum\nolimits_j x_j\, \overline{y}_j.$$

The associated norm is

$$\|\boldsymbol{x}\|^2 = \langle \boldsymbol{x},\ \boldsymbol{x} \rangle = \sum\nolimits_j |x_j|^2.$$

We usually measure the magnitude of a matrix $\boldsymbol{A}$ with the operator norm

$$\|\boldsymbol{A}\| = \max_{\boldsymbol{x} \neq \boldsymbol{0}} \frac{\|\boldsymbol{A}\boldsymbol{x}\|}{\|\boldsymbol{x}\|},$$

which is often referred to as the *spectral norm*. The Frobenius norm is given by

$$\|\boldsymbol{A}\|_{\mathrm{F}} = \Big[\sum\nolimits_{jk} |a_{jk}|^2 \Big]^{1/2}.$$

The conjugate transpose, or *adjoint*, of a matrix $\boldsymbol{A}$ is denoted by $\boldsymbol{A}^*$. The important identities

$$\|\boldsymbol{A}\|^2 = \|\boldsymbol{A}^*\boldsymbol{A}\| = \|\boldsymbol{A}\boldsymbol{A}^*\|$$

hold for each matrix $\boldsymbol{A}$.

We say that a matrix $\boldsymbol{U}$ is *orthonormal* if its columns form an orthonormal set with respect to the Hermitian inner product. An orthonormal matrix $\boldsymbol{U}$ preserves geometry in the sense that $\|\boldsymbol{U}\boldsymbol{x}\| = \|\boldsymbol{x}\|$ for every vector $\boldsymbol{x}$. A *unitary* matrix is a square orthonormal matrix, and an *orthogonal* matrix is a real unitary matrix. Unitary matrices satisfy the relations $\boldsymbol{U}\boldsymbol{U}^* = \boldsymbol{U}^*\boldsymbol{U} = \mathbf{I}$. Both the operator norm and the Frobenius norm are *unitarily invariant*, which means that

$$\|\boldsymbol{U}\boldsymbol{A}\boldsymbol{V}^*\| = \|\boldsymbol{A}\| \qquad \text{and} \qquad \|\boldsymbol{U}\boldsymbol{A}\boldsymbol{V}^*\|_{\mathrm{F}} = \|\boldsymbol{A}\|_{\mathrm{F}}$$

for every matrix $\boldsymbol{A}$ and all orthonormal matrices $\boldsymbol{U}$ and $\boldsymbol{V}$.

We use the notation of [61] to denote submatrices. If $\boldsymbol{A}$ is an $m \times n$ matrix with entries a_{ij}, and if $I = [i_1, i_2, \ldots, i_p]$ and $J = [j_1, j_2, \ldots, j_q]$ are two index vectors, then the associated $p \times q$ submatrix is expressed as

$$\boldsymbol{A}_{(I,J)} = \begin{bmatrix} a_{i_1,j_1} & \cdots & a_{i_1,j_q} \\ \vdots & & \vdots \\ a_{i_p,j_1} & \cdots & a_{i_p,j_q} \end{bmatrix}.$$

For column and row submatrices, we use the standard abbreviations

$$\boldsymbol{A}_{(:\,,J)} = \boldsymbol{A}_{([1,2,\ldots,m],J)} \qquad \text{and} \qquad \boldsymbol{A}_{(I,:\,)} = \boldsymbol{A}_{(I,[1,2,\ldots,n])}.$$

3.2. Standard Matrix Factorizations. This section defines three basic matrix decompositions. Methods for computing them are described in section 3.3.

3.2.1. The Pivoted QR Factorization. Each $m \times n$ matrix $\boldsymbol{A}$ of rank k admits a decomposition

$$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{R},$$

where $\boldsymbol{Q}$ is an $m \times k$ orthonormal matrix, and $\boldsymbol{R}$ is a $k \times n$ *weakly upper-triangular* matrix. That is, there exists a permutation J of the tuple $(1, 2, \ldots, n)$ such that $\boldsymbol{R}_{(:\,,J)}$ is upper triangular. Moreover, the diagonal entries of $\boldsymbol{R}_{(:\,,J)}$ are weakly decreasing. See [61, sect. 5.4.1] for details.

3.2.2. The Singular Value Decomposition (SVD). Each $m \times n$ matrix $\boldsymbol{A}$ of rank k admits a factorization

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^*,$$

where $\boldsymbol{U}$ is an $m \times k$ orthonormal matrix, $\boldsymbol{V}$ is an $n \times k$ orthonormal matrix, and $\boldsymbol{\Sigma}$ is a $k \times k$ nonnegative, diagonal matrix

$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_k \end{bmatrix}.$$

The numbers σ_j are called the *singular values* of $\boldsymbol{A}$. They are arranged in weakly decreasing order:

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k \geq 0.$$

The columns of $\boldsymbol{U}$ and $\boldsymbol{V}$ are called *left singular vectors* and *right singular vectors*, respectively.

Singular values are connected with the approximability of matrices. For each j, the number σ_{j+1} equals the spectral-norm discrepancy between $\boldsymbol{A}$ and an optimal rank-j approximation [98]. That is,

$$\sigma_{j+1} = \min\{\|\boldsymbol{A} - \boldsymbol{B}\| \,:\, \boldsymbol{B} \text{ has rank } j\}. \tag{3.1}$$

In particular, $\sigma_1 = \|\boldsymbol{A}\|$. See [61, sects. 2.5.3 and 5.4.5] for additional details.

3.2.3. The Interpolative Decomposition (ID). Our final factorization identifies a collection of k columns from a rank-k matrix $\boldsymbol{A}$ that span the range of $\boldsymbol{A}$. To be precise, we can compute an index set $J = [j_1, \ldots, j_k]$ such that

$$\boldsymbol{A} = \boldsymbol{A}_{(:\,,J)}\, \boldsymbol{X},$$

where $\boldsymbol{X}$ is a $k \times n$ matrix that satisfies $\boldsymbol{X}_{(:\,,J)} = \mathbf{I}_k$. Furthermore, no entry of $\boldsymbol{X}$ has magnitude larger than two. In other words, this decomposition expresses each column of $\boldsymbol{A}$ using a linear combination of k fixed columns with *bounded* coefficients. Stable and efficient algorithms for computing the ID appear in the papers [26, 68].

It is also possible to compute a two-sided ID,

$$\boldsymbol{A} = \boldsymbol{W}\, \boldsymbol{A}_{(J',J)}\, \boldsymbol{X},$$

where J' is an index set identifying k of the rows of $\boldsymbol{A}$, and $\boldsymbol{W}$ is an $m \times k$ matrix that satisfies $\boldsymbol{W}_{(J',:\,)} = \mathbf{I}_k$ and whose entries are all bounded by two.

Remark 3.1. There always exists an ID where the entries in the factor $\boldsymbol{X}$ have magnitude bounded by one. Known proofs of this fact are constructive, e.g., [104, Lem. 3.3], but they require us to find a collection of k columns that has "maximum volume." It is NP-hard to identify a subset of columns with this type of extremal property [27]. We find it remarkable that ID computations are possible as soon as the bound on $\boldsymbol{X}$ is relaxed.

3.3. Techniques for Computing Standard Factorizations. This section discusses some established deterministic techniques for computing the factorizations presented in section 3.2. The material on pivoted QR and SVD can be located in any major text on numerical linear algebra, such as [61, 133]. References for the ID include [26, 68].

3.3.1. Computing the Full Decomposition. It is possible to compute the full QR factorization or the full SVD of an $m \times n$ matrix to double-precision accuracy with $\mathrm{O}(mn \min\{m, n\})$ flops. Techniques for computing the SVD are iterative by necessity, but they converge so fast that we can treat them as finite for practical purposes.

3.3.2. Computing Partial Decompositions. Suppose that an $m \times n$ matrix has numerical rank k, where k is substantially smaller than m and n. In this case, it is possible to produce a structured low-rank decomposition that approximates the matrix well. Sections 4 and 5 describe a set of randomized techniques for obtaining these partial decompositions. This section briefly reviews the classical techniques, which also play a role in developing randomized methods.

To compute a partial QR decomposition, the classical device is the Businger–Golub algorithm, which performs successive orthogonalization with pivoting on the columns of the matrix. The procedure halts when the Frobenius norm of the remaining

columns is less than a computational tolerance ε. Letting ℓ denote the number of steps required, the process results in a partial factorization

$$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{R} + \boldsymbol{E}, \tag{3.2}$$

where $\boldsymbol{Q}$ is an $m \times \ell$ orthonormal matrix, $\boldsymbol{R}$ is an $\ell \times n$ weakly upper-triangular matrix, and $\boldsymbol{E}$ is a residual that satisfies $\|\boldsymbol{E}\|_{\mathrm{F}} \leq \varepsilon$. The computational cost is $\mathrm{O}(\ell mn)$, and the number ℓ of steps taken is typically close to the minimal rank k for which precision ε (in the Frobenius norm) is achievable. The Businger–Golub algorithm can in principle significantly overpredict the rank, but in practice this problem is very rare, provided that orthonormality is maintained scrupulously.

Subsequent research has led to strong rank-revealing QR algorithms that succeed for all matrices. For example, the Gu–Eisenstat algorithm [68] (setting their parameter $f = 2$) produces a QR decomposition of the form (3.2), where

$$\|\boldsymbol{E}\| \leq \sqrt{1 + 4k(n-k)} \cdot \sigma_{k+1}.$$

Recall that σ_{k+1} is the minimal error possible in a rank-k approximation [98]. The cost of the Gu–Eisenstat algorithm is typically close to $\mathrm{O}(kmn)$, but it can be higher in rare cases. The algorithm can also be used to obtain an approximate ID [26].

To compute an approximate SVD of a general $m \times n$ matrix, the most straightforward technique is to compute the full SVD and truncate it. This procedure is stable and accurate, but it requires $\mathrm{O}(mn \min\{m, n\})$ flops. A more efficient approach is to compute a partial QR factorization and postprocess the factors to obtain a partial SVD using the methods described below in section 3.3.3. This scheme takes only $\mathrm{O}(kmn)$ flops. Krylov subspace methods can also compute partial SVDs at a comparable cost of $\mathrm{O}(kmn)$, but they are less robust.

Note that all the techniques described in this section require extensive random access to the matrix, and they can be very slow when the matrix is stored out-of-core.

3.3.3. Converting from One Partial Factorization to Another. Suppose that we have obtained a partial decomposition of a matrix $\boldsymbol{A}$ by some means:

$$\|\boldsymbol{A} - \boldsymbol{C}\boldsymbol{B}\| \leq \varepsilon,$$

where $\boldsymbol{B}$ and $\boldsymbol{C}$ have rank k. Given this information, we can efficiently compute any of the basic factorizations.

We construct a partial QR factorization using the following three steps:

1. Compute a QR factorization of $\boldsymbol{C}$ so that $\boldsymbol{C} = \boldsymbol{Q}_1\boldsymbol{R}_1$.
2. Form the product $\boldsymbol{D} = \boldsymbol{R}_1\boldsymbol{B}$, and compute a QR factorization: $\boldsymbol{D} = \boldsymbol{Q}_2\boldsymbol{R}$.
3. Form the product $\boldsymbol{Q} = \boldsymbol{Q}_1\boldsymbol{Q}_2$.

The result is an orthonormal matrix $\boldsymbol{Q}$ and a weakly upper-triangular matrix $\boldsymbol{R}$ such that $\|\boldsymbol{A} - \boldsymbol{Q}\boldsymbol{R}\| \leq \varepsilon$.

An analogous technique yields a partial SVD:

1. Compute a QR factorization of $\boldsymbol{C}$ so that $\boldsymbol{C} = \boldsymbol{Q}_1\boldsymbol{R}_1$.
2. Form the product $\boldsymbol{D} = \boldsymbol{R}_1\boldsymbol{B}$, and compute an SVD: $\boldsymbol{D} = \boldsymbol{U}_2\boldsymbol{\Sigma}\boldsymbol{V}^*$.
3. Form the product $\boldsymbol{U} = \boldsymbol{Q}_1\boldsymbol{U}_2$.

The result is a diagonal matrix $\boldsymbol{\Sigma}$ and orthonormal matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ such that $\|\boldsymbol{A} - \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^*\| \leq \varepsilon$.

Converting $\boldsymbol{B}$ and $\boldsymbol{C}$ into a partial ID is a one-step process:

1. Compute J and $\boldsymbol{X}$ such that $\boldsymbol{B} = \boldsymbol{B}_{(:\,,J)}\boldsymbol{X}$.

Then $\boldsymbol{A} \approx \boldsymbol{A}_{(:\,,J)}\boldsymbol{X}$, but the approximation error may deteriorate from the initial estimate. For example, if we compute the ID using the Gu–Eisenstat algorithm [68] with the parameter $f = 2$, then we get the error $\left\|\boldsymbol{A} - \boldsymbol{A}_{(:\,,J)}\boldsymbol{X}\right\| \leq (1+\sqrt{1+4k(n-k)})\cdot\varepsilon$. Compare this bound with Lemma 5.1 below.

3.3.4. Krylov Subspace Methods. Suppose that the matrix $\boldsymbol{A}$ can be applied rapidly to vectors, as happens when $\boldsymbol{A}$ is sparse or structured. Then Krylov subspace techniques can very effectively and accurately compute partial spectral decompositions. For concreteness, assume that $\boldsymbol{A}$ is Hermitian. The idea of these techniques is to fix a starting vector $\boldsymbol{\omega}$ and to seek approximations to the eigenvectors within the corresponding *Krylov subspace*

$$\mathcal{V}_q(\boldsymbol{\omega}) = \operatorname{span}\{\boldsymbol{\omega}, \boldsymbol{A}\boldsymbol{\omega}, \boldsymbol{A}^2\boldsymbol{\omega}, \dots, \boldsymbol{A}^{q-1}\boldsymbol{\omega}\}.$$

Krylov methods also come in blocked versions, in which the starting *vector* $\boldsymbol{\omega}$ is replaced by a starting *matrix* $\boldsymbol{\Omega}$. A common recommendation is to draw a starting vector $\boldsymbol{\omega}$ (or starting matrix $\boldsymbol{\Omega}$) from a standardized Gaussian distribution, which indicates an overlap between Krylov methods and the methods in this paper.

The most basic versions of Krylov methods for computing spectral decompositions are numerically unstable. High-quality implementations require that we incorporate restarting strategies, techniques for maintaining high-quality bases for the Krylov subspaces, etc. The diversity and complexity of such methods make it hard to state a precise computational cost, but in the environment we consider in this paper, a typical cost for a fully stable implementation would be

$$T_{\text{Krylov}} \sim k\,T_{\text{mult}} + k^2(m+n), \tag{3.3}$$

where T_{mult} is the cost of a matrix–vector multiplication.

◇ ◇ ◇

Part II: Algorithms. This part of the paper, sections 4–7, provides detailed descriptions of randomized algorithms for constructing low-rank approximations to matrices. As discussed in section 1.2, we split the problem into two stages. In Stage A, we construct a subspace that captures the action of the input matrix. In Stage B, we use this subspace to obtain an approximate factorization of the matrix.

Section 4 develops randomized methods for completing Stage A, and section 5 describes deterministic methods for Stage B. Section 6 compares the computational costs of the resulting two-stage algorithm with the classical approaches outlined in section 3. Finally, section 7 illustrates the performance of the randomized schemes via numerical examples.

4. Stage A: Randomized Schemes for Approximating the Range. This section outlines techniques for constructing a subspace that captures most of the action of a matrix. We begin with a recapitulation of the proto-algorithm that we introduced in section 1.3. We discuss how it can be implemented in practice (section 4.1) and then consider the question of how many random samples to acquire (section 4.2). Afterward, we present several ways in which the basic scheme can be improved. Sections 4.3 and 4.4 explain how to address the situation where the numerical rank of the input matrix is not known in advance. Section 4.5 shows how to modify the scheme to improve its accuracy when the singular spectrum of the input matrix decays slowly. Finally, section 4.6 describes how the scheme can be accelerated by using a structured random matrix.

ALGORITHM 4.1: RANDOMIZED RANGE FINDER

Given an $m \times n$ matrix $\boldsymbol{A}$ and an integer ℓ, this scheme computes an $m \times \ell$ orthonormal matrix $\boldsymbol{Q}$ whose range approximates the range of $\boldsymbol{A}$.

1 Draw an $n \times \ell$ Gaussian random matrix $\boldsymbol{\Omega}$.
2 Form the $m \times \ell$ matrix $\boldsymbol{Y} = \boldsymbol{A\Omega}$.
3 Construct an $m \times \ell$ matrix $\boldsymbol{Q}$ whose columns form an orthonormal basis for the range of $\boldsymbol{Y}$, e.g., using the QR factorization $\boldsymbol{Y} = \boldsymbol{QR}$.

4.1. The Proto-algorithm Revisited. The most natural way to implement the proto-algorithm from section 1.3 is to draw a random test matrix $\boldsymbol{\Omega}$ from the standard Gaussian distribution. That is, each entry of $\boldsymbol{\Omega}$ is an independent Gaussian random variable with mean zero and variance one. For reference, we formulate the resulting scheme as Algorithm 4.1.

The number T_{basic} of flops required by Algorithm 4.1 satisfies

$$T_{\text{basic}} \sim \ell n\, T_{\text{rand}} + \ell\, T_{\text{mult}} + \ell^2 m, \tag{4.1}$$

where T_{rand} is the cost of generating a Gaussian random number and T_{mult} is the cost of multiplying $\boldsymbol{A}$ by a vector. The three terms in (4.1) correspond directly with the three steps of Algorithm 4.1.

Empirically, we have found that the performance of Algorithm 4.1 depends very little on the quality of the random number generator used in step 1.

The actual cost of step 2 depends substantially on the matrix $\boldsymbol{A}$ and the computational environment that we are working in. The estimate (4.1) suggests that Algorithm 4.1 is especially efficient when the matrix–vector product $\boldsymbol{x} \mapsto \boldsymbol{Ax}$ can be evaluated rapidly. In particular, the scheme is appropriate for approximating sparse or structured matrices. Turn to section 6 for more details.

The most important implementation issue arises when performing the basis calculation in step 3. Typically, the columns of the sample matrix $\boldsymbol{Y}$ are almost linearly dependent, so it is imperative to use stable methods for performing the orthonormalization. We have found that the Gram–Schmidt procedure, augmented with the *double orthogonalization* described in [12], is both convenient and reliable. Methods based on Householder reflectors or Givens rotations also work very well. Note that very little is gained by pivoting because the columns of the random matrix $\boldsymbol{Y}$ are independent samples drawn from the same distribution.

4.2. The Number of Samples Required. The goal of Algorithm 4.1 is to produce an orthonormal matrix $\boldsymbol{Q}$ with few columns that achieves

$$\|(\mathbf{I} - \boldsymbol{QQ}^*)\boldsymbol{A}\| \leq \varepsilon, \tag{4.2}$$

where ε is a specified tolerance. The number of columns ℓ that the algorithm needs to reach this threshold is usually slightly larger than the rank k of the smallest basis that verifies (4.2). We refer to this discrepancy $p = \ell - k$ as the *oversampling parameter*. The size of the oversampling parameter depends on several factors:

The Matrix Dimensions. Very large matrices may require more oversampling.

The Singular Spectrum. The more rapid the decay of the singular values, the less oversampling is needed. In the extreme case that the matrix has exact rank k, it is not necessary to oversample.

The Random Test Matrix. Gaussian matrices succeed with very little oversampling, but are not always the most cost-effective option. The structured random matrices discussed in section 4.6 may require substantial oversampling, but they still yield computational gains in certain settings.

The theoretical results in Part III provide detailed information about how the behavior of randomized schemes depends on these factors. For the moment, we limit ourselves to some general remarks on implementation issues.

For Gaussian test matrices, it is adequate to choose the oversampling parameter to be a small constant, such as $p = 5$ or $p = 10$. There is rarely any advantage to select $p > k$. This observation, first presented in [92], demonstrates that a Gaussian test matrix results in a negligible amount of extra computation.

In practice, the target rank k is rarely known in advance. Randomized algorithms are usually implemented in an adaptive fashion where the number of samples is increased until the error satisfies the desired tolerance. In other words, the user never *chooses* the oversampling parameter. Theoretical results that bound the amount of oversampling are valuable primarily as aids for designing algorithms. We develop an adaptive approach in sections 4.3–4.4.

The computational bottleneck in Algorithm 4.1 is usually the formation of the product $\boldsymbol{A\Omega}$. As a result, it often pays to draw a larger number ℓ of samples than necessary because the user can minimize the cost of the matrix multiplication with tools such as blocking of operations, high-level linear algebra subroutines, parallel processors, etc. This approach may lead to an ill-conditioned sample matrix $\boldsymbol{Y}$, but the orthogonalization in step 3 of Algorithm 4.1 can easily identify the numerical rank of the sample matrix and ignore the excess samples. Furthermore, Stage B of the matrix approximation process succeeds even when the basis matrix $\boldsymbol{Q}$ has a larger dimension than necessary.

4.3. A Posteriori Error Estimation. Algorithm 4.1 is designed for solving the fixed-rank problem, where the target rank of the input matrix is specified in advance. To handle the fixed-precision problem, where the parameter is the computational tolerance, we need a scheme for estimating how well a putative basis matrix $\boldsymbol{Q}$ captures the action of the matrix $\boldsymbol{A}$. To do so, we develop a probabilistic error estimator. These methods are inspired by work of Dixon [39]; our treatment follows [89, 138].

The exact approximation error is $\|(\mathbf{I} - \boldsymbol{QQ}^*)\boldsymbol{A}\|$. It is intuitively plausible that we can obtain some information about this quantity by computing $\|(\mathbf{I} - \boldsymbol{QQ}^*)\boldsymbol{A\omega}\|$, where $\boldsymbol{\omega}$ is a standard Gaussian vector. This notion leads to the following method. Draw a sequence $\{\boldsymbol{\omega}^{(i)} : i = 1, 2, \ldots, r\}$ of standard Gaussian vectors, where r is a small integer that balances computational cost and reliability. Then

$$\|(\mathbf{I} - \boldsymbol{QQ}^*)\boldsymbol{A}\| \leq 10\sqrt{\frac{2}{\pi}} \max_{i=1,\ldots,r} \left\|(\mathbf{I} - \boldsymbol{QQ}^*)\boldsymbol{A\omega}^{(i)}\right\| \tag{4.3}$$

with probability at least $1 - 10^{-r}$. This statement follows by setting $\boldsymbol{B} = (\mathbf{I} - \boldsymbol{QQ}^*)\boldsymbol{A}$ and $\alpha = 10$ in the following lemma, whose proof appears in [138, sect. 3.4].

LEMMA 4.1. *Let $\boldsymbol{B}$ be a real $m \times n$ matrix. Fix a positive integer r and a real number $\alpha > 1$. Draw an independent family $\{\boldsymbol{\omega}^{(i)} : i = 1, 2, \ldots, r\}$ of standard Gaussian vectors. Then*

$$\|\boldsymbol{B}\| \leq \alpha\sqrt{\frac{2}{\pi}} \max_{i=1,\ldots,r} \left\|\boldsymbol{B\omega}^{(i)}\right\|$$

except with probability α^{-r}.

The critical point is that the error estimate (4.3) is computationally inexpensive because it requires only a small number of matrix–vector products. Therefore, we can make a lowball guess for the numerical rank of $\boldsymbol{A}$ and add more samples if the error estimate is too large. The asymptotic cost of Algorithm 4.1 is preserved if we double our guess for the rank at each step. For example, we can start with 32 samples, compute another 32, then another 64, etc.

Remark 4.1. The estimate (4.3) is actually somewhat crude. We can obtain a better estimate at a similar computational cost by initializing a power iteration with a random vector and repeating the process several times [89].

4.4. Error Estimation (Almost) for Free. The error estimate described in section 4.3 can be combined with any method for constructing an approximate basis for the range of a matrix. In this section, we explain how the error estimator can be incorporated into Algorithm 4.1 at almost no additional cost.

To be precise, let us suppose that $\boldsymbol{A}$ is an $m \times n$ matrix and ε is a computational tolerance. We seek an integer ℓ and an $m \times \ell$ orthonormal matrix $\boldsymbol{Q}^{(\ell)}$ such that

$$\left\| \left(\mathbf{I} - \boldsymbol{Q}^{(\ell)} (\boldsymbol{Q}^{(\ell)})^* \right) \boldsymbol{A} \right\| \leq \varepsilon. \tag{4.4}$$

The size ℓ of the basis will typically be slightly larger than the size k of the smallest basis that achieves this error.

The basic observation behind the adaptive scheme is that we can generate the basis in step 3 of Algorithm 4.1 incrementally. Starting with an empty basis matrix $\boldsymbol{Q}^{(0)}$, the following scheme generates an orthonormal matrix whose range captures the action of $\boldsymbol{A}$:

for $i = 1, 2, 3, \ldots$
  Draw an $n \times 1$ Gaussian random vector $\boldsymbol{\omega}^{(i)}$, and set $\boldsymbol{y}^{(i)} = \boldsymbol{A}\boldsymbol{\omega}^{(i)}$.
  Compute $\tilde{\boldsymbol{q}}^{(i)} = \left(\mathbf{I} - \boldsymbol{Q}^{(i-1)} (\boldsymbol{Q}^{(i-1)})^* \right) \boldsymbol{y}^{(i)}$.
  Normalize $\boldsymbol{q}^{(i)} = \tilde{\boldsymbol{q}}^{(i)} / \left\| \tilde{\boldsymbol{q}}^{(i)} \right\|$, and form $\boldsymbol{Q}^{(i)} = [\boldsymbol{Q}^{(i-1)} \ \boldsymbol{q}^{(i)}]$.
end for

How do we know when we have reached a basis $\boldsymbol{Q}^{(\ell)}$ that verifies (4.4)? The answer becomes apparent once we observe that the vectors $\tilde{\boldsymbol{q}}^{(i)}$ are precisely the vectors that appear in the error bound (4.3). The resulting rule is that we break the loop once we observe r consecutive vectors $\tilde{\boldsymbol{q}}^{(i)}$ whose norms are smaller than $\varepsilon / (10\sqrt{2/\pi})$.

A formal description of the resulting algorithm appears as Algorithm 4.2. A potential complication of the method is that the vectors $\tilde{\boldsymbol{q}}^{(i)}$ become small as the basis starts to capture most of the action of $\boldsymbol{A}$. In finite-precision arithmetic, their direction is extremely unreliable. To address this problem, we simply reproject the normalized vector $\boldsymbol{q}^{(i)}$ onto $\text{range}(\boldsymbol{Q}^{(i-1)})^{\perp}$ in steps 7 and 8 of Algorithm 4.2.

The CPU time requirements of Algorithms 4.2 and 4.1 are essentially identical. Although Algorithm 4.2 computes the last few samples purely to obtain the error estimate, this apparent extra cost is offset by the fact that Algorithm 4.1 always includes an oversampling factor. The failure probability stated for Algorithm 4.2 is pessimistic because it is derived from a simple union bound argument. In practice, the error estimator is reliable in a range of circumstances when we take $r = 10$.

Remark 4.2. The calculations in Algorithm 4.2 can be organized so that each iteration processes a block of samples simultaneously. This revision can lead to dramatic improvements in speed because it allows us to exploit higher-level linear algebra subroutines (e.g., BLAS3) or parallel processors. Although blocking can lead to the

ALGORITHM 4.2: ADAPTIVE RANDOMIZED RANGE FINDER

Given an $m \times n$ matrix $\boldsymbol{A}$, a tolerance ε, and an integer r (e.g., $r = 10$), the following scheme computes an orthonormal matrix $\boldsymbol{Q}$ such that (4.2) holds with probability at least $1 - \min\{m, n\}10^{-r}$.

1 Draw standard Gaussian vectors $\boldsymbol{\omega}^{(1)}, \ldots, \boldsymbol{\omega}^{(r)}$ of length n.
2 For $i = 1, 2, \ldots, r$, compute $\boldsymbol{y}^{(i)} = \boldsymbol{A}\boldsymbol{\omega}^{(i)}$.
3 $j = 0$.
4 $\boldsymbol{Q}^{(0)} = [\,]$, the $m \times 0$ empty matrix.
5 **while** $\max\left\{ \left\|\boldsymbol{y}^{(j+1)}\right\|, \left\|\boldsymbol{y}^{(j+2)}\right\|, \ldots, \left\|\boldsymbol{y}^{(j+r)}\right\| \right\} > \varepsilon/(10\sqrt{2/\pi})$,
6 $j = j + 1$.
7 Overwrite $\boldsymbol{y}^{(j)}$ by $\left(\mathbf{I} - \boldsymbol{Q}^{(j-1)}(\boldsymbol{Q}^{(j-1)})^*\right)\boldsymbol{y}^{(j)}$.
8 $\boldsymbol{q}^{(j)} = \boldsymbol{y}^{(j)} / \left\|\boldsymbol{y}^{(j)}\right\|$.
9 $\boldsymbol{Q}^{(j)} = [\boldsymbol{Q}^{(j-1)}\ \boldsymbol{q}^{(j)}]$.
10 Draw a standard Gaussian vector $\boldsymbol{\omega}^{(j+r)}$ of length n.
11 $\boldsymbol{y}^{(j+r)} = \left(\mathbf{I} - \boldsymbol{Q}^{(j)}(\boldsymbol{Q}^{(j)})^*\right)\boldsymbol{A}\boldsymbol{\omega}^{(j+r)}$.
12 **for** $i = (j+1), (j+2), \ldots, (j+r-1)$,
13 Overwrite $\boldsymbol{y}^{(i)}$ by $\boldsymbol{y}^{(i)} - \boldsymbol{q}^{(j)}\left\langle \boldsymbol{q}^{(j)},\ \boldsymbol{y}^{(i)} \right\rangle$.
14 **end for**
15 **end while**
16 $\boldsymbol{Q} = \boldsymbol{Q}^{(j)}$.

generation of unnecessary samples, this outcome is generally harmless, as noted in section 4.2.

4.5. A Modified Scheme for Matrices Whose Singular Values Decay Slowly. The techniques described in sections 4.1 and 4.4 work well for matrices whose singular values exhibit some decay, but they may produce a poor basis when the input matrix has a flat singular spectrum or when the input matrix is very large. In this section, we describe techniques, originally proposed in [67, 113], for improving the accuracy of randomized algorithms in these situations. Related earlier work includes [115] and the literature on classical orthogonal iteration methods [61, p. 332].

The intuition behind these techniques is that the singular vectors associated with small singular values interfere with the calculation, so we reduce their weight relative to the dominant singular vectors by taking powers of the matrix to be analyzed. More precisely, we wish to apply the randomized sampling scheme to the matrix $\boldsymbol{B} = (\boldsymbol{A}\boldsymbol{A}^*)^q\boldsymbol{A}$, where q is a small integer. The matrix $\boldsymbol{B}$ has the same singular vectors as the input matrix $\boldsymbol{A}$, but its singular values decay much more quickly:

$$\sigma_j(\boldsymbol{B}) = \sigma_j(\boldsymbol{A})^{2q+1}, \quad j = 1, 2, 3, \ldots. \tag{4.5}$$

We modify Algorithm 4.1 by replacing the formula $\boldsymbol{Y} = \boldsymbol{A}\boldsymbol{\Omega}$ in step 2 by the formula $\boldsymbol{Y} = \boldsymbol{B}\boldsymbol{\Omega} = \left(\boldsymbol{A}\boldsymbol{A}^*\right)^q\boldsymbol{A}\boldsymbol{\Omega}$, and we obtain Algorithm 4.3.

Algorithm 4.3 requires $2q + 1$ times as many matrix–vector multiplies as Algorithm 4.1, but is far more accurate in situations where the singular values of $\boldsymbol{A}$ decay slowly. A good heuristic is that when the original scheme produces a basis whose approximation error is within a factor C of the optimum, the power scheme produces an approximation error within $C^{1/(2q+1)}$ of the optimum. In other words, the power iteration drives the approximation gap to one exponentially fast. See Theorem 9.2 and section 10.4 for the details.

ALGORITHM 4.3: RANDOMIZED POWER ITERATION

Given an $m \times n$ matrix $\boldsymbol{A}$ and integers ℓ and q, this algorithm computes an $m \times \ell$ orthonormal matrix $\boldsymbol{Q}$ whose range approximates the range of $\boldsymbol{A}$.

1 Draw an $n \times \ell$ Gaussian random matrix $\boldsymbol{\Omega}$.
2 Form the $m \times \ell$ matrix $\boldsymbol{Y} = (\boldsymbol{A}\boldsymbol{A}^*)^q \boldsymbol{A}\boldsymbol{\Omega}$ via alternating application of $\boldsymbol{A}$ and $\boldsymbol{A}^*$.
3 Construct an $m \times \ell$ matrix $\boldsymbol{Q}$ whose columns form an orthonormal basis for the range of $\boldsymbol{Y}$, e.g., via the QR factorization $\boldsymbol{Y} = \boldsymbol{Q}\boldsymbol{R}$.

Note: This procedure is vulnerable to round-off errors; see Remark 4.3. The recommended implementation appears as Algorithm 4.4.

ALGORITHM 4.4: RANDOMIZED SUBSPACE ITERATION

Given an $m \times n$ matrix $\boldsymbol{A}$ and integers ℓ and q, this algorithm computes an $m \times \ell$ orthonormal matrix $\boldsymbol{Q}$ whose range approximates the range of $\boldsymbol{A}$.

1 Draw an $n \times \ell$ standard Gaussian matrix $\boldsymbol{\Omega}$.
2 Form $\boldsymbol{Y}_0 = \boldsymbol{A}\boldsymbol{\Omega}$ and compute its QR factorization $\boldsymbol{Y}_0 = \boldsymbol{Q}_0\boldsymbol{R}_0$.
3 **for** $j = 1, 2, \ldots, q$
4 Form $\widetilde{\boldsymbol{Y}}_j = \boldsymbol{A}^*\boldsymbol{Q}_{j-1}$ and compute its QR factorization $\widetilde{\boldsymbol{Y}}_j = \widetilde{\boldsymbol{Q}}_j\widetilde{\boldsymbol{R}}_j$.
5 Form $\boldsymbol{Y}_j = \boldsymbol{A}\widetilde{\boldsymbol{Q}}_j$ and compute its QR factorization $\boldsymbol{Y}_j = \boldsymbol{Q}_j\boldsymbol{R}_j$.
6 **end**
7 $\boldsymbol{Q} = \boldsymbol{Q}_q$.

Algorithm 4.3 targets the fixed-rank problem. To address the fixed-precision problem, we can incorporate the error estimators described in section 4.3 to obtain an adaptive scheme analogous with Algorithm 4.2. In situations where it is critical to achieve near-optimal approximation errors, one can increase the oversampling beyond our standard recommendation $\ell = k + 5$ all the way to $\ell = 2k$ without changing the scaling of the asymptotic computational cost. A supporting analysis appears in Corollary 10.10.

Remark 4.3. Unfortunately, when Algorithm 4.3 is executed in floating-point arithmetic, rounding errors will extinguish all information pertaining to singular modes associated with singular values that are small compared with $\|\boldsymbol{A}\|$. (Roughly, if machine precision is μ, then all information associated with singular values smaller than $\mu^{1/(2q+1)}\,\|\boldsymbol{A}\|$ is lost.) This problem can easily be remedied by orthonormalizing the columns of the sample matrix between each application of $\boldsymbol{A}$ and $\boldsymbol{A}^*$. The resulting scheme, summarized as Algorithm 4.4, is algebraically equivalent to Algorithm 4.3 when executed in exact arithmetic [93, 125]. We recommend Algorithm 4.4 because its computational costs are similar to those of Algorithm 4.3, even though the former is substantially more accurate in floating-point arithmetic.

4.6. An Accelerated Technique for General Dense Matrices. This section describes a set of techniques that allow us to compute an approximate rank-ℓ factorization of a general dense $m \times n$ matrix in roughly $\mathrm{O}(mn\log(\ell))$ flops, in contrast to the asymptotic cost $\mathrm{O}(mn\ell)$ required by earlier methods. We can tailor this scheme for the real or complex case, but we focus on the conceptually simpler complex case. These algorithms were introduced in [138]; similar techniques were proposed in [119].

The first step toward this accelerated technique is to observe that the bottleneck in Algorithm 4.1 is the computation of the matrix product $\boldsymbol{A}\boldsymbol{\Omega}$. When the test matrix

ALGORITHM 4.5: FAST RANDOMIZED RANGE FINDER

Given an $m \times n$ matrix $\boldsymbol{A}$ and an integer ℓ, this scheme computes an $m \times \ell$ orthonormal matrix $\boldsymbol{Q}$ whose range approximates the range of $\boldsymbol{A}$.

1 Draw an $n \times \ell$ SRFT test matrix $\boldsymbol{\Omega}$, as defined by (4.6).
2 Form the $m \times \ell$ matrix $\boldsymbol{Y} = \boldsymbol{A\Omega}$ using a (subsampled) FFT.
3 Construct an $m \times \ell$ matrix $\boldsymbol{Q}$ whose columns form an orthonormal basis for the range of $\boldsymbol{Y}$, e.g., using the QR factorization $\boldsymbol{Y} = \boldsymbol{QR}$.

$\boldsymbol{\Omega}$ is standard Gaussian, the cost of this multiplication is $\mathrm{O}(mn\ell)$, the same as a rank-revealing QR algorithm [68]. The key idea is to use a *structured* random matrix that allows us to compute the product in $\mathrm{O}(mn\log(\ell))$ flops.

The *subsampled random Fourier transform*, or SRFT, is perhaps the simplest example of a structured random matrix that meets our goals. An SRFT is an $n \times \ell$ matrix of the form

$$\boldsymbol{\Omega} = \sqrt{\frac{n}{\ell}}\, \boldsymbol{DFR}, \tag{4.6}$$

where

- $\boldsymbol{D}$ is an $n \times n$ diagonal matrix whose entries are independent random variables uniformly distributed on the complex unit circle,
- $\boldsymbol{F}$ is the $n \times n$ unitary discrete Fourier transform (DFT), whose entries take the values $f_{pq} = n^{-1/2}\, \mathrm{e}^{-2\pi \mathrm{i}(p-1)(q-1)/n}$ for $p, q = 1, 2, \ldots, n$, and
- $\boldsymbol{R}$ is an $n \times \ell$ matrix that samples ℓ coordinates from n uniformly at random; i.e., its ℓ columns are drawn randomly without replacement from the columns of the $n \times n$ identity matrix.

When $\boldsymbol{\Omega}$ is defined by (4.6), we can compute the sample matrix $\boldsymbol{Y} = \boldsymbol{A\Omega}$ using $\mathrm{O}(mn\log(\ell))$ flops via a subsampled FFT [138]. Then we form the basis $\boldsymbol{Q}$ by orthonormalizing the columns of $\boldsymbol{Y}$, as described in section 4.1. This scheme appears as Algorithm 4.5. The total number T_{struct} of flops required by this procedure is

$$T_{\mathrm{struct}} \sim mn\log(\ell) + \ell^2 n. \tag{4.7}$$

Note that if ℓ is substantially larger than the numerical rank k of the input matrix, we can perform the orthogonalization with $\mathrm{O}(k\ell n)$ flops because the columns of the sample matrix are almost linearly dependent.

The test matrix (4.6) is just one choice among many possibilities. Other suggestions that appear in the literature include subsampled Hadamard transforms, chains of Givens rotations acting on randomly chosen coordinates, and many more. See [87] and its bibliography. Empirically, we have found that the transform summarized in Remark 4.6 below performs very well in a variety of environments [114].

At this point, it is not well understood how to quantify and compare the behavior of structured random transforms. One reason for this uncertainty is that it has been difficult to analyze the amount of oversampling that various transforms require. Section 11 establishes that the random matrix (4.6) can be used to identify a near-optimal basis for a rank-k matrix using $\ell \sim (k + \log(n))\log(k)$ samples. In practice, the transforms (4.6) and (4.8) typically require no more oversampling than a Gaussian test matrix requires. (For a numerical example, see section 7.4.) As a consequence, setting $\ell = k + 10$ or $\ell = k + 20$ is typically more than adequate. Further research on these questions would be valuable.

Remark 4.4. The structured random matrices discussed in this section do not adapt readily to the fixed-precision problem, where the computational tolerance is specified, because the samples from the range are usually computed in bulk. Fortunately, these schemes are sufficiently inexpensive that we can progressively increase the number of samples computed starting with $\ell = 32$, say, and then proceeding to $\ell = 64, 128, 256, \dots$ until we achieve the desired tolerance.

Remark 4.5. When using the SRFT (4.6) for matrix approximation, we have a choice whether to use a subsampled FFT or a full FFT. The complete FFT is so inexpensive that it often pays to construct an extended sample matrix $\boldsymbol{Y}_{\text{large}} = \boldsymbol{ADF}$ and then generate the actual samples by drawing columns at random from $\boldsymbol{Y}_{\text{large}}$ and rescaling as needed. The asymptotic cost increases to $\mathrm{O}(mn \log(n))$ flops, but the full FFT is actually faster for moderate problem sizes because the constant suppressed by the big-O notation is so small. Adaptive rank determination is easy because we just examine extra samples as needed.

Remark 4.6. Among the structured random matrices that we have tried, one of the strongest candidates involves sequences of random Givens rotations [114]. This matrix takes the form

$$\boldsymbol{\Omega} = \boldsymbol{D}'' \, \boldsymbol{\Theta}' \, \boldsymbol{D}' \, \boldsymbol{\Theta} \, \boldsymbol{D} \, \boldsymbol{F} \, \boldsymbol{R}, \tag{4.8}$$

where the prime symbol $'$ indicates an independent realization of a random matrix. The matrices $\boldsymbol{R}$, $\boldsymbol{F}$, and $\boldsymbol{D}$ are defined after (4.6). The matrix $\boldsymbol{\Theta}$ is a chain of random Givens rotations:

$$\boldsymbol{\Theta} = \boldsymbol{\Pi} \, \boldsymbol{G}(1, 2; \theta_1) \, \boldsymbol{G}(2, 3; \theta_2) \cdots \boldsymbol{G}(n-1, n; \theta_{n-1}),$$

where $\boldsymbol{\Pi}$ is a random $n \times n$ permutation matrix, where $\theta_1, \dots, \theta_{n-1}$ are independent random variables uniformly distributed on the interval $[0, 2\pi]$, and where $\boldsymbol{G}(i, j; \theta)$ denotes a rotation on $\mathbb{C}^n$ by the angle θ in the (i, j) coordinate plane [61, sect. 5.1.8].

Remark 4.7. When the singular values of the input matrix $\boldsymbol{A}$ decay slowly, Algorithm 4.5 may perform poorly in terms of accuracy. When randomized sampling is used with a Gaussian random matrix, the recourse is to take a couple of steps of a power iteration; see Algorithm 4.4. However, it is not currently known whether such an iterative scheme can be accelerated to $\mathrm{O}(mn \log(k))$ complexity using "fast" random transforms such as the SRFT.

5. Stage B: Construction of Standard Factorizations.

The algorithms for Stage A described in section 4 produce an orthonormal matrix $\boldsymbol{Q}$ whose range captures the action of an input matrix $\boldsymbol{A}$:

$$\|\boldsymbol{A} - \boldsymbol{QQ}^* \boldsymbol{A}\| \leq \varepsilon, \tag{5.1}$$

where ε is a computational tolerance. This section describes methods for approximating standard factorizations of $\boldsymbol{A}$ using the information in the basis $\boldsymbol{Q}$.

To accomplish this task, we pursue the idea from section 3.3.3 that any low-rank factorization $\boldsymbol{A} \approx \boldsymbol{CB}$ can be manipulated to produce a standard decomposition. When the bound (5.1) holds, the low-rank factors are simply $\boldsymbol{C} = \boldsymbol{Q}$ and $\boldsymbol{B} = \boldsymbol{Q}^* \boldsymbol{A}$. The simplest scheme (section 5.1) computes the factor $\boldsymbol{B}$ directly with a matrix–matrix product to ensure a minimal error in the final approximation. An alternative approach (section 5.2) constructs factors $\boldsymbol{B}$ and $\boldsymbol{C}$ without forming any matrix–matrix product. The approach of section 5.2 is often faster than the approach of section 5.1

ALGORITHM 5.1: DIRECT SVD

Given matrices $\boldsymbol{A}$ *and* $\boldsymbol{Q}$ *such that* (5.1) *holds, this procedure computes an approximate factorization* $\boldsymbol{A} \approx \boldsymbol{U\Sigma V}^*$*, where* $\boldsymbol{U}$ *and* $\boldsymbol{V}$ *are orthonormal, and* $\boldsymbol{\Sigma}$ *is a nonnegative diagonal matrix.*

1 Form the matrix $\boldsymbol{B} = \boldsymbol{Q}^*\boldsymbol{A}$.
2 Compute an SVD of the small matrix: $\boldsymbol{B} = \widetilde{\boldsymbol{U}}\boldsymbol{\Sigma}\boldsymbol{V}^*$.
3 Form the orthonormal matrix $\boldsymbol{U} = \boldsymbol{Q}\widetilde{\boldsymbol{U}}$.

but typically results in larger errors. Both schemes can be streamlined for an Hermitian input matrix (section 5.3) and a positive semidefinite input matrix (section 5.4). Finally, we develop single-pass algorithms that exploit other information generated in Stage A to avoid revisiting the input matrix (section 5.5).

Throughout this section, $\boldsymbol{A}$ denotes an $m \times n$ matrix, and $\boldsymbol{Q}$ is an $m \times k$ orthonormal matrix that verifies (5.1). For purposes of exposition, we concentrate on methods for constructing the partial SVD.

5.1. Factorizations Based on Forming $\boldsymbol{Q}^*\boldsymbol{A}$ Directly. The relation (5.1) implies that $\|\boldsymbol{A} - \boldsymbol{QB}\| \leq \varepsilon$, where $\boldsymbol{B} = \boldsymbol{Q}^*\boldsymbol{A}$. Once we have computed $\boldsymbol{B}$, we can produce any standard factorization using the methods of section 3.3.3. Algorithm 5.1 illustrates how to build an approximate SVD.

The factors produced by Algorithm 5.1 satisfy

$$\|\boldsymbol{A} - \boldsymbol{U\Sigma V}^*\| \leq \varepsilon. \tag{5.2}$$

In other words, the approximation error does not degrade.

The cost of Algorithm 5.1 is generally dominated by the cost of the product $\boldsymbol{Q}^*\boldsymbol{A}$ in step 1, which takes $\mathrm{O}(kmn)$ flops for a general dense matrix. Note that this scheme is particularly well suited to environments where we have a fast method for computing the matrix–vector product $\boldsymbol{x} \mapsto \boldsymbol{A}^*\boldsymbol{x}$—for example, when $\boldsymbol{A}$ is sparse or structured. This approach retains a strong advantage over Krylov subspace methods and rank-revealing QR because step 1 can be accelerated using BLAS3, parallel processors, and so forth. Steps 2 and 3 require $\mathrm{O}(k^2n)$ and $\mathrm{O}(k^2m)$ flops, respectively.

Remark 5.1. Algorithm 5.1 produces an approximate SVD with the same rank as the basis matrix $\boldsymbol{Q}$. When the size of the basis exceeds the desired rank k of the SVD, it may be preferable to retain only the dominant k singular values and singular vectors. Equivalently, we replace the diagonal matrix $\boldsymbol{\Sigma}$ of computed singular values with the matrix $\boldsymbol{\Sigma}_{(k)}$ formed by zeroing out all but the largest k entries of $\boldsymbol{\Sigma}$. In the worst case, this truncation step can increase the approximation error by σ_{k+1}; see section 9.4 for an analysis. Our numerical experience suggests that this error analysis is pessimistic, and the term σ_{k+1} often does not appear in practice.

5.2. Postprocessing via Row Extraction. Given a matrix $\boldsymbol{Q}$ such that (5.1) holds, we can obtain a rank-k factorization

$$\boldsymbol{A} \approx \boldsymbol{XB}, \tag{5.3}$$

where $\boldsymbol{B}$ is a $k \times n$ matrix consisting of k rows extracted from $\boldsymbol{A}$. The approximation (5.3) can be produced without computing any matrix–matrix products, which makes this approach to postprocessing very fast. The drawback comes because the error $\|\boldsymbol{A} - \boldsymbol{XB}\|$ is usually larger than the initial error $\|\boldsymbol{A} - \boldsymbol{QQ}^*\boldsymbol{A}\|$, especially when the dimensions of $\boldsymbol{A}$ are large. See Remark 5.3 for more discussion.

ALGORITHM 5.2: SVD VIA ROW EXTRACTION

Given matrices $\boldsymbol{A}$ *and* $\boldsymbol{Q}$ *such that* (5.1) *holds, this procedure computes an approximate factorization* $\boldsymbol{A} \approx \boldsymbol{U\Sigma V}^*$*, where* $\boldsymbol{U}$ *and* $\boldsymbol{V}$ *are orthonormal, and* $\boldsymbol{\Sigma}$ *is a nonnegative diagonal matrix.*

1 Compute an ID $\boldsymbol{Q} = \boldsymbol{XQ}_{(J,:)}$. (The ID is defined in section 3.2.3.)
2 Extract $\boldsymbol{A}_{(J,:)}$, and compute a QR factorization $\boldsymbol{A}_{(J,:)} = \boldsymbol{R}^*\boldsymbol{W}^*$.
3 Form the product $\boldsymbol{Z} = \boldsymbol{XR}^*$.
4 Compute an SVD $\boldsymbol{Z} = \boldsymbol{U\Sigma}\widetilde{\boldsymbol{V}}^*$.
5 Form the orthonormal matrix $\boldsymbol{V} = \boldsymbol{W}\widetilde{\boldsymbol{V}}$.

Note: Algorithm 5.2 is faster than Algorithm 5.1 but less accurate.

Note: It is advantageous to replace the basis $\boldsymbol{Q}$ by the sample matrix $\boldsymbol{Y}$ produced in Stage A; cf. Remark 5.2.

To obtain the factorization (5.3), we simply construct the interpolative decomposition (section 3.2.3) of the matrix $\boldsymbol{Q}$:

$$\boldsymbol{Q} = \boldsymbol{XQ}_{(J,:)}. \tag{5.4}$$

The index set J marks k rows of $\boldsymbol{Q}$ that span the row space of $\boldsymbol{Q}$, and $\boldsymbol{X}$ is an $m \times k$ matrix whose entries are bounded in magnitude by two and contains the $k \times k$ identity as a submatrix: $\boldsymbol{X}_{(J,:)} = \mathbf{I}_k$. Combining (5.4) and (5.1), we reach

$$\boldsymbol{A} \approx \boldsymbol{QQ}^*\boldsymbol{A} = \boldsymbol{XQ}_{(J,:)}\boldsymbol{Q}^*\boldsymbol{A}. \tag{5.5}$$

Since $\boldsymbol{X}_{(J,:)} = \mathbf{I}_k$, equation (5.5) implies that $\boldsymbol{A}_{(J,:)} \approx \boldsymbol{Q}_{(J,:)}\boldsymbol{Q}^*\boldsymbol{A}$. Therefore, (5.3) follows when we put $\boldsymbol{B} = \boldsymbol{A}_{(J,:)}$.

Provided with the factorization (5.3), we can obtain any standard factorization using the techniques of section 3.3.3. Algorithm 5.2 illustrates an SVD calculation. This procedure requires $\mathrm{O}(k^2(m+n))$ flops. The following lemma guarantees the accuracy of the computed factors.

LEMMA 5.1. *Let* $\boldsymbol{A}$ *be an* $m \times n$ *matrix and let* $\boldsymbol{Q}$ *be an* $m \times k$ *matrix that satisfy* (5.1)*. Suppose that* $\boldsymbol{U}$*,* $\boldsymbol{\Sigma}$*, and* $\boldsymbol{V}$ *are the matrices constructed by Algorithm* 5.2*. Then*

$$\|\boldsymbol{A} - \boldsymbol{U\Sigma V}^*\| \leq \left[1 + \sqrt{1 + 4k(n-k)}\right]\varepsilon. \tag{5.6}$$

Proof. The factors $\boldsymbol{U}$, $\boldsymbol{\Sigma}$, $\boldsymbol{V}$ constructed by the algorithm satisfy

$$\boldsymbol{U\Sigma V}^* = \boldsymbol{U\Sigma}\widetilde{\boldsymbol{V}}^*\boldsymbol{W}^* = \boldsymbol{ZW}^* = \boldsymbol{XR}^*\boldsymbol{W}^* = \boldsymbol{XA}_{(J,:)}.$$

Define the approximation

$$\widehat{\boldsymbol{A}} = \boldsymbol{QQ}^*\boldsymbol{A}. \tag{5.7}$$

Since $\widehat{\boldsymbol{A}} = \boldsymbol{XQ}_{(J,:)}\boldsymbol{Q}^*\boldsymbol{A}$ and since $\boldsymbol{X}_{(J,:)} = \mathbf{I}_k$, it must be that $\widehat{\boldsymbol{A}}_{(J,:)} = \boldsymbol{Q}_{(J,:)}\boldsymbol{Q}^*\boldsymbol{A}$. Consequently,

$$\widehat{\boldsymbol{A}} = \boldsymbol{X}\widehat{\boldsymbol{A}}_{(J,:)}.$$

We have the chain of relations

$$\begin{aligned}
\|\boldsymbol{A} - \boldsymbol{U\Sigma V}^*\| &= \left\|\boldsymbol{A} - \boldsymbol{XA}_{(J,:)}\right\| \\
&= \left\|\left(\boldsymbol{A} - \boldsymbol{X}\widehat{\boldsymbol{A}}_{(J,:)}\right) + \left(\boldsymbol{X}\widehat{\boldsymbol{A}}_{(J,:)} - \boldsymbol{XA}_{(J,:)}\right)\right\| \\
&\leq \left\|\boldsymbol{A} - \widehat{\boldsymbol{A}}\right\| + \left\|\boldsymbol{X}\widehat{\boldsymbol{A}}_{(J,:)} - \boldsymbol{XA}_{(J,:)}\right\| \\
&\leq \left\|\boldsymbol{A} - \widehat{\boldsymbol{A}}\right\| + \|\boldsymbol{X}\|\left\|\boldsymbol{A}_{(J,:)} - \widehat{\boldsymbol{A}}_{(J,:)}\right\|. \qquad (5.8)
\end{aligned}$$

Inequality (5.1) ensures that $\|\boldsymbol{A} - \widehat{\boldsymbol{A}}\| \leq \varepsilon$. Since $\boldsymbol{A}_{(J,:)} - \widehat{\boldsymbol{A}}_{(J,:)}$ is a submatrix of $\boldsymbol{A} - \widehat{\boldsymbol{A}}$, we must also have $\|\boldsymbol{A}_{(J,:)} - \widehat{\boldsymbol{A}}_{(J,:)}\| \leq \varepsilon$. Thus, (5.8) reduces to

$$\|\boldsymbol{A} - \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^*\| \leq (1 + \|\boldsymbol{X}\|)\,\varepsilon. \tag{5.9}$$

The bound (5.6) follows from (5.9) after we observe that $\boldsymbol{X}$ contains a $k \times k$ identity matrix and that the entries of the remaining $(n-k) \times k$ submatrix are bounded in magnitude by two. ∎

Remark 5.2. To maintain a unified presentation, we have formulated all the postprocessing techniques so they take an orthonormal matrix $\boldsymbol{Q}$ as input. Recall that, in Stage A of our framework, we construct the matrix $\boldsymbol{Q}$ by orthonormalizing the columns of the sample matrix $\boldsymbol{Y}$. With finite-precision arithmetic, it is preferable to adapt Algorithm 5.2 to start directly from the sample matrix $\boldsymbol{Y}$. To be precise, we modify step 1 to compute $\boldsymbol{X}$ and J so that $\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{Y}_{(J,:)}$. This revision is recommended even when $\boldsymbol{Q}$ is available from the adaptive rank determination of Algorithm 4.2.

Remark 5.3. As inequality (5.6) suggests, the factorization produced by Algorithm 5.2 is potentially less accurate than the basis that it uses as input. This loss of accuracy is problematic when ε is not so small or when kn is large. In such cases, we recommend Algorithm 5.1 over Algorithm 5.2; the former is more costly, but it does not amplify the error, as shown in (5.2).

5.3. Postprocessing an Hermitian Matrix. When $\boldsymbol{A}$ is Hermitian, the postprocessing becomes particularly elegant. In this case, the columns of $\boldsymbol{Q}$ form a good basis for both the column space *and* the row space of $\boldsymbol{A}$ so that we have $\boldsymbol{A} \approx \boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{A}\boldsymbol{Q}\boldsymbol{Q}^*$. More precisely, when (5.1) is in force, we have

$$\begin{aligned} \|\boldsymbol{A} - \boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{A}\boldsymbol{Q}\boldsymbol{Q}^*\| &= \|\boldsymbol{A} - \boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{A} + \boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{A} - \boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{A}\boldsymbol{Q}\boldsymbol{Q}^*\| \\ &\leq \|\boldsymbol{A} - \boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{A}\| + \|\boldsymbol{Q}\boldsymbol{Q}^*(\boldsymbol{A} - \boldsymbol{A}\boldsymbol{Q}\boldsymbol{Q}^*)\| \leq 2\varepsilon. \end{aligned} \tag{5.10}$$

The last inequality relies on the facts that $\|\boldsymbol{Q}\boldsymbol{Q}^*\| = 1$ and that

$$\|\boldsymbol{A} - \boldsymbol{A}\boldsymbol{Q}\boldsymbol{Q}^*\| = \|(\boldsymbol{A} - \boldsymbol{A}\boldsymbol{Q}\boldsymbol{Q}^*)^*\| = \|\boldsymbol{A} - \boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{A}\|.$$

Since $\boldsymbol{A} \approx \boldsymbol{Q}(\boldsymbol{Q}^*\boldsymbol{A}\boldsymbol{Q})\boldsymbol{Q}^*$ is a low-rank approximation of $\boldsymbol{A}$, we can form any standard factorization using the techniques from section 3.3.3.

For Hermitian $\boldsymbol{A}$, it is more common to compute an eigenvalue decomposition than an SVD. We can accomplish this goal using Algorithm 5.3, which adapts the scheme from section 5.1. This procedure delivers a factorization that satisfies the error bound $\|\boldsymbol{A} - \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^*\| \leq 2\varepsilon$. The calculation requires $\mathrm{O}(kn^2)$ flops.

We can also pursue the row extraction approach from section 5.2, which is faster but less accurate. See Algorithm 5.4 for the details. The total cost is $\mathrm{O}(k^2 n)$ flops.

5.4. Postprocessing a Positive Semidefinite Matrix. When the input matrix $\boldsymbol{A}$ is positive semidefinite, the *Nyström method* can be used to improve the quality of standard factorizations at almost no additional cost; see [48] and its bibliography. To describe the main idea, we first recall that the direct method presented in section 5.3 manipulates the approximate rank-k factorization

$$\boldsymbol{A} \approx \boldsymbol{Q}\,(\boldsymbol{Q}^*\boldsymbol{A}\boldsymbol{Q})\,\boldsymbol{Q}^*. \tag{5.11}$$

ALGORITHM 5.3: DIRECT EIGENVALUE DECOMPOSITION

Given an Hermitian matrix $\boldsymbol{A}$ *and a basis* $\boldsymbol{Q}$ *such that* (5.1) *holds, this procedure computes an approximate eigenvalue decomposition* $\boldsymbol{A} \approx \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^*$*, where* $\boldsymbol{U}$ *is orthonormal, and* $\boldsymbol{\Lambda}$ *is a real diagonal matrix.*

1 Form the small matrix $\boldsymbol{B} = \boldsymbol{Q}^*\boldsymbol{A}\boldsymbol{Q}$.
2 Compute an eigenvalue decomposition $\boldsymbol{B} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^*$.
3 Form the orthonormal matrix $\boldsymbol{U} = \boldsymbol{Q}\boldsymbol{V}$.

ALGORITHM 5.4: EIGENVALUE DECOMPOSITION VIA ROW EXTRACTION

Given an Hermitian matrix $\boldsymbol{A}$ *and a basis* $\boldsymbol{Q}$ *such that* (5.1) *holds, this procedure computes an approximate eigenvalue decomposition* $\boldsymbol{A} \approx \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^*$*, where* $\boldsymbol{U}$ *is orthonormal, and* $\boldsymbol{\Lambda}$ *is a real diagonal matrix.*

1 Compute an ID $\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{Q}_{(J,\,:\,)}$.
2 Perform a QR factorization $\boldsymbol{X} = \boldsymbol{V}\boldsymbol{R}$.
3 Form the product $\boldsymbol{Z} = \boldsymbol{R}\boldsymbol{A}_{(J,J)}\boldsymbol{R}^*$.
4 Compute an eigenvalue decomposition $\boldsymbol{Z} = \boldsymbol{W}\boldsymbol{\Lambda}\boldsymbol{W}^*$.
5 Form the orthonormal matrix $\boldsymbol{U} = \boldsymbol{V}\boldsymbol{W}$.

Note: Algorithm 5.4 is faster than Algorithm 5.3 but less accurate.

Note: It is advantageous to replace the basis $\boldsymbol{Q}$ by the sample matrix $\boldsymbol{Y}$ produced in Stage A; cf. Remark 5.2.

In contrast, the Nyström scheme builds a more sophisticated rank-k approximation, namely,

$$\begin{aligned}\boldsymbol{A} &\approx (\boldsymbol{A}\boldsymbol{Q})\left(\boldsymbol{Q}^*\boldsymbol{A}\boldsymbol{Q}\right)^{-1}(\boldsymbol{A}\boldsymbol{Q})^* \\ &= \left[(\boldsymbol{A}\boldsymbol{Q})\left(\boldsymbol{Q}^*\boldsymbol{A}\boldsymbol{Q}\right)^{-1/2}\right]\left[(\boldsymbol{A}\boldsymbol{Q})\left(\boldsymbol{Q}^*\boldsymbol{A}\boldsymbol{Q}\right)^{-1/2}\right]^* = \boldsymbol{F}\boldsymbol{F}^*,\end{aligned} \tag{5.12}$$

where $\boldsymbol{F}$ is an approximate Cholesky factor of $\boldsymbol{A}$ with dimension $n \times k$. To compute the factor $\boldsymbol{F}$ numerically, first form the matrices $\boldsymbol{B}_1 = \boldsymbol{A}\boldsymbol{Q}$ and $\boldsymbol{B}_2 = \boldsymbol{Q}^*\boldsymbol{B}_1$. Then decompose the positive semidefinite matrix $\boldsymbol{B}_2 = \boldsymbol{C}^*\boldsymbol{C}$ into its Cholesky factors. Finally compute the factor $\boldsymbol{F} = \boldsymbol{B}_1\boldsymbol{C}^{-1}$ by performing a triangular solve. The low-rank factorization (5.12) can be converted to a standard decomposition using the techniques from section 3.3.3.

The literature contains an explicit expression [48, Lem. 4] for the approximation error in (5.12). This result implies that, in the spectral norm, the Nyström approximation error never exceeds $\|\boldsymbol{A} - \boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{A}\|$, and it is often substantially smaller. We omit a detailed discussion.

For an example of the Nyström technique, consider Algorithm 5.5, which computes an approximate eigenvalue decomposition of a positive semidefinite matrix. This method should be compared with the scheme for Hermitian matrices, Algorithm 5.3. In both cases, the dominant cost occurs when we form $\boldsymbol{A}\boldsymbol{Q}$, so the two procedures have roughly the same running time. On the other hand, Algorithm 5.5 is typically much more accurate than Algorithm 5.3. In a sense, we are exploiting the fact that $\boldsymbol{A}$ is positive semidefinite to take one step of subspace iteration (Algorithm 4.4) for free.

5.5. Single-Pass Algorithms. The techniques described in sections 5.1–5.4 all require us to revisit the input matrix. This may not be feasible in environments

ALGORITHM 5.5: EIGENVALUE DECOMPOSITION VIA NYSTRÖM METHOD
Given a positive semidefinite matrix $\boldsymbol{A}$ and a basis $\boldsymbol{Q}$ such that (5.1) *holds, this procedure computes an approximate eigenvalue decomposition $\boldsymbol{A} \approx \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^*$, where $\boldsymbol{U}$ is orthonormal, and $\boldsymbol{\Lambda}$ is nonnegative and diagonal.*
1 Form the matrices $\boldsymbol{B}_1 = \boldsymbol{A}\boldsymbol{Q}$ and $\boldsymbol{B}_2 = \boldsymbol{Q}^*\boldsymbol{B}_1$.
2 Perform a Cholesky factorization $\boldsymbol{B}_2 = \boldsymbol{C}^*\boldsymbol{C}$.
3 Form $\boldsymbol{F} = \boldsymbol{B}_1\boldsymbol{C}^{-1}$ using a triangular solve.
4 Compute an SVD $\boldsymbol{F} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^*$ and set $\boldsymbol{\Lambda} = \boldsymbol{\Sigma}^2$.

where the matrix is too large to be stored. In this section, we develop a method that requires just one pass over the matrix to construct not only an approximate basis but also a complete factorization. Similar techniques appear in [138] and [29].

For motivation, we begin with the case where $\boldsymbol{A}$ is Hermitian. Let us recall the proto-algorithm from section 1.3.3: Draw a random test matrix $\boldsymbol{\Omega}$; form the sample matrix $\boldsymbol{Y} = \boldsymbol{A}\boldsymbol{\Omega}$; then construct a basis $\boldsymbol{Q}$ for the range of $\boldsymbol{Y}$. It turns out that the matrices $\boldsymbol{\Omega}$, $\boldsymbol{Y}$, and $\boldsymbol{Q}$ contain all the information we need to approximate $\boldsymbol{A}$.

To see why, define the (currently unknown) matrix $\boldsymbol{B}$ via $\boldsymbol{B} = \boldsymbol{Q}^*\boldsymbol{A}\boldsymbol{Q}$. Postmultiplying the definition by $\boldsymbol{Q}^*\boldsymbol{\Omega}$, we obtain the identity $\boldsymbol{B}\boldsymbol{Q}^*\boldsymbol{\Omega} = \boldsymbol{Q}^*\boldsymbol{A}\boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{\Omega}$. The relationships $\boldsymbol{A}\boldsymbol{Q}\boldsymbol{Q}^* \approx \boldsymbol{A}$ and $\boldsymbol{A}\boldsymbol{\Omega} = \boldsymbol{Y}$ show that $\boldsymbol{B}$ must satisfy

$$\boldsymbol{B}\boldsymbol{Q}^*\boldsymbol{\Omega} \approx \boldsymbol{Q}^*\boldsymbol{Y}. \tag{5.13}$$

All three matrices $\boldsymbol{\Omega}$, $\boldsymbol{Y}$, and $\boldsymbol{Q}$ are available, so we can solve (5.13) to obtain the matrix $\boldsymbol{B}$. Then the low-rank factorization $\boldsymbol{A} \approx \boldsymbol{Q}\boldsymbol{B}\boldsymbol{Q}^*$ can be converted to an eigenvalue decomposition via familiar techniques. The entire procedure requires $\mathrm{O}(k^2 n)$ flops, and it is summarized as Algorithm 5.6.

ALGORITHM 5.6: EIGENVALUE DECOMPOSITION IN ONE PASS
Given an Hermitian matrix $\boldsymbol{A}$, a random test matrix $\boldsymbol{\Omega}$, a sample matrix $\boldsymbol{Y} = \boldsymbol{A}\boldsymbol{\Omega}$, and an orthonormal matrix $\boldsymbol{Q}$ that verifies (5.1) *and $\boldsymbol{Y} = \boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{Y}$, this algorithm computes an approximate eigenvalue decomposition $\boldsymbol{A} \approx \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^*$.*
1 Use a standard least-squares solver to find an Hermitian matrix $\boldsymbol{B}_{\text{approx}}$ that approximately satisfies the equation $\boldsymbol{B}_{\text{approx}}(\boldsymbol{Q}^*\boldsymbol{\Omega}) \approx \boldsymbol{Q}^*\boldsymbol{Y}$.
2 Compute the eigenvalue decomposition $\boldsymbol{B}_{\text{approx}} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^*$.
3 Form the product $\boldsymbol{U} = \boldsymbol{Q}\boldsymbol{V}$.

When $\boldsymbol{A}$ is not Hermitian, it is still possible to devise single-pass algorithms, but we must modify the initial Stage A of the approximation framework to simultaneously construct bases for the ranges of $\boldsymbol{A}$ and $\boldsymbol{A}^*$:

1. Generate random matrices $\boldsymbol{\Omega}$ and $\widetilde{\boldsymbol{\Omega}}$.
2. Compute $\boldsymbol{Y} = \boldsymbol{A}\boldsymbol{\Omega}$ and $\widetilde{\boldsymbol{Y}} = \boldsymbol{A}^*\widetilde{\boldsymbol{\Omega}}$ in a single pass over $\boldsymbol{A}$.
3. Compute QR factorizations $\boldsymbol{Y} = \boldsymbol{Q}\boldsymbol{R}$ and $\widetilde{\boldsymbol{Y}} = \widetilde{\boldsymbol{Q}}\widetilde{\boldsymbol{R}}$.

This procedure results in matrices $\boldsymbol{Q}$ and $\widetilde{\boldsymbol{Q}}$ such that $\boldsymbol{A} \approx \boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{A}\widetilde{\boldsymbol{Q}}\widetilde{\boldsymbol{Q}}^*$. The reduced matrix we must approximate is $\boldsymbol{B} = \boldsymbol{Q}^*\boldsymbol{A}\widetilde{\boldsymbol{Q}}$. In analogy with (5.13), we find that

$$\boldsymbol{Q}^*\boldsymbol{Y} = \boldsymbol{Q}^*\boldsymbol{A}\boldsymbol{\Omega} \approx \boldsymbol{Q}^*\boldsymbol{A}\widetilde{\boldsymbol{Q}}\widetilde{\boldsymbol{Q}}^*\boldsymbol{\Omega} = \boldsymbol{B}\widetilde{\boldsymbol{Q}}^*\boldsymbol{\Omega}. \tag{5.14}$$

An analogous calculation shows that $\boldsymbol{B}$ should also satisfy

$$\widetilde{\boldsymbol{Q}}^*\widetilde{\boldsymbol{Y}} \approx \boldsymbol{B}^*\boldsymbol{Q}^*\widetilde{\boldsymbol{\Omega}}. \tag{5.15}$$

Now, the reduced matrix $\boldsymbol{B}_{\text{approx}}$ can be determined by finding a minimum-residual solution to the system of relations (5.14) and (5.15).

Remark 5.4. The single-pass approaches described in this section can degrade the approximation error in the final decomposition significantly. To explain the issue, we focus on the Hermitian case. It turns out that the coefficient matrix $\boldsymbol{Q}^*\boldsymbol{\Omega}$ in the linear system (5.13) is usually ill-conditioned. In a worst-case scenario, the error $\|\boldsymbol{A} - \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^*\|$ in the factorization produced by Algorithm 5.6 could be larger than the error resulting from the two-pass method of section 5.3 by a factor of $1/\tau_{\min}$, where $\tau_{\min}$ is the minimal singular value of the matrix $\boldsymbol{Q}^*\boldsymbol{\Omega}$.

The situation can be improved by oversampling. Suppose that we seek a rank-k approximate eigenvalue decomposition. Pick a small oversampling parameter p. Draw an $n \times (k+p)$ random matrix $\boldsymbol{\Omega}$, and form the sample matrix $\boldsymbol{Y} = \boldsymbol{A}\boldsymbol{\Omega}$. Let $\boldsymbol{Q}$ denote the $n \times k$ matrix formed by the k leading left singular vectors of $\boldsymbol{Y}$. Now, the linear system (5.13) has a coefficient matrix $\boldsymbol{Q}^*\boldsymbol{\Omega}$ of size $k \times (k+p)$, so it is overdetermined. An approximate solution of this system yields a $k \times k$ matrix $\boldsymbol{B}$.

6. Computational Costs. So far, we have postponed a detailed discussion of the computational cost of randomized matrix approximation algorithms because it is necessary to account for both the first stage, where we compute an approximate basis for the range (section 4), and the second stage, where we postprocess the basis to complete the factorization (section 5). We are now prepared to compare the cost of the two-stage scheme with the cost of traditional techniques.

Choosing an appropriate algorithm, whether classical or randomized, requires us to consider the properties of the input matrix. To draw a nuanced picture, we discuss three representative computational environments in section 6.1–6.3. We close with some comments on parallel implementations in section 6.4.

For concreteness, we focus on the problem of computing an approximate SVD of an $m \times n$ matrix $\boldsymbol{A}$ with numerical rank k. The costs for other factorizations are similar.

6.1. General Matrices That Fit in Core Memory. Suppose that $\boldsymbol{A}$ is a general matrix presented as an array of numbers that fits in core memory. In this case, the appropriate method for Stage A is to use a structured random matrix (section 4.6), which allows us to find a basis that captures the action of the matrix using $\mathrm{O}(mn\log(k) + k^2m)$ flops. For Stage B, we apply the row-extraction technique (section 5.2), which costs an additional $\mathrm{O}(k^2(m+n))$ flops. The total number of operations T_{random} for this approach satisfies

$$T_{\text{random}} \sim mn\log(k) + k^2(m+n).$$

As a rule of thumb, the approximation error of this procedure satisfies

$$\|\boldsymbol{A} - \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^*\| \lesssim n \cdot \sigma_{k+1}, \tag{6.1}$$

where σ_{k+1} is the $(k+1)$th singular value of $\boldsymbol{A}$. The estimate (6.1), which follows from Theorem 11.2 and Lemma 5.1, reflects the worst-case scenario; actual errors are usually smaller.

This algorithm should be compared with modern deterministic techniques, such as rank-revealing QR followed by postprocessing (section 3.3.2) which typically require at least

$$T_{\text{RRQR}} \sim kmn$$

operations to achieve a comparable error.

In this setting, the randomized algorithm can be several times faster than classical techniques even for problems of moderate size, say, $m, n \sim 10^3$ and $k \sim 10^2$. See section 7.4 for numerical evidence.

Remark 6.1. If row extraction is impractical, there is an alternative $\mathrm{O}(mn \log(k))$ technique described in [138, sect. 5.2]. When the error (6.1) is unacceptably large, we can use the direct method (section 5.1) for Stage B, which brings the total cost to $\mathrm{O}(kmn)$ flops.

6.2. Matrices for which Matrix–Vector Products Can Be Rapidly Evaluated. In many problems in data mining and scientific computing, the cost T_{mult} of performing the matrix–vector multiplication $\boldsymbol{x} \mapsto \boldsymbol{A}\boldsymbol{x}$ is substantially smaller than the nominal cost $\mathrm{O}(mn)$ for the dense case. It is not uncommon that $\mathrm{O}(m+n)$ flops suffice. Standard examples include (i) very sparse matrices; (ii) structured matrices, such as Töplitz operators, that can be applied using the FFT or other means; and (iii) matrices that arise from physical problems, such as discretized integral operators, that can be applied via, e.g., the fast multipole method [66].

Suppose that both $\boldsymbol{A}$ and $\boldsymbol{A}^*$ admit fast multiplies. The appropriate randomized approach for this scenario completes Stage A using Algorithm 4.1 with p constant (for the fixed-rank problem) or Algorithm 4.2 (for the fixed-precision problem) at a cost of $(k+p)\, T_{\mathrm{mult}} + \mathrm{O}(k^2 m)$ flops. For Stage B, we invoke Algorithm 5.1, which requires $(k+p)\, T_{\mathrm{mult}} + \mathrm{O}(k^2(m+n))$ flops. The total cost T_{sparse} satisfies

$$T_{\mathrm{sparse}} = 2\,(k+p)\, T_{\mathrm{mult}} + \mathrm{O}(k^2(m+n)). \tag{6.2}$$

As a rule of thumb, the approximation error of this procedure satisfies

$$\|\boldsymbol{A} - \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^*\| \lesssim \sqrt{kn} \cdot \sigma_{k+1}. \tag{6.3}$$

The estimate (6.3) follows from Corollary 10.9 and the discussion in section 5.1. Actual errors are usually smaller.

When the singular spectrum of $\boldsymbol{A}$ decays slowly, we can incorporate q iterations of the power method (Algorithm 4.3) to obtain superior solutions to the fixed-rank problem. The computational cost increases to (cf. (6.2))

$$T_{\mathrm{sparse}} = (2q+2)\,(k+p)\, T_{\mathrm{mult}} + \mathrm{O}(k^2(m+n)), \tag{6.4}$$

while the error (6.3) improves to

$$\|\boldsymbol{A} - \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^*\| \lesssim (kn)^{1/2(2q+1)} \cdot \sigma_{k+1}. \tag{6.5}$$

The estimate (6.5) takes into account the discussion in section 10.4. The power scheme can also be adapted for the fixed-precision problem (section 4.5).

In this setting, the classical prescription for obtaining a partial SVD is some variation of a Krylov subspace method; see section 3.3.4. These methods exhibit great diversity, so it is hard to specify a "typical" computational cost. To a first approximation, it is fair to say that in order to obtain an approximate SVD of rank k, the cost of a numerically stable implementation of a Krylov method is no less than the cost (6.2) with p set to zero. At this price, the Krylov method often obtains better accuracy than the basic randomized method obtained by combining Algorithms 4.1 and 5.1, especially for matrices whose singular values decay slowly. On the other hand, the randomized schemes are inherently more robust and allow much more freedom in organizing the computation to suit a particular application or a particular hardware

architecture. The latter point is in practice of crucial importance because it is usually much faster to apply a matrix to k vectors simultaneously than it is to execute k matrix–vector multiplications consecutively. In practice, blocking and parallelism can lead to enough gain that a few steps of the power method (Algorithm 4.3) can be performed more quickly than k steps of a Krylov method.

Remark 6.2. Any comparison between randomized sampling schemes and Krylov variants becomes complicated because of the fact that "basic" Krylov schemes such as Lanczos [61, p. 473] or Arnoldi [61, p. 499] are inherently unstable. To obtain numerical robustness, we must incorporate sophisticated modifications such as restarts, reorthogonalization procedures, etc. Constructing a high-quality implementation is sufficiently hard that the authors of a popular book on "numerical recipes" qualify their treatment of spectral computations as follows [110, p. 567]:

> You have probably gathered by now that the solution of eigensystems is a fairly complicated business. It is. It is one of the few subjects covered in this book for which we do *not* recommend that you avoid canned routines. On the contrary, the purpose of this chapter is precisely to give you some appreciation of what is going on inside such canned routines, so that you can make intelligent choices about using them, and intelligent diagnoses when something goes wrong.

Randomized sampling does not eliminate the difficulties referred to in this quotation; however, it reduces the task of computing a *partial* spectral decomposition of a very large matrix to the task of computing a *full* decomposition of a small dense matrix. (For example, in Algorithm 5.1, the input matrix $\boldsymbol{A}$ is large and $\boldsymbol{B}$ is small.) The latter task is much better understood and is eminently suitable for using canned routines. Random sampling schemes interact with the large matrix only through matrix–matrix products, which can easily be implemented by a user in a manner appropriate to the application and to the available hardware.

The comparison is further complicated by the fact that there is an overlap between the two sets of ideas. Algorithm 4.3 is conceptually similar to a "block Lanczos method" [61, p. 485] with a random starting matrix. Indeed, we believe that there are significant opportunities for cross-fertilization in this area. Hybrid schemes that combine the best ideas from both fields may perform very well.

6.3. General Matrices Stored in Slow Memory or Streamed. The traditional metric for numerical algorithms is the number of flops they require. When the data do not fit in fast memory, however, the computational time is often dominated by the cost of memory access. In this setting, a more appropriate measure of algorithmic performance is *pass-efficiency*, which counts how many times the data needs to be cycled through fast memory. Flop counts become largely irrelevant.

All the classical matrix factorization techniques that we discuss in section 3.2—including dense SVD, rank-revealing QR, Krylov methods, and so forth—require at least k passes over the the matrix, which is prohibitively expensive for huge data matrices. A desire to reduce the pass count of matrix approximation algorithms served as one of the early motivations for developing randomized schemes [46, 58, 106]. Detailed recent work appears in [29].

For many matrices, randomized techniques can produce an accurate approximation using just one pass over the data. For Hermitian matrices, we obtain a single-pass algorithm by combining Algorithm 4.1, which constructs an approximate basis, with Algorithm 5.6, which produces an eigenvalue decomposition without any additional access to the matrix. Section 5.5 describes the analogous technique for general matrices.

For the huge matrices that arise in applications such as data mining, it is common that the singular spectrum decays slowly. Relevant applications include image processing (see sections 7.2–7.3 for numerical examples), statistical data analysis, and network monitoring. To compute approximate factorizations in these environments, it is crucial to enhance the accuracy of the randomized approach using the power scheme, Algorithm 4.3, or some other device. This approach increases the pass count somewhat, but in our experience it is very rare that more than five passes are required.

6.4. Gains from Parallelization. As mentioned in sections 6.2–6.3, randomized methods often outperform classical techniques not because they involve fewer flops but rather because they allow us to reorganize the calculations to exploit the matrix properties and the computer architecture more fully. In addition, these methods are well suited for parallel implementation. For example, in Algorithm 4.1, the computational bottleneck is the evaluation of the matrix product $\boldsymbol{A\Omega}$, which is embarrassingly parallelizable.

7. Numerical Examples. By this time, the reader has surely formulated a pointed question: Do these randomized matrix approximation algorithms actually work in practice? In this section, we attempt to address this concern by illustrating how the algorithms perform on a diverse collection of test cases.

Section 7.1 starts with two examples from the physical sciences involving discrete approximations to operators with exponentially decaying spectra. Sections 7.2 and 7.3 continue with two examples of matrices arising in "data mining." These are large matrices whose singular spectra decay slowly; one is sparse and fits in RAM, one is dense and is stored out-of-core. Finally, section 7.4 investigates the performance of randomized methods based on structured random matrices.

Sections 7.1–7.3 focus on the algorithms for Stage A that we presented in section 4 because we wish to isolate the performance of the randomized step.

Computational examples illustrating truly large data matrices have been reported elsewhere, for instance, in [69].

7.1. Two Matrices with Rapidly Decaying Singular Values. We first illustrate the behavior of the adaptive range approximation method, Algorithm 4.2. We apply it to two matrices associated with the numerical analysis of differential and integral operators. The matrices in question have rapidly decaying singular values and our intent is to demonstrate that in this environment, the approximation error of a barebones randomized method such as Algorithm 4.2 is *very* close to the minimal error achievable by any method. We observe that the approximation error of a randomized method is itself a random variable (it is a function of the random matrix $\boldsymbol{\Omega}$), so what we need to demonstrate is not only that the error is small in a typical realization, but also that it clusters tightly around the mean value.

We first consider a 200×200 matrix $\boldsymbol{A}$ that results from discretizing the following single-layer operator associated with the Laplace equation:

$$[S\sigma](x) = \text{const} \cdot \int_{\Gamma_1} \log|x-y|\ \sigma(y)\,\mathrm{d}A(y), \qquad x \in \Gamma_2, \tag{7.1}$$

where Γ_1 and Γ_2 are the two contours in $\mathbb{R}^2$ illustrated in Figure 7.1(a). We approximate the integral with the trapezoidal rule, which converges superalgebraically because the kernel is smooth. In the absence of floating-point errors, we estimate that the discretization error would be less than 10^{-20} for a smooth source σ. The leading constant is selected so the matrix $\boldsymbol{A}$ has unit operator norm.

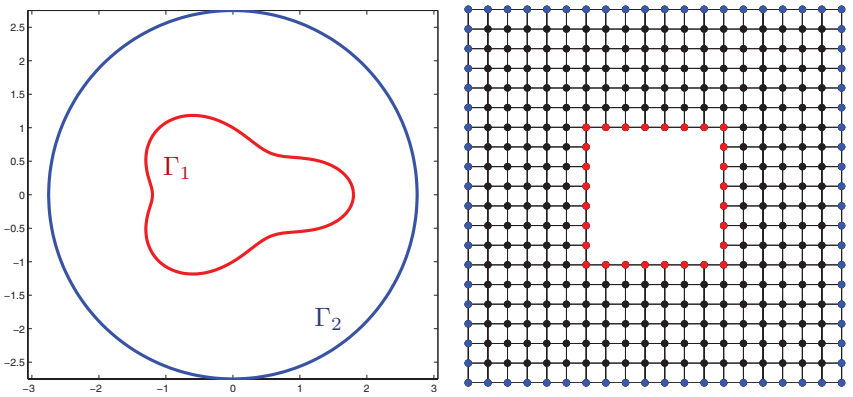

Fig. 7.1 *Configurations for physical problems.* (a) *The contours* Γ_1 *(red) and* Γ_2 *(blue) for the integral operator* (7.1). (b) *Geometry of the lattice problem associated with matrix* $\boldsymbol{B}$ *in section* 7.1.

We implement Algorithm 4.2 in MATLAB v6.5. Gaussian test matrices are generated using the `randn` command. For each number ℓ of samples, we compare the following three quantities:

1. The minimum rank-ℓ approximation error $\sigma_{\ell+1}$ is determined using `svd`.
2. The actual error $e_\ell = \left\| \left(\mathbf{I} - \boldsymbol{Q}^{(\ell)}(\boldsymbol{Q}^{(\ell)})^*\right)\boldsymbol{A} \right\|$ is computed with `norm`.
3. A random estimator f_ℓ for the actual error e_ℓ is obtained from (4.3), with the parameter r set to 5.

Note that any values less than 10^{-15} should be considered numerical artifacts.

Figure 7.2 tracks a characteristic execution of Algorithm 4.2. We make three observations: (i) The error e_ℓ incurred by the algorithm is remarkably close to the theoretical minimum $\sigma_{\ell+1}$. (ii) The error estimate always produces an upper bound for the actual error. Without the built-in $10\times$ safety margin, the estimate would track the actual error almost exactly. (iii) The basis constructed by the algorithm essentially reaches full double-precision accuracy.

How typical is the trial documented in Figure 7.2? To answer this question, we examine the empirical performance of the algorithm over 2000 independent trials. Figure 7.3 charts the error estimate versus the actual error at four points during the course of execution: $\ell = 25, 50, 75, 100$. We offer four observations: (i) The initial run detailed in Figure 7.2 is entirely typical. (ii) Both the actual and estimated error concentrate about their mean value. (iii) The actual error drifts slowly away from the optimal error as the number ℓ of samples increases. (iv) The error estimator is *always* pessimistic by a factor of about ten, which means that the algorithm *never* produces a basis with lower accuracy than requested. The only effect of selecting an unlucky sample matrix $\boldsymbol{\Omega}$ is that the algorithm proceeds for a few additional steps.

We next consider a matrix $\boldsymbol{B}$ which is defined implicitly in the sense that we cannot access its elements directly; we can only evaluate the map $\boldsymbol{x} \mapsto \boldsymbol{B}\boldsymbol{x}$ for a given vector $\boldsymbol{x}$. To be precise, $\boldsymbol{B}$ represents a transfer matrix for a network of resistors like the one shown in Figure 7.1(b). The vector $\boldsymbol{x}$ represents a set of electric potentials specified on the red nodes in the figure. These potentials induce a unique equilibrium field on the network in which the potential of each black and blue node is the average of the potentials of its three or four neighbors. The vector $\boldsymbol{B}\boldsymbol{x}$ is then the restriction

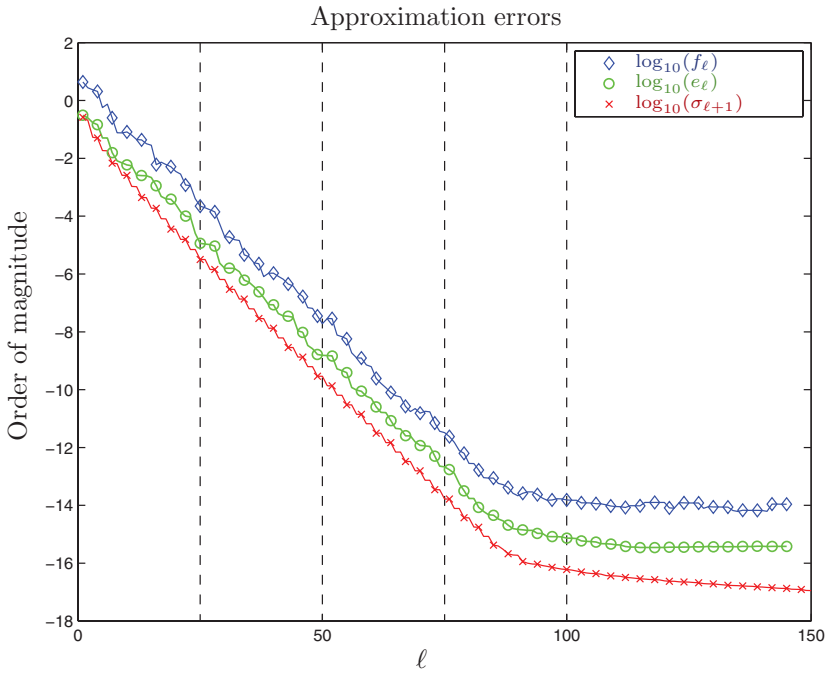

Fig. 7.2 *Approximating a Laplace integral operator. One execution of Algorithm* 4.2 *for the* 200×200 *input matrix* $\boldsymbol{A}$ *described in section* 7.1. *The number* ℓ *of random samples varies along the horizontal axis; the vertical axis measures the base-*10 *logarithm of error magnitudes. The dashed vertical lines mark the points during execution at which Figure* 7.3 *provides additional statistics.*

of the potential to the blue exterior nodes. Given a vector $\boldsymbol{x}$, the vector $\boldsymbol{B}\boldsymbol{x}$ can be obtained by solving a large sparse linear system whose coefficient matrix is the classical five-point stencil approximating the two-dimensional Laplace operator.

We applied Algorithm 4.2 to the 1596×532 matrix $\boldsymbol{B}$ associated with a lattice in which there were 532 nodes (red) on the "inner ring" and 1596 nodes on the (blue) "outer ring." Each application of $\boldsymbol{B}$ to a vector requires the solution of a sparse linear system of size roughly $140\,000 \times 140\,000$. We implemented the scheme in MATLAB using the "backslash" operator for the linear solve. The results of a typical trial appear in Figure 7.4. Qualitatively, the performance matches the results in Figure 7.3.

7.2. A Large, Sparse, Noisy Matrix Arising in Image Processing. Our next example involves a matrix that arises in image processing. A recent line of work uses information about the local geometry of an image to develop promising new algorithms for standard tasks, such as denoising, inpainting, and so forth. These methods are based on approximating a *graph Laplacian* associated with the image. The dominant eigenvectors of this matrix provide "coordinates" that help us smooth out noisy image patches [121, 132].

We begin with a 95×95 pixel grayscale image. The intensity of each pixel is represented as an integer in the range 0 to 4095. We form for each pixel i a vector $\boldsymbol{x}^{(i)} \in \mathbb{R}^{25}$ by gathering the 25 intensities of the pixels in a 5×5 neighborhood centered at pixel i (with appropriate modifications near the edges). Next, we form the 9025×9025 *weight matrix* $\widetilde{\boldsymbol{W}}$ that reflects the similarities between patches:

$$\widetilde{w}_{ij} = \exp\big\{ - \big\|\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)}\big\|^2 / \sigma^2 \big\},$$

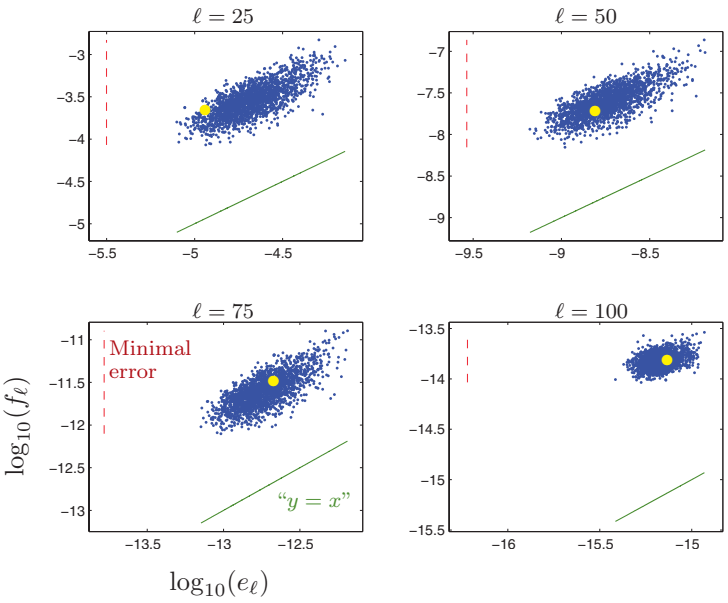

Fig. 7.3 *Error statistics for approximating a Laplace integral operator. 2,000 trials of Algorithm 4.2 applied to a 200×200 matrix approximating the integral operator (7.1). The panels isolate the moments at which $\ell = 25, 50, 75, 100$ random samples have been drawn. Each solid point compares the estimated error f_ℓ versus the actual error e_ℓ in one trial; the extra-large point in each panel indicates the trial detailed in Figure 7.2. The dashed line identifies the minimal error $\sigma_{\ell+1}$, and the solid line marks the contour where the error estimator would equal the actual error.*

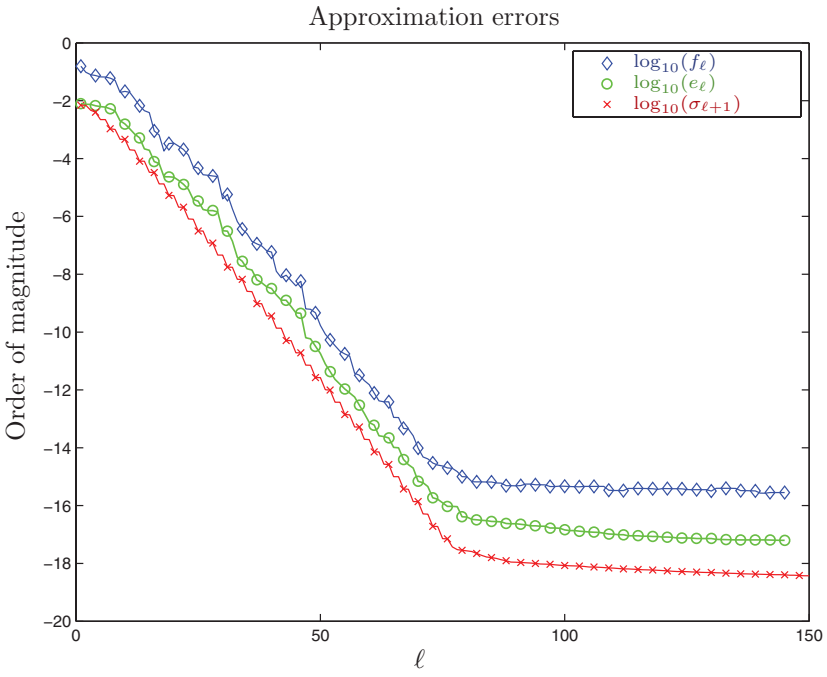

Fig. 7.4 *Approximating the inverse of a discrete Laplacian. One execution of Algorithm 4.2 for the 1596×532 input matrix $\boldsymbol{B}$ described in section 7.1. See Figure 7.2 for notations.*

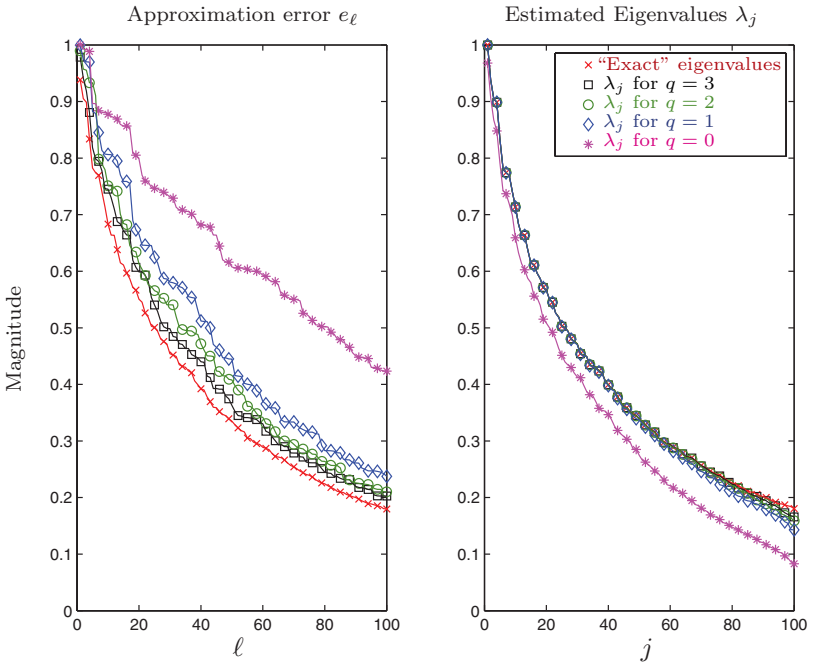

Fig. 7.5 *Approximating a graph Laplacian. For varying exponent q, one trial of the power scheme, Algorithm 4.3, applied to the 9025×9025 matrix $\boldsymbol{A}$ described in section 7.2.* (Left) *Approximation errors as a function of the number ℓ of random samples.* (Right) *Estimates for the 100 largest eigenvalues given $\ell = 100$ random samples compared with the 100 largest eigenvalues of $\boldsymbol{A}$.*

where the parameter $\sigma = 50$ controls the level of sensitivity. We obtain a sparse weight matrix $\boldsymbol{W}$ by zeroing out all entries in $\widetilde{\boldsymbol{W}}$ except the seven largest ones in each row. The object is then to construct the low-frequency eigenvectors of the graph Laplacian matrix

$$\boldsymbol{L} = \mathbf{I} - \boldsymbol{D}^{-1/2}\boldsymbol{W}\boldsymbol{D}^{-1/2},$$

where $\boldsymbol{D}$ is the diagonal matrix with entries $d_{ii} = \sum_j w_{ij}$. These are the eigenvectors associated with the dominant eigenvalues of the auxiliary matrix $\boldsymbol{A} = \boldsymbol{D}^{-1/2}\boldsymbol{W}\boldsymbol{D}^{-1/2}$.

The matrix $\boldsymbol{A}$ is large, and its eigenvalues decay slowly, so we use the power scheme summarized in Algorithm 4.3 to approximate it. Figure 7.5(left) illustrates how the approximation error e_ℓ declines as the number ℓ of samples increases. When we set the exponent $q = 0$, which corresponds with the basic Algorithm 4.1, the approximation is rather poor. The graph illustrates that slightly increasing the exponent q results in a tremendous improvement in the accuracy of the power scheme.

Next, we illustrate the results of using the two-stage approach to approximate the eigenvalues of $\boldsymbol{A}$. In Stage A, we construct a basis for $\boldsymbol{A}$ using Algorithm 4.3 with $\ell = 100$ samples for different values of q. In Stage B, we apply the Hermitian variant of Algorithm 5.1 described in section 5.3 to compute an approximate eigenvalue decomposition. Figure 7.5(right) shows the approximate eigenvalues and the actual eigenvalues of $\boldsymbol{A}$. Once again, we see that the minimal exponent $q = 0$ produces miserable results, but the largest eigenvalues are quite accurate even for $q = 1$.

7.3. Eigenfaces. Our next example involves a large, dense matrix derived from the FERET databank of face images [108, 109]. A simple method for performing face recognition is to identify the principal directions of the image data, which are called *eigenfaces*. Each of the original photographs can be summarized by its components along these principal directions. To identify the subject in a new picture, we compute its decomposition in this basis and use a classification technique, such as nearest neighbors, to select the closest image in the database [123].

We construct a data matrix $\boldsymbol{A}$ as follows: The FERET database contains 7254 images, and each 384×256 image contains 98 304 pixels. First, we build a $98\,304 \times 7254$ matrix $\widetilde{\boldsymbol{A}}$ whose columns are the images. We form $\boldsymbol{A}$ by centering each column of $\widetilde{\boldsymbol{A}}$ and scaling it to unit norm, so that the images are roughly comparable. The eigenfaces are the dominant left singular vectors of this matrix.

Our goal then is to compute an approximate SVD of the matrix $\boldsymbol{A}$. Represented as an array of double-precision real numbers, $\boldsymbol{A}$ would require 5.4 GB of storage, which does not fit within the fast memory of many machines. It is possible to compress the database down to 57 MB or less (in JPEG format), but then the data would have to be uncompressed with each sweep over the matrix. Furthermore, the matrix $\boldsymbol{A}$ has slowly decaying singular values, so we need to use the power scheme, Algorithm 4.3, to capture the range of the matrix accurately.

To address these concerns, we implemented the power scheme to run in a pass-efficient manner. An additional difficulty arises because the size of the data makes it expensive to calculate the actual error e_ℓ incurred by the approximation or to determine the minimal error $\sigma_{\ell+1}$. To estimate the errors, we use the technique described in Remark 4.1.

Figure 7.6 describes the behavior of the power scheme, which is similar to its performance for the graph Laplacian in section 7.2. When the exponent $q = 0$, the approximation of the data matrix is very poor, but it improves quickly as q increases. Likewise, the estimate for the spectrum of $\boldsymbol{A}$ appears to converge rapidly; the largest singular values are already quite accurate when $q = 1$. We see essentially no improvement in the estimates after the first 3–5 passes over the matrix.

7.4. Performance of Structured Random Matrices. Our final set of experiments illustrates that the structured random matrices described in section 4.6 lead to matrix approximation algorithms that are both fast and accurate.

First, we compare the computational speeds of four methods for computing an approximation to the ℓ dominant terms in the SVD of an $n \times n$ matrix $\boldsymbol{A}$. For now, we are interested in execution time only (not accuracy), so the choice of matrix is irrelevant and we have selected $\boldsymbol{A}$ to be a Gaussian matrix. The four methods are summarized in the following table; Remark 7.1 provides more details on the implementation.

Method	Stage A	Stage B
`direct`	Rank-revealing QR executed using column pivoting and Householder reflectors	Algorithm 5.1
`gauss`	Algorithm 4.1 with a Gaussian random matrix	Algorithm 5.1
`srft`	Algorithm 4.1 with the modified SRFT (4.8)	Algorithm 5.2
`svd`	Full SVD with LAPACK routine `dgesdd`	Truncate to ℓ terms

Table 7.1 lists the measured runtime of a single execution of each algorithm for various choices of the dimension n of the input matrix and the rank ℓ of the approximation. Of course, the cost of the full SVD does not depend on the number ℓ

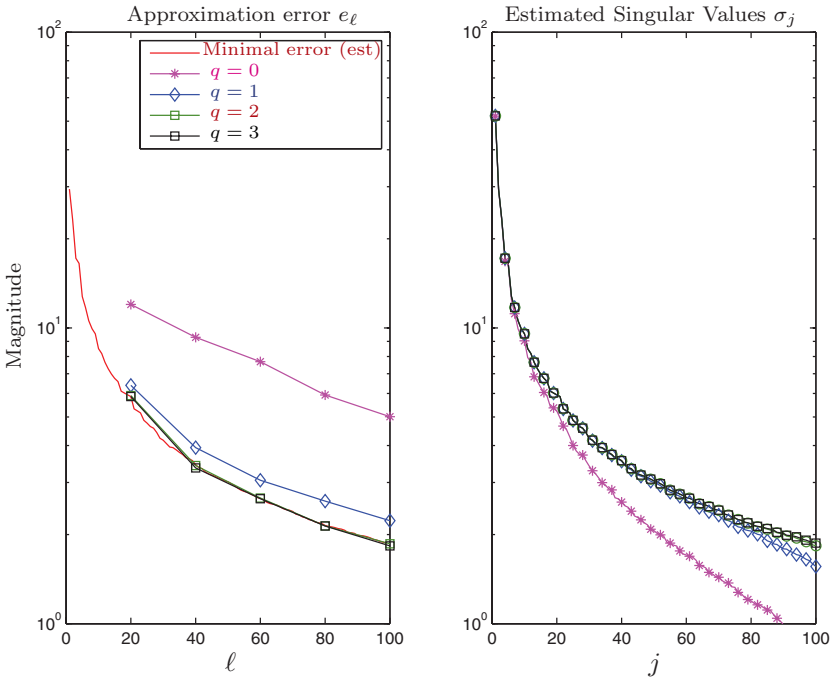

Fig. 7.6 *Computing eigenfaces. For varying exponent q, one trial of the power scheme, Algorithm* 4.3*, applied to the* $98\,304 \times 7254$ *matrix $\boldsymbol{A}$ described in section* 7.3. (Left) *Approximation errors as a function of the number ℓ of random samples. The solid line indicates the minimal errors as estimated by the singular values computed using $\ell = 100$ and $q = 3$.* (Right) *Estimates for the* 100 *largest eigenvalues given $\ell = 100$ random samples.*

of components required. A more informative way to look at the runtime data is to compare the *relative* cost of the algorithms. The `direct` method is the best deterministic approach for dense matrices, so we calculate the factor by which the randomized methods improve on this benchmark. Figure 7.7 displays the results. We make two observations: (i) Using an SRFT often leads to a dramatic speed-up over classical techniques, even for moderate problem sizes. (ii) Using a standard Gaussian test matrix typically leads to a moderate speed-up over classical methods, primarily because performing a matrix–matrix multiplication is faster than a QR factorization.

Second, we investigate how the choice of random test matrix influences the error in approximating an input matrix. For these experiments, we return to the 200×200 matrix $\boldsymbol{A}$ defined in section 7.1. Consider variations of Algorithm 4.1 obtained when the random test matrix $\boldsymbol{\Omega}$ is drawn from the following four distributions:

Gauss: The standard Gaussian distribution.

Ortho: The uniform distribution on $n \times \ell$ orthonormal matrices.

SRFT: The SRFT distribution defined in (4.6).

GSRFT: The modified SRFT distribution defined in (4.8).

Intuitively, we expect that Ortho should provide the best performance.

For each distribution, we perform 100 000 trials of the following experiment. Apply the corresponding version of Algorithm 4.1 to the matrix $\boldsymbol{A}$, and calculate the approximation error $e_\ell = \|\boldsymbol{A} - \boldsymbol{Q}_\ell \boldsymbol{Q}_\ell^* \boldsymbol{A}\|$. Figure 7.8 displays the empirical proba-

Table 7.1 *Computational times for a partial SVD. The time, in seconds, required to compute the ℓ leading components in the SVD of an $n \times n$ matrix using each of the methods from section 7.4. The last row indicates the time needed to obtain a full SVD.*

	$n = 1024$			$n = 2048$			$n = 4096$		
ℓ	`direct`	`gauss`	`srft`	`direct`	`gauss`	`srft`	`direct`	`gauss`	`srft`
10	1.08e-1	5.63e-2	9.06e-2	4.22e-1	2.16e-1	3.56e-1	1.70e 0	8.94e-1	1.45e 0
20	1.97e-1	9.69e-2	1.03e-1	7.67e-1	3.69e-1	3.89e-1	3.07e 0	1.44e 0	1.53e 0
40	3.91e-1	1.84e-1	1.27e-1	1.50e 0	6.69e-1	4.33e-1	6.03e 0	2.64e 0	1.63e 0
80	7.84e-1	4.00e-1	2.19e-1	3.04e 0	1.43e 0	6.64e-1	1.20e 1	5.43e 0	2.08e 0
160	1.70e 0	9.92e-1	6.92e-1	6.36e 0	3.36e 0	1.61e 0	2.46e 1	1.16e 1	3.94e 0
320	3.89e 0	2.65e 0	2.98e 0	1.34e 1	7.45e 0	5.87e 0	5.00e 1	2.41e 1	1.21e 1
640	1.03e 1	8.75e 0	1.81e 1	3.14e 1	2.13e 1	2.99e 1	1.06e 2	5.80e 1	5.35e 1
1280	—	—	—	7.97e 1	6.69e 1	3.13e 2	2.40e 2	1.68e 2	4.03e 2
`svd`	1.19e 1			8.77e 1			6.90e 2		

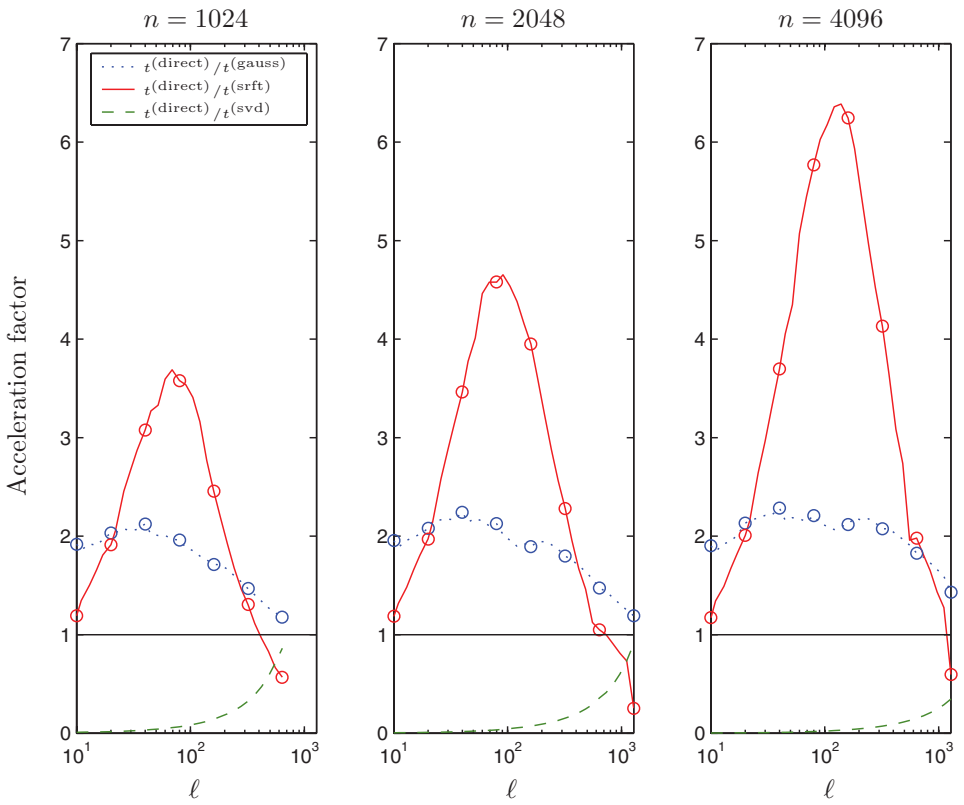

Fig. 7.7 *Acceleration factor. The relative cost of computing an ℓ-term partial SVD of an $n \times n$ Gaussian matrix using* `direct`*, a benchmark classical algorithm, versus each of the three competitors described in section 7.4. The solid curve shows the speedup using an SRFT test matrix, and the dotted curve shows the speedup with a Gaussian test matrix. The dashed curve indicates that a full SVD computation using classical methods is substantially* slower*. Table 7.1 reports the absolute runtimes that yield the circled data points.*

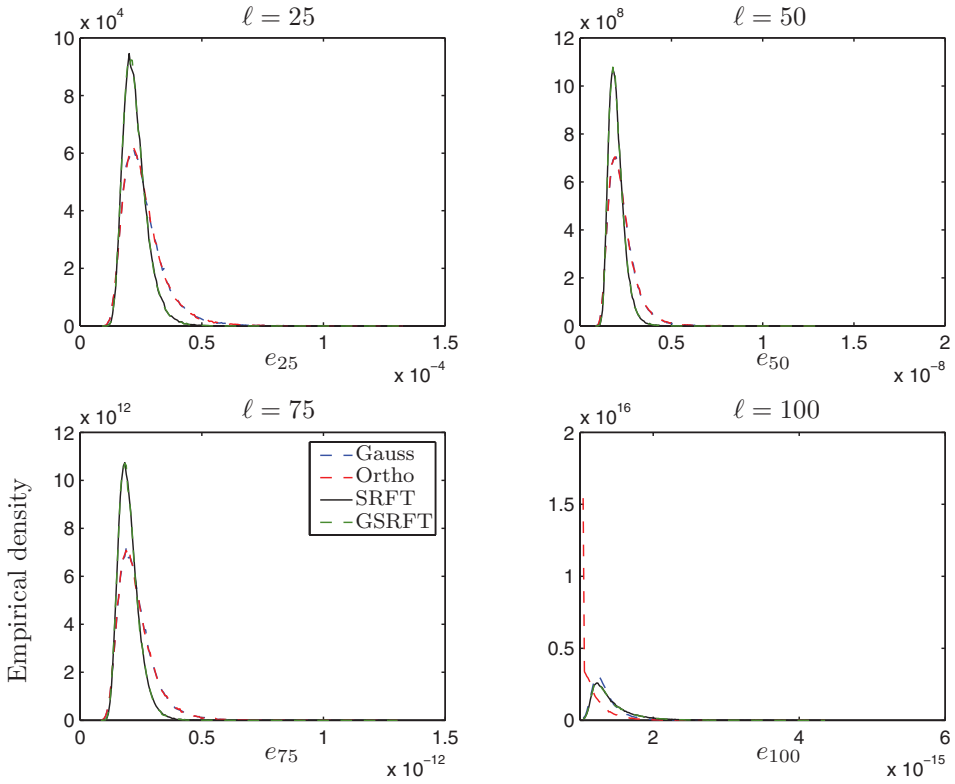

Fig. 7.8 *Empirical probability density functions for the error in Algorithm* 4.1. *As described in section* 7.4, *the algorithm is implemented with four distributions for the random test matrix and used to approximate the* 200×200 *input matrix obtained by discretizing the integral operator* (7.1). *The four panels capture the empirical error distribution for each version of the algorithm at the moment when* $\ell = 25, 50, 75, 100$ *random samples have been drawn.*

bility density function for the error e_ℓ obtained with each algorithm. We offer three observations: (i) The SRFT actually performs slightly *better* than a Gaussian random matrix for this example. (ii) The standard SRFT and the modified SRFT have essentially identical errors. (iii) There is almost no difference between the Gaussian random matrix and the random orthonormal matrix in the first three plots, while the fourth plot shows that the random orthonormal matrix performs better. This behavior occurs because, with high probability, a tall Gaussian matrix is well conditioned and a (nearly) square Gaussian matrix is not.

Remark 7.1. The running times reported in Table 7.1 and in Figure 7.7 depend strongly on both the computer hardware and the coding of the algorithms. The experiments reported here were performed on a standard office desktop with a 3.2 GHz Pentium IV processor and 2 GB of RAM. The algorithms were implemented in Fortran 90 and compiled with the Lahey compiler. The Lahey versions of BLAS and LAPACK were used to accelerate all matrix–matrix multiplications, as well as the SVD computations in Algorithms 5.1 and 5.2. We used the code for the modified SRFT (4.8) provided in the publicly available software package `id_dist` [91].

◇ ◇ ◇

Part III: Theory. This part of the paper, sections 8–11, provides a detailed analysis of randomized sampling schemes for constructing an approximate basis for the range of a matrix, the task we refer to as Stage A in the framework of section 1.2. More precisely, we assess the quality of the basis $\boldsymbol{Q}$ that the proto-algorithm of section 1.3 produces by establishing rigorous bounds for the approximation error

$$\|\!|\boldsymbol{A} - \boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{A}|\!\|,$$

where $\|\!|\cdot|\!\|$ denotes either the spectral norm or the Frobenius norm. The difficulty in developing these bounds is that the matrix $\boldsymbol{Q}$ is random, and its distribution is a complicated nonlinear function of the input matrix $\boldsymbol{A}$ and the random test matrix $\boldsymbol{\Omega}$. Naturally, any estimate for the approximation error must depend on the properties of the input matrix and the distribution of the test matrix.

To address these challenges, we split the argument into two pieces. The first part exploits techniques from linear algebra to deliver a generic error bound that depends on the interaction between the test matrix $\boldsymbol{\Omega}$ and the right singular vectors of the input matrix $\boldsymbol{A}$, as well as the tail singular values of $\boldsymbol{A}$. In the second part of the argument, we take into account the distribution of the random matrix to estimate the error for specific instantiations of the proto-algorithm. This bipartite proof is common in the literature on randomized linear algebra, but our argument is most similar in spirit to [17].

Section 8 surveys the basic linear algebraic tools we need. Section 9 uses these methods to derive a generic error bound. Afterward, we specialize these results to the case where the test matrix is Gaussian (section 10) and the case where the test matrix is a subsampled random Fourier transform (section 11).

8. Theoretical Preliminaries.

We proceed with some additional background from linear algebra. Section 8.1 sets out properties of positive-semidefinite matrices, and section 8.2 offers some results for orthogonal projectors. Standard references for this material include [11, 72].

8.1. Positive Semidefinite Matrices.

An Hermitian matrix $\boldsymbol{M}$ is *positive semidefinite* (briefly, *psd*) when $\boldsymbol{u}^*\boldsymbol{M}\boldsymbol{u} \geq 0$ for all $\boldsymbol{u} \neq \boldsymbol{0}$. If the inequalities are strict, $\boldsymbol{M}$ is *positive definite* (briefly, *pd*). The psd matrices form a convex cone, which induces a partial ordering on the linear space of Hermitian matrices: $\boldsymbol{M} \preccurlyeq \boldsymbol{N}$ if and only if $\boldsymbol{N} - \boldsymbol{M}$ is psd. This ordering allows us to write $\boldsymbol{M} \succcurlyeq \boldsymbol{0}$ to indicate that the matrix $\boldsymbol{M}$ is psd.

Alternatively, we can define a psd (resp., pd) matrix as an Hermitian matrix with nonnegative (resp., positive) eigenvalues. In particular, each psd matrix is diagonalizable, and the inverse of a pd matrix is also pd. The spectral norm of a psd matrix $\boldsymbol{M}$ has the variational characterization

$$\|\boldsymbol{M}\| = \max_{\boldsymbol{u}\neq\boldsymbol{0}} \frac{\boldsymbol{u}^*\boldsymbol{M}\boldsymbol{u}}{\boldsymbol{u}^*\boldsymbol{u}} \tag{8.1}$$

according to the Rayleigh–Ritz theorem [72, Thm. 4.2.2]. It follows that

$$\boldsymbol{M} \preccurlyeq \boldsymbol{N} \quad\Longrightarrow\quad \|\boldsymbol{M}\| \leq \|\boldsymbol{N}\|. \tag{8.2}$$

A fundamental fact is that conjugation preserves the psd property.

PROPOSITION 8.1 (conjugation rule). *Suppose that $\boldsymbol{M} \succcurlyeq \boldsymbol{0}$. For every $\boldsymbol{A}$, the matrix $\boldsymbol{A}^*\boldsymbol{M}\boldsymbol{A} \succcurlyeq \boldsymbol{0}$. In particular,*

$$\boldsymbol{M} \preccurlyeq \boldsymbol{N} \quad\Longrightarrow\quad \boldsymbol{A}^*\boldsymbol{M}\boldsymbol{A} \preccurlyeq \boldsymbol{A}^*\boldsymbol{N}\boldsymbol{A}.$$

Our argument invokes the conjugation rule repeatedly. As a first application, we establish a perturbation bound for the matrix inverse near the identity matrix.

PROPOSITION 8.2 (perturbation of inverses). *Suppose that* $\boldsymbol{M} \succcurlyeq \mathbf{0}$. *Then*

$$\mathbf{I} - (\mathbf{I} + \boldsymbol{M})^{-1} \preccurlyeq \boldsymbol{M}.$$

Proof. Define $\boldsymbol{R} = \boldsymbol{M}^{1/2}$, the psd square root of $\boldsymbol{M}$ promised by [72, Thm. 7.2.6]. We have the chain of relations

$$\mathbf{I} - (\mathbf{I} + \boldsymbol{R}^2)^{-1} = (\mathbf{I} + \boldsymbol{R}^2)^{-1}\boldsymbol{R}^2 = \boldsymbol{R}(\mathbf{I} + \boldsymbol{R}^2)^{-1}\boldsymbol{R} \preccurlyeq \boldsymbol{R}^2.$$

The first equality can be verified algebraically. The second holds because rational functions of a diagonalizable matrix, such as $\boldsymbol{R}$, commute. The last relation follows from the conjugation rule because $(\mathbf{I} + \boldsymbol{R}^2)^{-1} \preccurlyeq \mathbf{I}$. ∎

Next, we present a generalization of the fact that the spectral norm of a psd matrix is controlled by its trace.

PROPOSITION 8.3. *We have* $\|\boldsymbol{M}\| \le \|\boldsymbol{A}\| + \|\boldsymbol{C}\|$ *for each partitioned psd matrix*

$$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{B}^* & \boldsymbol{C} \end{bmatrix}.$$

Proof. The variational characterization (8.1) of the spectral norm implies that

$$\begin{aligned} \|\boldsymbol{M}\| &= \sup_{\|\boldsymbol{x}\|^2 + \|\boldsymbol{y}\|^2 = 1} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix}^* \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{B}^* & \boldsymbol{C} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix} \\ &\le \sup_{\|\boldsymbol{x}\|^2 + \|\boldsymbol{y}\|^2 = 1} \big(\|\boldsymbol{A}\| \, \|\boldsymbol{x}\|^2 + 2 \, \|\boldsymbol{B}\| \, \|\boldsymbol{x}\| \, \|\boldsymbol{y}\| + \|\boldsymbol{C}\| \, \|\boldsymbol{y}\|^2 \big). \end{aligned}$$

The block generalization of Hadamard's psd criterion [72, Thm. 7.7.7] states that $\|\boldsymbol{B}\|^2 \le \|\boldsymbol{A}\| \, \|\boldsymbol{C}\|$. Thus,

$$\|\boldsymbol{M}\| \le \sup_{\|\boldsymbol{x}\|^2 + \|\boldsymbol{y}\|^2 = 1} \big(\|\boldsymbol{A}\|^{1/2} \, \|\boldsymbol{x}\| + \|\boldsymbol{C}\|^{1/2} \, \|\boldsymbol{y}\| \big)^2 = \|\boldsymbol{A}\| + \|\boldsymbol{C}\|.$$

This point completes the argument. ∎

8.2. Orthogonal Projectors. An *orthogonal projector* is an Hermitian matrix $\boldsymbol{P}$ that satisfies the polynomial $\boldsymbol{P}^2 = \boldsymbol{P}$. This identity implies $\mathbf{0} \preccurlyeq \boldsymbol{P} \preccurlyeq \mathbf{I}$. An orthogonal projector is completely determined by its range. For a given matrix $\boldsymbol{M}$, we write $\boldsymbol{P}_{\boldsymbol{M}}$ for the unique orthogonal projector with $\operatorname{range}(\boldsymbol{P}_{\boldsymbol{M}}) = \operatorname{range}(\boldsymbol{M})$. When $\boldsymbol{M}$ has full column rank, we can express this projector explicitly:

$$\boldsymbol{P}_{\boldsymbol{M}} = \boldsymbol{M}(\boldsymbol{M}^*\boldsymbol{M})^{-1}\boldsymbol{M}^*. \tag{8.3}$$

The orthogonal projector onto the complementary subspace, $\operatorname{range}(\boldsymbol{P})^{\perp}$, is the matrix $\mathbf{I} - \boldsymbol{P}$. Our argument hinges on several other facts about orthogonal projectors.

PROPOSITION 8.4. *Suppose* $\boldsymbol{U}$ *is unitary. Then* $\boldsymbol{U}^*\boldsymbol{P}_{\boldsymbol{M}}\boldsymbol{U} = \boldsymbol{P}_{\boldsymbol{U}^*\boldsymbol{M}}$.

Proof. Abbreviate $\boldsymbol{P} = \boldsymbol{U}^*\boldsymbol{P}_{\boldsymbol{M}}\boldsymbol{U}$. It is clear that $\boldsymbol{P}$ is an orthogonal projector since it is Hermitian and $\boldsymbol{P}^2 = \boldsymbol{P}$. Evidently,

$$\operatorname{range}(\boldsymbol{P}) = \boldsymbol{U}^* \operatorname{range}(\boldsymbol{M}) = \operatorname{range}(\boldsymbol{U}^*\boldsymbol{M}).$$

Since the range determines the orthogonal projector, we conclude $\boldsymbol{P} = \boldsymbol{P}_{\boldsymbol{U}^*\boldsymbol{M}}$. ∎

PROPOSITION 8.5. *Suppose* $\operatorname{range}(\boldsymbol{N}) \subset \operatorname{range}(\boldsymbol{M})$. *Then, for each matrix* $\boldsymbol{A}$, *it holds that* $\|\boldsymbol{P_N A}\| \le \|\boldsymbol{P_M A}\|$ *and that* $\|(\mathbf{I} - \boldsymbol{P_M})\boldsymbol{A}\| \le \|(\mathbf{I} - \boldsymbol{P_N})\boldsymbol{A}\|$.

Proof. The projector $\boldsymbol{P_N} \preccurlyeq \mathbf{I}$, so the conjugation rule yields $\boldsymbol{P_M P_N P_M} \preccurlyeq \boldsymbol{P_M}$. The hypothesis $\operatorname{range}(\boldsymbol{N}) \subset \operatorname{range}(\boldsymbol{M})$ implies that $\boldsymbol{P_M P_N} = \boldsymbol{P_N}$, which results in

$$\boldsymbol{P_M P_N P_M} = \boldsymbol{P_N P_M} = (\boldsymbol{P_M P_N})^* = \boldsymbol{P_N}.$$

In summary, $\boldsymbol{P_N} \preccurlyeq \boldsymbol{P_M}$. The conjugation rule shows that $\boldsymbol{A}^*\boldsymbol{P_N A} \preccurlyeq \boldsymbol{A}^*\boldsymbol{P_M A}$. We conclude from (8.2) that

$$\|\boldsymbol{P_N A}\|^2 = \|\boldsymbol{A}^*\boldsymbol{P_N A}\| \le \|\boldsymbol{A}^*\boldsymbol{P_M A}\| = \|\boldsymbol{P_M A}\|^2.$$

The second statement follows from the first by taking orthogonal complements. ∎

Finally, we need a generalization of the scalar inequality $|px|^q \le |p|\,|x|^q$, which holds when $|p| \le 1$ and $q \ge 1$.

PROPOSITION 8.6. *Let* $\boldsymbol{P}$ *be an orthogonal projector, and let* $\boldsymbol{M}$ *be a matrix. For each positive number* q,

$$\|\boldsymbol{PM}\| \le \|\boldsymbol{P}(\boldsymbol{MM}^*)^q\boldsymbol{M}\|^{1/(2q+1)}. \tag{8.4}$$

Proof. Suppose that $\boldsymbol{R}$ is an orthogonal projector, $\boldsymbol{D}$ is a nonnegative diagonal matrix, and $t \ge 1$. We claim that

$$\|\boldsymbol{RDR}\|^t \le \left\|\boldsymbol{RD}^t\boldsymbol{R}\right\|. \tag{8.5}$$

Granted this inequality, we quickly complete the proof. Using an SVD $\boldsymbol{M} = \boldsymbol{U\Sigma V}^*$, we compute

$$\begin{aligned}\|\boldsymbol{PM}\|^{2(2q+1)} &= \|\boldsymbol{PMM}^*\boldsymbol{P}\|^{2q+1} = \left\|(\boldsymbol{U}^*\boldsymbol{PU})\cdot\boldsymbol{\Sigma}^2\cdot(\boldsymbol{U}^*\boldsymbol{PU})\right\|^{2q+1} \\ &\le \left\|(\boldsymbol{U}^*\boldsymbol{PU})\cdot\boldsymbol{\Sigma}^{2(2q+1)}\cdot(\boldsymbol{U}^*\boldsymbol{PU})\right\| = \left\|\boldsymbol{P}(\boldsymbol{MM}^*)^{2(2q+1)}\boldsymbol{P}\right\| \\ &= \left\|\boldsymbol{P}(\boldsymbol{MM}^*)^q\boldsymbol{M}\cdot\boldsymbol{M}^*(\boldsymbol{MM}^*)^q\boldsymbol{P}\right\| = \|\boldsymbol{P}(\boldsymbol{MM}^*)^q\boldsymbol{M}\|^2.\end{aligned}$$

We have used the unitary invariance of the spectral norm in the second and fourth relations. Inequality (8.5) applies because $\boldsymbol{U}^*\boldsymbol{PU}$ is an orthogonal projector. Take a square root to finish the argument.

Now, we turn to the claim (8.5). This relation follows immediately from [11, Thm. IX.2.10], but we offer a direct argument based on more elementary considerations. Let $\boldsymbol{x}$ be a unit vector at which

$$\boldsymbol{x}^*(\boldsymbol{RDR})\boldsymbol{x} = \|\boldsymbol{RDR}\|.$$

We must have $\boldsymbol{Rx} = \boldsymbol{x}$. Otherwise, $\|\boldsymbol{Rx}\| < 1$ because $\boldsymbol{R}$ is an orthogonal projector, which implies that the unit vector $\boldsymbol{y} = \boldsymbol{Rx}/\|\boldsymbol{Rx}\|$ verifies

$$\boldsymbol{y}^*(\boldsymbol{RDR})\boldsymbol{y} = \frac{(\boldsymbol{Rx})^*(\boldsymbol{RDR})(\boldsymbol{Rx})}{\|\boldsymbol{Rx}\|^2} = \frac{\boldsymbol{x}^*(\boldsymbol{RDR})\boldsymbol{x}}{\|\boldsymbol{Rx}\|^2} > \boldsymbol{x}^*(\boldsymbol{RDR})\boldsymbol{x}.$$

Writing x_j for the entries of $\boldsymbol{x}$ and d_j for the diagonal entries of $\boldsymbol{D}$, we find that

$$\begin{aligned}\|\boldsymbol{RDR}\|^t = [\boldsymbol{x}^*(\boldsymbol{RDR})\boldsymbol{x}]^t = [\boldsymbol{x}^*\boldsymbol{Dx}]^t &= \left[\textstyle\sum_j d_j x_j^2\right]^t \\ &\le \left[\textstyle\sum_j d_j^t x_j^2\right] = \boldsymbol{x}^*\boldsymbol{D}^t\boldsymbol{x} = (\boldsymbol{Rx})^*\boldsymbol{D}^t(\boldsymbol{Rx}) \le \left\|\boldsymbol{RD}^t\boldsymbol{R}\right\|.\end{aligned}$$

The inequality is Jensen's, which applies because $\sum x_j^2 = 1$ and the function $z \mapsto |z|^t$ is convex for $t \ge 1$. ∎

9. Error Bounds via Linear Algebra. We are now prepared to develop a deterministic error analysis for the proto-algorithm described in section 1.3. To begin, we must introduce some notation. Afterward, we establish the key error bound, which strengthens a result from the literature [17, Lem. 4.2]. Finally, we explain why the power method can be used to improve the performance of the proto-algorithm.

9.1. Setup. Let $\boldsymbol{A}$ be an $m \times n$ matrix that has an SVD $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^*$, as described in section 3.2.2. Roughly speaking, the proto-algorithm tries to approximate the subspace spanned by the first k left singular vectors, where k is now a fixed number. To perform the analysis, it is appropriate to partition the SVD as follows:

$$\boldsymbol{A} = \boldsymbol{U} \begin{array}{c} \\ k \\ n-k \end{array} \begin{array}{c} \begin{array}{cc} k & n-k \end{array} \\ \begin{bmatrix} \boldsymbol{\Sigma}_1 & \\ & \boldsymbol{\Sigma}_2 \end{bmatrix} \end{array} \begin{array}{c} \begin{array}{c} n \end{array} \\ \begin{bmatrix} \boldsymbol{V}_1^* \\ \boldsymbol{V}_2^* \end{bmatrix} \end{array} \begin{array}{c} \\ k \\ n-k \end{array}. \tag{9.1}$$

The matrices $\boldsymbol{\Sigma}_1$ and $\boldsymbol{\Sigma}_2$ are square. We will see that the left unitary factor $\boldsymbol{U}$ does not play a significant role in the analysis.

Let $\boldsymbol{\Omega}$ be an $n \times \ell$ test matrix, where ℓ denotes the number of samples. We assume only that $\ell \geq k$. Decompose the test matrix in the coordinate system determined by the right unitary factor of $\boldsymbol{A}$:

$$\boldsymbol{\Omega}_1 = \boldsymbol{V}_1^* \boldsymbol{\Omega} \quad \text{and} \quad \boldsymbol{\Omega}_2 = \boldsymbol{V}_2^* \boldsymbol{\Omega}. \tag{9.2}$$

The error bound for the proto-algorithm depends critically on the properties of the matrices $\boldsymbol{\Omega}_1$ and $\boldsymbol{\Omega}_2$. With this notation, the sample matrix $\boldsymbol{Y}$ can be expressed as

$$\boldsymbol{Y} = \boldsymbol{A}\boldsymbol{\Omega} = \boldsymbol{U} \begin{array}{c} \\ k \\ n-k \end{array} \begin{array}{c} \begin{array}{c} \ell \end{array} \\ \begin{bmatrix} \boldsymbol{\Sigma}_1\boldsymbol{\Omega}_1 \\ \boldsymbol{\Sigma}_2\boldsymbol{\Omega}_2 \end{bmatrix} \end{array} \begin{array}{c} \\ k \\ n-k \end{array}.$$

It is a useful intuition that the block $\boldsymbol{\Sigma}_1\boldsymbol{\Omega}_1$ in (9.1) reflects the gross behavior of $\boldsymbol{A}$, while the block $\boldsymbol{\Sigma}_2\boldsymbol{\Omega}_2$ represents a perturbation.

9.2. A Deterministic Error Bound for the Proto-algorithm. The proto-algorithm constructs an orthonormal basis $\boldsymbol{Q}$ for the range of the sample matrix $\boldsymbol{Y}$, and our goal is to quantify how well this basis captures the action of the input $\boldsymbol{A}$. Since $\boldsymbol{Q}\boldsymbol{Q}^* = \boldsymbol{P}_{\boldsymbol{Y}}$, the challenge is to obtain bounds on the approximation error

$$\|\boldsymbol{A} - \boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{A}\| = \|(\mathbf{I} - \boldsymbol{P}_{\boldsymbol{Y}})\boldsymbol{A}\|.$$

The following theorem shows that the behavior of the proto-algorithm depends on the interaction between the test matrix and the right singular vectors of the input matrix, as well as the singular spectrum of the input matrix.

THEOREM 9.1 (deterministic error bound). *Let $\boldsymbol{A}$ be an $m \times n$ matrix with SVD $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^*$, and fix $k \geq 0$. Choose a test matrix $\boldsymbol{\Omega}$, and construct the sample matrix $\boldsymbol{Y} = \boldsymbol{A}\boldsymbol{\Omega}$. Partition $\boldsymbol{\Sigma}$ as specified in* (9.1)*, and define $\boldsymbol{\Omega}_1$ and $\boldsymbol{\Omega}_2$ via* (9.2)*. Assuming that $\boldsymbol{\Omega}_1$ has full row rank, the approximation error satisfies*

$$\|(\mathbf{I} - \boldsymbol{P}_{\boldsymbol{Y}})\boldsymbol{A}\|^2 \leq \|\boldsymbol{\Sigma}_2\|^2 + \left\|\boldsymbol{\Sigma}_2\boldsymbol{\Omega}_2\boldsymbol{\Omega}_1^\dagger\right\|^2, \tag{9.3}$$

where $\|\cdot\|$ denotes either the spectral norm or the Frobenius norm.

Theorem 9.1 sharpens the result [17, Lem. 2], which lacks the squares present in (9.3). This refinement yields slightly better error estimates than the earlier bound,

and it has consequences for the probabilistic behavior of the error when the test matrix $\boldsymbol{\Omega}$ is random. The proof here is different in spirit from the earlier analysis; our argument is inspired by the perturbation theory of orthogonal projectors [126].

Proof. We establish the bound for the spectral-norm error. The bound for the Frobenius-norm error follows from an analogous argument that is slightly easier.

Let us begin with some preliminary simplifications. First, we argue that the left unitary factor $\boldsymbol{U}$ plays no essential role in the argument. In effect, we execute the proof for an auxiliary input matrix $\widetilde{\boldsymbol{A}}$ and an associated sample matrix $\widetilde{\boldsymbol{Y}}$ defined by

$$\widetilde{\boldsymbol{A}} = \boldsymbol{U}^*\boldsymbol{A} = \begin{bmatrix} \boldsymbol{\Sigma}_1\boldsymbol{V}_1^* \\ \boldsymbol{\Sigma}_2\boldsymbol{V}_2^* \end{bmatrix} \quad \text{and} \quad \widetilde{\boldsymbol{Y}} = \widetilde{\boldsymbol{A}}\boldsymbol{\Omega} = \begin{bmatrix} \boldsymbol{\Sigma}_1\boldsymbol{\Omega}_1 \\ \boldsymbol{\Sigma}_2\boldsymbol{\Omega}_2 \end{bmatrix}. \tag{9.4}$$

Owing to the unitary invariance of the spectral norm and to Proposition 8.4, we have the identity

$$\|(\mathbf{I} - \boldsymbol{P}_{\boldsymbol{Y}})\boldsymbol{A}\| = \big\|\boldsymbol{U}^*(\mathbf{I} - \boldsymbol{P}_{\boldsymbol{Y}})\boldsymbol{U}\widetilde{\boldsymbol{A}}\big\| = \big\|(\mathbf{I} - \boldsymbol{P}_{\boldsymbol{U}^*\boldsymbol{Y}})\widetilde{\boldsymbol{A}}\big\| = \big\|(\mathbf{I} - \boldsymbol{P}_{\widetilde{\boldsymbol{Y}}})\widetilde{\boldsymbol{A}}\big\|. \tag{9.5}$$

In view of (9.5), it suffices to prove that

$$\big\|(\mathbf{I} - \boldsymbol{P}_{\widetilde{\boldsymbol{Y}}})\widetilde{\boldsymbol{A}}\big\| \le \big\|\boldsymbol{\Sigma}_2\big\|^2 + \big\|\boldsymbol{\Sigma}_2\boldsymbol{\Omega}_2\boldsymbol{\Omega}_1^\dagger\big\|^2. \tag{9.6}$$

Second, we assume that the number k is chosen so the diagonal entries of $\boldsymbol{\Sigma}_1$ are strictly positive. Suppose not. Then $\boldsymbol{\Sigma}_2$ is zero because of the ordering of the singular values. As a consequence,

$$\operatorname{range}(\widetilde{\boldsymbol{A}}) = \operatorname{range}\begin{bmatrix} \boldsymbol{\Sigma}_1\boldsymbol{V}_1^* \\ \mathbf{0} \end{bmatrix} = \operatorname{range}\begin{bmatrix} \boldsymbol{\Sigma}_1\boldsymbol{\Omega}_1 \\ \mathbf{0} \end{bmatrix} = \operatorname{range}(\widetilde{\boldsymbol{Y}}).$$

This calculation uses the decompositions presented in (9.4), as well as the fact that both $\boldsymbol{V}_1^*$ and $\boldsymbol{\Omega}_1$ have full row rank. We conclude that

$$\big\|(\mathbf{I} - \boldsymbol{P}_{\widetilde{\boldsymbol{Y}}})\widetilde{\boldsymbol{A}}\big\| = 0,$$

so the error bound (9.6) holds trivially. (In fact, both sides are zero.)

The main argument is based on ideas from perturbation theory. To illustrate the concept, we start with a matrix related to $\widetilde{\boldsymbol{Y}}$:

$$\boldsymbol{W} = \begin{array}{c} \ell \\ \left[\begin{array}{c} \boldsymbol{\Sigma}_1\boldsymbol{\Omega}_1 \\ \mathbf{0} \end{array}\right] \\ \end{array} \begin{array}{c} \\ \begin{array}{c} k \\ n-k \end{array} \end{array}.$$

The matrix $\boldsymbol{W}$ has the same range as a related matrix formed by "flattening out" the spectrum of the top block. Indeed, since $\boldsymbol{\Sigma}_1\boldsymbol{\Omega}_1$ has full row rank,

$$\operatorname{range}(\boldsymbol{W}) = \operatorname{range} \begin{array}{c} k \\ \left[\begin{array}{c} \mathbf{I} \\ \mathbf{0} \end{array}\right] \\ \end{array} \begin{array}{c} \\ \begin{array}{c} k \\ n-k \end{array} \end{array}.$$

The matrix on the right-hand side has full column rank, so it is legal to apply formula (8.3) for an orthogonal projector, which immediately yields

$$\boldsymbol{P}_{\boldsymbol{W}} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \quad \text{and} \quad \mathbf{I} - \boldsymbol{P}_{\boldsymbol{W}} = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}. \tag{9.7}$$

In other words, the range of $\boldsymbol{W}$ aligns with the first k coordinates, which span the same subspace as the first k left singular vectors of the auxiliary input matrix $\widetilde{\boldsymbol{A}}$. Therefore, range($\boldsymbol{W}$) captures the action of $\widetilde{\boldsymbol{A}}$, which is what we wanted from range($\widetilde{\boldsymbol{Y}}$).

We treat the auxiliary sample matrix $\widetilde{\boldsymbol{Y}}$ as a perturbation of $\boldsymbol{W}$, and we hope that their ranges are close to each other. To make the comparison rigorous, let us emulate the arguments outlined in the last paragraph. Referring to the display (9.4), we flatten out the top block of $\widetilde{\boldsymbol{Y}}$ to obtain the matrix

$$\boldsymbol{Z} = \widetilde{\boldsymbol{Y}} \cdot \boldsymbol{\Omega}_1^\dagger \boldsymbol{\Sigma}_1^{-1} = \begin{bmatrix} \mathbf{I} \\ \boldsymbol{F} \end{bmatrix}, \quad \text{where} \quad \boldsymbol{F} = \boldsymbol{\Sigma}_2 \boldsymbol{\Omega}_2 \boldsymbol{\Omega}_1^\dagger \boldsymbol{\Sigma}_1^{-1}. \tag{9.8}$$

Let us return to the error bound (9.6). The construction (9.8) ensures that range($\boldsymbol{Z}$) $\subset$ range($\widetilde{\boldsymbol{Y}}$), so Proposition 8.5 implies that the error satisfies

$$\left\|(\mathbf{I} - \boldsymbol{P}_{\widetilde{\boldsymbol{Y}}})\widetilde{\boldsymbol{A}}\right\| \le \left\|(\mathbf{I} - \boldsymbol{P}_{\boldsymbol{Z}})\widetilde{\boldsymbol{A}}\right\|.$$

Squaring this relation, we obtain

$$\left\|(\mathbf{I} - \boldsymbol{P}_{\widetilde{\boldsymbol{Y}}})\widetilde{\boldsymbol{A}}\right\|^2 \le \left\|(\mathbf{I} - \boldsymbol{P}_{\boldsymbol{Z}})\widetilde{\boldsymbol{A}}\right\|^2 = \left\|\widetilde{\boldsymbol{A}}^*(\mathbf{I} - \boldsymbol{P}_{\boldsymbol{Z}})\widetilde{\boldsymbol{A}}\right\| = \left\|\boldsymbol{\Sigma}^*(\mathbf{I} - \boldsymbol{P}_{\boldsymbol{Z}})\boldsymbol{\Sigma}\right\|. \tag{9.9}$$

The last identity follows from the definition $\widetilde{\boldsymbol{A}} = \boldsymbol{\Sigma}\boldsymbol{V}^*$ and the unitary invariance of the spectral norm. Therefore, we can complete the proof of (9.6) by producing a suitable bound for the right-hand side of (9.9).

To continue, we need a detailed representation of the projector $\mathbf{I} - \boldsymbol{P}_{\boldsymbol{Z}}$. The construction (9.8) ensures that $\boldsymbol{Z}$ has full column rank, so we can apply formula (8.3) for an orthogonal projector to see that

$$\boldsymbol{P}_{\boldsymbol{Z}} = \boldsymbol{Z}(\boldsymbol{Z}^*\boldsymbol{Z})^{-1}\boldsymbol{Z}^* = \begin{bmatrix} \mathbf{I} \\ \boldsymbol{F} \end{bmatrix} (\mathbf{I} + \boldsymbol{F}^*\boldsymbol{F})^{-1} \begin{bmatrix} \mathbf{I} \\ \boldsymbol{F} \end{bmatrix}^*.$$

Expanding this expression, we determine that the complementary projector satisfies

$$\mathbf{I} - \boldsymbol{P}_{\boldsymbol{Z}} = \begin{bmatrix} \mathbf{I} - (\mathbf{I} + \boldsymbol{F}^*\boldsymbol{F})^{-1} & -(\mathbf{I} + \boldsymbol{F}^*\boldsymbol{F})^{-1}\boldsymbol{F}^* \\ -\boldsymbol{F}(\mathbf{I} + \boldsymbol{F}^*\boldsymbol{F})^{-1} & \mathbf{I} - \boldsymbol{F}(\mathbf{I} + \boldsymbol{F}^*\boldsymbol{F})^{-1}\boldsymbol{F}^* \end{bmatrix}. \tag{9.10}$$

The partitioning here conforms with the partitioning of $\boldsymbol{\Sigma}$. When we conjugate the matrix by $\boldsymbol{\Sigma}$, copies of $\boldsymbol{\Sigma}_1^{-1}$, presently hidden in the top-left block, will cancel to happy effect.

The latter point may not seem obvious, owing to the complicated form of (9.10). In reality, the block matrix is less fearsome than it looks. Proposition 8.2, on the perturbation of inverses, shows that the top-left block verifies

$$\mathbf{I} - (\mathbf{I} + \boldsymbol{F}^*\boldsymbol{F})^{-1} \preccurlyeq \boldsymbol{F}^*\boldsymbol{F}.$$

The bottom-right block satisfies

$$\mathbf{I} - \boldsymbol{F}(\mathbf{I} + \boldsymbol{F}^*\boldsymbol{F})^{-1}\boldsymbol{F}^* \preccurlyeq \mathbf{I}$$

because the conjugation rule guarantees that $\boldsymbol{F}(\mathbf{I} + \boldsymbol{F}^*\boldsymbol{F})^{-1}\boldsymbol{F}^* \succcurlyeq \mathbf{0}$. We abbreviate the off-diagonal blocks with the symbol $\boldsymbol{B} = -(\mathbf{I} + \boldsymbol{F}^*\boldsymbol{F})^{-1}\boldsymbol{F}^*$. In summary,

$$\mathbf{I} - \boldsymbol{P}_{\boldsymbol{Z}} \preccurlyeq \begin{bmatrix} \boldsymbol{F}^*\boldsymbol{F} & \boldsymbol{B} \\ \boldsymbol{B}^* & \mathbf{I} \end{bmatrix}.$$

This relation exposes the key structural properties of the projector. Compare this relation with the expression (9.7) for the "ideal" projector $\mathbf{I} - \boldsymbol{P}_{\boldsymbol{W}}$.

Moving toward the estimate required by (9.9), we conjugate the last relation by $\boldsymbol{\Sigma}$ to obtain

$$\boldsymbol{\Sigma}^*(\mathbf{I} - \boldsymbol{P}_{\boldsymbol{Z}})\boldsymbol{\Sigma} \preccurlyeq \begin{bmatrix} \boldsymbol{\Sigma}_1^*\boldsymbol{F}^*\boldsymbol{F}\boldsymbol{\Sigma}_1 & \boldsymbol{\Sigma}_1^*\boldsymbol{B}\boldsymbol{\Sigma}_2 \\ \boldsymbol{\Sigma}_2^*\boldsymbol{B}^*\boldsymbol{\Sigma}_1 & \boldsymbol{\Sigma}_2^*\boldsymbol{\Sigma}_2 \end{bmatrix}.$$

The conjugation rule demonstrates that the matrix on the left-hand side is psd, so the matrix on the right-hand side is too. Proposition 8.3 results in the norm bound

$$\|\boldsymbol{\Sigma}^*(\mathbf{I} - \boldsymbol{P}_{\boldsymbol{Z}})\boldsymbol{\Sigma}\| \le \|\boldsymbol{\Sigma}_1^*\boldsymbol{F}^*\boldsymbol{F}\boldsymbol{\Sigma}_1\| + \|\boldsymbol{\Sigma}_2^*\boldsymbol{\Sigma}_2\| = \|\boldsymbol{F}\boldsymbol{\Sigma}_1\|^2 + \|\boldsymbol{\Sigma}_2\|^2.$$

Recall that $\boldsymbol{F} = \boldsymbol{\Sigma}_2\boldsymbol{\Omega}_2\boldsymbol{\Omega}_1^\dagger\boldsymbol{\Sigma}_1^{-1}$, so the factor $\boldsymbol{\Sigma}_1$ cancels neatly. Therefore,

$$\|\boldsymbol{\Sigma}^*(\mathbf{I} - \boldsymbol{P}_{\boldsymbol{Z}})\boldsymbol{\Sigma}\| \le \left\|\boldsymbol{\Sigma}_2\boldsymbol{\Omega}_2\boldsymbol{\Omega}_1^\dagger\right\|^2 + \|\boldsymbol{\Sigma}_2\|^2.$$

Finally, introduce the latter inequality into (9.9) to complete the proof. ∎

9.3. Analysis of the Power Scheme. Theorem 9.1 suggests that the performance of the proto-algorithm depends strongly on the relationship between the large singular values of $\boldsymbol{A}$ listed in $\boldsymbol{\Sigma}_1$ and the small singular values listed in $\boldsymbol{\Sigma}_2$. When a substantial proportion of the mass of $\boldsymbol{A}$ appears in the small singular values, the constructed basis $\boldsymbol{Q}$ may have low accuracy. Conversely, when the large singular values dominate, it is much easier to identify a good low-rank basis.

To improve the performance of the proto-algorithm, we can run it with a closely related input matrix whose singular values decay more rapidly [67, 113]. Fix a positive integer q, and set

$$\boldsymbol{B} = (\boldsymbol{A}\boldsymbol{A}^*)^q\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}^{2q+1}\boldsymbol{V}^*.$$

We apply the proto-algorithm to $\boldsymbol{B}$, which generates a sample matrix $\boldsymbol{Z} = \boldsymbol{B}\boldsymbol{\Omega}$ and constructs a basis $\boldsymbol{Q}$ for the range of $\boldsymbol{Z}$. Section 4.5 elaborates on the implementation details and describes a reformulation that sometimes improves the accuracy when the scheme is executed in finite-precision arithmetic. The following result describes how well we can approximate the *original* matrix $\boldsymbol{A}$ within the range of $\boldsymbol{Z}$.

THEOREM 9.2 (power scheme). *Let $\boldsymbol{A}$ be an $m \times n$ matrix, and let $\boldsymbol{\Omega}$ be an $n \times \ell$ matrix. Fix a nonnegative integer q, form $\boldsymbol{B} = (\boldsymbol{A}^*\boldsymbol{A})^q\boldsymbol{A}$, and compute the sample matrix $\boldsymbol{Z} = \boldsymbol{B}\boldsymbol{\Omega}$. Then*

$$\|(\mathbf{I} - \boldsymbol{P}_{\boldsymbol{Z}})\boldsymbol{A}\| \le \|(\mathbf{I} - \boldsymbol{P}_{\boldsymbol{Z}})\boldsymbol{B}\|^{1/(2q+1)}.$$

Proof. We determine that

$$\|(\mathbf{I} - \boldsymbol{P}_{\boldsymbol{Z}})\boldsymbol{A}\| \le \|(\mathbf{I} - \boldsymbol{P}_{\boldsymbol{Z}})(\boldsymbol{A}\boldsymbol{A}^*)^q\boldsymbol{A}\|^{1/(2q+1)} = \|(\mathbf{I} - \boldsymbol{P}_{\boldsymbol{Z}})\boldsymbol{B}\|^{1/(2q+1)}$$

as a direct consequence of Proposition 8.6. ∎

Let us illustrate how the power scheme interacts with the main error bound (9.3). Let σ_{k+1} denote the $(k+1)$th singular value of $\boldsymbol{A}$. First, suppose we approximate $\boldsymbol{A}$ in the range of the sample matrix $\boldsymbol{Y} = \boldsymbol{A}\boldsymbol{\Omega}$. Since $\|\boldsymbol{\Sigma}_2\| = \sigma_{k+1}$, Theorem 9.1 implies that

$$\|(\mathbf{I} - \boldsymbol{P}_{\boldsymbol{Y}})\boldsymbol{A}\| \le \left(1 + \left\|\boldsymbol{\Omega}_2\boldsymbol{\Omega}_1^\dagger\right\|^2\right)^{1/2}\sigma_{k+1}. \tag{9.11}$$

Now, define $\boldsymbol{B} = (\boldsymbol{A}\boldsymbol{A}^*)^q \boldsymbol{A}$, and suppose we approximate $\boldsymbol{A}$ within the range of the sample matrix $\boldsymbol{Z} = \boldsymbol{B}\boldsymbol{\Omega}$. Together, Theorems 9.2 and 9.1 imply that

$$\|(\mathbf{I} - \boldsymbol{P}_{\boldsymbol{Z}})\boldsymbol{A}\| \le \|(\mathbf{I} - \boldsymbol{P}_{\boldsymbol{Z}})\boldsymbol{B}\|^{1/(2q+1)} \le \left(1 + \left\|\boldsymbol{\Omega}_2 \boldsymbol{\Omega}_1^{\dagger}\right\|^2\right)^{1/(4q+2)} \sigma_{k+1}$$

because σ_{k+1}^{2q+1} is the $(k+1)$th singular value of $\boldsymbol{B}$. In effect, the power scheme drives down the suboptimality of the bound (9.11) exponentially fast as the power q increases. In principle, we can make the extra factor as close to one as we like, although this increases the cost of the algorithm.

9.4. Analysis of Truncated SVD. Finally, let us study the truncated SVD described in Remark 5.1. Suppose that we approximate the input matrix $\boldsymbol{A}$ inside the range of the sample matrix $\boldsymbol{Z}$. In essence, the truncation step computes a best rank-k approximation $\widehat{\boldsymbol{A}}_{(k)}$ of the compressed matrix $\boldsymbol{P}_{\boldsymbol{Z}}\boldsymbol{A}$. The next result provides a simple error bound for this method; this argument was proposed by Ming Gu.

THEOREM 9.3 (analysis of truncated SVD). *Let $\boldsymbol{A}$ be an $m \times n$ matrix with singular values $\sigma_1 \ge \sigma_2 \ge \sigma_3 \ge \cdots$, and let $\boldsymbol{Z}$ be an $m \times \ell$ matrix, where $\ell \ge k$. Suppose that $\widehat{\boldsymbol{A}}_{(k)}$ is a best rank-k approximation of $\boldsymbol{P}_{\boldsymbol{Z}}\boldsymbol{A}$ with respect to the spectral norm. Then*

$$\left\|\boldsymbol{A} - \widehat{\boldsymbol{A}}_{(k)}\right\| \le \sigma_{k+1} + \|(\mathbf{I} - \boldsymbol{P}_{\boldsymbol{Z}})\boldsymbol{A}\|.$$

Proof. Apply the triangle inequality to split the error into two components.

$$\left\|\boldsymbol{A} - \widehat{\boldsymbol{A}}_k\right\| \le \left\|\boldsymbol{A} - \boldsymbol{P}_{\boldsymbol{Z}}\boldsymbol{A}\right\| + \left\|\boldsymbol{P}_{\boldsymbol{Z}}\boldsymbol{A} - \widehat{\boldsymbol{A}}_{(k)}\right\|. \tag{9.12}$$

We have already developed a detailed theory for estimating the first term. To analyze the second term, we introduce a best rank-k approximation $\boldsymbol{A}_{(k)}$ of the matrix $\boldsymbol{A}$. Note that

$$\left\|\boldsymbol{P}_{\boldsymbol{Z}}\boldsymbol{A} - \widehat{\boldsymbol{A}}_{(k)}\right\| \le \left\|\boldsymbol{P}_{\boldsymbol{Z}}\boldsymbol{A} - \boldsymbol{P}_{\boldsymbol{Z}}\boldsymbol{A}_{(k)}\right\|$$

because $\widehat{\boldsymbol{A}}_{(k)}$ is a best rank-k approximation to the matrix $\boldsymbol{P}_{\boldsymbol{Z}}\boldsymbol{A}$, whereas $\boldsymbol{P}_{\boldsymbol{Z}}\boldsymbol{A}_{(k)}$ is an undistinguished rank-k matrix. It follows that

$$\left\|\boldsymbol{P}_{\boldsymbol{Z}}\boldsymbol{A} - \widehat{\boldsymbol{A}}_{(k)}\right\| \le \left\|\boldsymbol{P}_{\boldsymbol{Z}}(\boldsymbol{A} - \boldsymbol{A}_{(k)})\right\| \le \left\|\boldsymbol{A} - \boldsymbol{A}_{(k)}\right\| = \sigma_{k+1}. \tag{9.13}$$

The second inequality holds because the orthogonal projector is a contraction; the last identity follows from Mirsky's theorem [98]. Combine (9.12) and (9.13) to reach the main result. ∎

Remark 9.1. In the randomized setting, the truncation step appears to be less damaging than the error bound of Theorem 9.3 suggests, but we currently lack a complete theoretical understanding of its behavior.

10. Gaussian Test Matrices. The error bound in Theorem 9.1 shows that the performance of the proto-algorithm depends on the interaction between the test matrix $\boldsymbol{\Omega}$ and the right singular vectors of the input matrix $\boldsymbol{A}$. Algorithm 4.1 is a particularly simple version of the proto-algorithm that draws the test matrix according to the standard Gaussian distribution. The literature contains a wealth of information about these matrices, which allows us to perform a very precise error analysis.

We focus on the real case in this section. Analogous results hold in the complex case, where the algorithm even exhibits superior performance.

10.1. Technical Background. A *standard Gaussian matrix* is a random matrix whose entries are independent standard normal variables. The distribution of a standard Gaussian matrix is rotationally invariant: If $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthonormal matrices, then $\boldsymbol{U}^*\boldsymbol{G}\boldsymbol{V}$ also has the standard Gaussian distribution.

Our analysis requires detailed information about the properties of Gaussian matrices. In particular, we must understand how the norm of a Gaussian matrix and its pseudoinverse vary. We summarize the relevant results and citations here, reserving the details for the appendix.

PROPOSITION 10.1 (expected norm of a scaled Gaussian matrix). *Fix matrices $\boldsymbol{S}, \boldsymbol{T}$, and draw a standard Gaussian matrix $\boldsymbol{G}$. Then*

$$\left(\mathbb{E}\,\|\boldsymbol{S}\boldsymbol{G}\boldsymbol{T}\|_{\mathrm{F}}^2\right)^{1/2} = \|\boldsymbol{S}\|_{\mathrm{F}}\,\|\boldsymbol{T}\|_{\mathrm{F}} \quad \textit{and} \tag{10.1}$$

$$\mathbb{E}\,\|\boldsymbol{S}\boldsymbol{G}\boldsymbol{T}\| \le \|\boldsymbol{S}\|\,\|\boldsymbol{T}\|_{\mathrm{F}} + \|\boldsymbol{S}\|_{\mathrm{F}}\,\|\boldsymbol{T}\|\,. \tag{10.2}$$

The identity (10.1) follows from a direct calculation. The second bound (10.2) relies on methods developed by Gordon [62, 63]. See Propositions A.1 and A.2.

PROPOSITION 10.2 (expected norm of a pseudoinverted Gaussian matrix). *Draw a $k \times (k+p)$ standard Gaussian matrix $\boldsymbol{G}$ with $k \ge 2$ and $p \ge 2$. Then*

$$\left(\mathbb{E}\,\left\|\boldsymbol{G}^\dagger\right\|_{\mathrm{F}}^2\right)^{1/2} = \sqrt{\frac{k}{p-1}} \quad \textit{and} \tag{10.3}$$

$$\mathbb{E}\,\left\|\boldsymbol{G}^\dagger\right\| \le \frac{\mathrm{e}\sqrt{k+p}}{p}. \tag{10.4}$$

The first identity is a standard result from multivariate statistics [100, p. 96]. The second follows from work of Chen and Dongarra [25]. See Propositions A.4 and A.6.

To study the probability that Algorithm 4.1 produces a large error, we rely on tail bounds for functions of Gaussian matrices. The next proposition rephrases a well-known result on concentration of measure [14, Thm. 4.5.7]. See also [84, sect. 1.1] and [83, sect. 5.1].

PROPOSITION 10.3 (concentration for functions of a Gaussian matrix). *Suppose that h is a Lipschitz function on matrices:*

$$|h(\boldsymbol{X}) - h(\boldsymbol{Y})| \le L\,\|\boldsymbol{X} - \boldsymbol{Y}\|_{\mathrm{F}} \quad \textit{for all } \boldsymbol{X}, \boldsymbol{Y}.$$

Draw a standard Gaussian matrix $\boldsymbol{G}$. Then

$$\mathbb{P}\left\{h(\boldsymbol{G}) \ge \mathbb{E}\,h(\boldsymbol{G}) + Lt\right\} \le \mathrm{e}^{-t^2/2}.$$

Finally, we state some large deviation bounds for the norm of a pseudoinverted Gaussian matrix.

PROPOSITION 10.4 (norm bounds for a pseudoinverted Gaussian matrix). *Let $\boldsymbol{G}$ be a $k \times (k+p)$ Gaussian matrix where $p \ge 4$. For all $t \ge 1$,*

$$\mathbb{P}\left\{\left\|\boldsymbol{G}^\dagger\right\|_{\mathrm{F}} \ge \sqrt{\frac{3k}{p+1}} \cdot t\right\} \le t^{-p} \quad \textit{and} \tag{10.5}$$

$$\mathbb{P}\left\{\left\|\boldsymbol{G}^\dagger\right\| \ge \frac{\mathrm{e}\sqrt{k+p}}{p+1} \cdot t\right\} \le t^{-(p+1)}. \tag{10.6}$$

Compare these estimates with Proposition 10.2. It seems that (10.5) is new; we were unable to find a comparable analysis in the random matrix literature. Although the form of (10.5) is not optimal, it allows us to produce more transparent results than a fully detailed estimate. The bound (10.6) essentially appears in the work of Chen and Dongarra [25]. See Proposition A.3 and Theorem A.7 for more information.

10.2. Average-Case Analysis of Algorithm 4.1. We separate our analysis into two pieces. First, we present information about expected values. In the next subsection, we describe bounds on the probability of a large deviation.

We begin with the simplest result, which provides an estimate for the expected approximation error in the Frobenius norm. All proofs are postponed to the end of the section.

THEOREM 10.5 (average Frobenius error). *Suppose that* $\boldsymbol{A}$ *is a* real $m \times n$ *matrix with singular values* $\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq \cdots$. *Choose a target rank* $k \geq 2$ *and an oversampling parameter* $p \geq 2$, *where* $k + p \leq \min\{m, n\}$. *Draw an* $n \times (k + p)$ *standard Gaussian matrix* $\boldsymbol{\Omega}$, *and construct the sample matrix* $\boldsymbol{Y} = \boldsymbol{A\Omega}$. *Then the expected approximation error*

$$\mathbb{E}\,\|(\mathbf{I} - \boldsymbol{P_Y})\boldsymbol{A}\|_{\mathrm{F}} \leq \left(1 + \frac{k}{p-1}\right)^{1/2} \left(\sum\nolimits_{j>k} \sigma_j^2\right)^{1/2}.$$

This theorem predicts several intriguing behaviors of Algorithm 4.1. The Eckart–Young theorem [54] shows that $(\sum_{j>k} \sigma_j^2)^{1/2}$ is the minimal Frobenius-norm error when approximating $\boldsymbol{A}$ with a rank-k matrix. This quantity is the appropriate benchmark for the performance of the algorithm. If the small singular values of $\boldsymbol{A}$ are very flat, the series may be as large as $\sigma_{k+1}\sqrt{\min\{m, n\} - k}$. On the other hand, when the singular values exhibit some decay, the error may be on the same order as σ_{k+1}.

The error bound always exceeds this baseline error, but it may be polynomially larger, depending on the ratio between the target rank k and the oversampling parameter p. For p small (say, less than five), the error is somewhat variable because the small singular values of a nearly square Gaussian matrix are very unstable. As the oversampling increases, the performance improves quickly. When $p \sim k$, the error is already within a constant factor of the baseline.

The error bound for the spectral norm is somewhat more complicated, but it reveals some interesting new features.

THEOREM 10.6 (average spectral error). *Under the hypotheses of Theorem* 10.5,

$$\mathbb{E}\,\|(\mathbf{I} - \boldsymbol{P_Y})\boldsymbol{A}\| \leq \left(1 + \sqrt{\frac{k}{p-1}}\right) \sigma_{k+1} + \frac{\mathrm{e}\sqrt{k+p}}{p} \left(\sum\nolimits_{j>k} \sigma_j^2\right)^{1/2}.$$

Mirsky [98] has shown that the quantity σ_{k+1} is the minimum spectral-norm error when approximating $\boldsymbol{A}$ with a rank-k matrix, so the first term in Theorem 10.6 is analogous with the error bound in Theorem 10.5. The second term represents a new phenomenon: we also pay for the Frobenius-norm error in approximating $\boldsymbol{A}$. Note that as the amount p of oversampling increases, the polynomial factor in the second term declines much more quickly than the factor in the first term. When $p \sim k$, the factor on the σ_{k+1} term is constant, while the factor on the series has order $k^{-1/2}$.

We also note that the bound in Theorem 10.6 implies

$$\mathbb{E}\,\|(\mathbf{I} - \boldsymbol{P_Y})\boldsymbol{A}\| \leq \left[1 + \sqrt{\frac{k}{p-1}} + \frac{\mathrm{e}\sqrt{k+p}}{p} \cdot \sqrt{\min\{m, n\} - k}\right] \sigma_{k+1},$$

so the average spectral-norm error always lies within a small polynomial factor of the baseline σ_{k+1}.

Let us continue with the proofs of these results.

Proof of Theorem 10.5. Let $\boldsymbol{V}$ be the right unitary factor of $\boldsymbol{A}$. Partition $\boldsymbol{V} = [\boldsymbol{V}_1 \mid \boldsymbol{V}_2]$ into blocks containing, respectively, k and $n-k$ columns. Recall that

$$\boldsymbol{\Omega}_1 = \boldsymbol{V}_1^* \boldsymbol{\Omega} \quad \text{and} \quad \boldsymbol{\Omega}_2 = \boldsymbol{V}_2^* \boldsymbol{\Omega}.$$

The Gaussian distribution is rotationally invariant, so $\boldsymbol{V}^*\boldsymbol{\Omega}$ is also a standard Gaussian matrix. Observe that $\boldsymbol{\Omega}_1$ and $\boldsymbol{\Omega}_2$ are *nonoverlapping* submatrices of $\boldsymbol{V}^*\boldsymbol{\Omega}$, so these two matrices are not only standard Gaussian but also stochastically independent. Furthermore, the rows of a (fat) Gaussian matrix are almost surely in general position, so the $k \times (k+p)$ matrix $\boldsymbol{\Omega}_1$ has full row rank with probability one.

Hölder's inequality and Theorem 9.1 together imply that

$$\mathbb{E}\,\|(\mathbf{I} - \boldsymbol{P}_{\boldsymbol{Y}})\boldsymbol{A}\|_{\mathrm{F}} \leq \left(\mathbb{E}\,\|(\mathbf{I} - \boldsymbol{P}_{\boldsymbol{Y}})\boldsymbol{A}\|_{\mathrm{F}}^2\right)^{1/2} \leq \left(\|\boldsymbol{\Sigma}_2\|_{\mathrm{F}}^2 + \mathbb{E}\,\|\boldsymbol{\Sigma}_2\boldsymbol{\Omega}_2\boldsymbol{\Omega}_1^\dagger\|_{\mathrm{F}}^2\right)^{1/2}.$$

We compute this expectation by conditioning on the value of $\boldsymbol{\Omega}_1$ and applying Proposition 10.1 to the scaled Gaussian matrix $\boldsymbol{\Omega}_2$. Thus,

$$\begin{aligned}\mathbb{E}\,\|\boldsymbol{\Sigma}_2\boldsymbol{\Omega}_2\boldsymbol{\Omega}_1^\dagger\|_{\mathrm{F}}^2 = \mathbb{E}\left(\mathbb{E}\left[\|\boldsymbol{\Sigma}_2\boldsymbol{\Omega}_2\boldsymbol{\Omega}_1^\dagger\|_{\mathrm{F}}^2 \mid \boldsymbol{\Omega}_1\right]\right) &= \mathbb{E}\left(\|\boldsymbol{\Sigma}_2\|_{\mathrm{F}}^2\,\|\boldsymbol{\Omega}_1^\dagger\|_{\mathrm{F}}^2\right) \\ &= \|\boldsymbol{\Sigma}_2\|_{\mathrm{F}}^2 \cdot \mathbb{E}\,\|\boldsymbol{\Omega}_1^\dagger\|_{\mathrm{F}}^2 = \frac{k}{p-1} \cdot \|\boldsymbol{\Sigma}_2\|_{\mathrm{F}}^2,\end{aligned}$$

where the last expectation follows from relation (10.3) of Proposition 10.2. In summary,

$$\mathbb{E}\,\|(\mathbf{I} - \boldsymbol{P}_{\boldsymbol{Y}})\boldsymbol{A}\|_{\mathrm{F}} \leq \left(1 + \frac{k}{p-1}\right)^{1/2} \|\boldsymbol{\Sigma}_2\|_{\mathrm{F}}.$$

Observe that $\|\boldsymbol{\Sigma}_2\|_{\mathrm{F}}^2 = \sum_{j>k} \sigma_j^2$ to complete the proof. ∎

Proof of Theorem 10.6. The argument is similar to the proof of Theorem 10.5. First, Theorem 9.1 implies that

$$\mathbb{E}\,\|(\mathbf{I} - \boldsymbol{P}_{\boldsymbol{Y}})\boldsymbol{A}\| \leq \mathbb{E}\left(\|\boldsymbol{\Sigma}_2\|^2 + \|\boldsymbol{\Sigma}_2\boldsymbol{\Omega}_2\boldsymbol{\Omega}_1^\dagger\|^2\right)^{1/2} \leq \|\boldsymbol{\Sigma}_2\| + \mathbb{E}\,\|\boldsymbol{\Sigma}_2\boldsymbol{\Omega}_2\boldsymbol{\Omega}_1^\dagger\|.$$

We condition on $\boldsymbol{\Omega}_1$ and apply Proposition 10.1 to bound the expectation with respect to $\boldsymbol{\Omega}_2$. Thus,

$$\begin{aligned}\mathbb{E}\,\|\boldsymbol{\Sigma}_2\boldsymbol{\Omega}_2\boldsymbol{\Omega}_1^\dagger\| &\leq \mathbb{E}\left(\|\boldsymbol{\Sigma}_2\|\,\|\boldsymbol{\Omega}_1^\dagger\|_{\mathrm{F}} + \|\boldsymbol{\Sigma}_2\|_{\mathrm{F}}\,\|\boldsymbol{\Omega}_1^\dagger\|\right) \\ &\leq \|\boldsymbol{\Sigma}_2\|\left(\mathbb{E}\,\|\boldsymbol{\Omega}_1^\dagger\|_{\mathrm{F}}^2\right)^{1/2} + \|\boldsymbol{\Sigma}_2\|_{\mathrm{F}} \cdot \mathbb{E}\,\|\boldsymbol{\Omega}_1^\dagger\|,\end{aligned}$$

where the second relation requires Hölder's inequality. Applying both parts of Proposition 10.2, we obtain

$$\mathbb{E}\,\|\boldsymbol{\Sigma}_2\boldsymbol{\Omega}_2\boldsymbol{\Omega}_1^\dagger\| \leq \sqrt{\frac{k}{p-1}}\,\|\boldsymbol{\Sigma}_2\| + \frac{\mathrm{e}\sqrt{k+p}}{p}\,\|\boldsymbol{\Sigma}_2\|_{\mathrm{F}}.$$

Note that $\|\boldsymbol{\Sigma}_2\| = \sigma_{k+1}$ to wrap up. ∎

10.3. Probabilistic Error Bounds for Algorithm 4.1. We can develop tail bounds for the approximation error, which demonstrate that the average performance of the algorithm is representative of the actual performance. We begin with the Frobenius norm because the result is somewhat simpler.

THEOREM 10.7 (deviation bounds for the Frobenius error). *Frame the hypotheses of Theorem* 10.5. *Assume further that* $p \geq 4$. *For all* $u, t \geq 1$,

$$\|(\mathbf{I} - \boldsymbol{P}_{\boldsymbol{Y}})\boldsymbol{A}\|_{\mathrm{F}} \leq \left(1 + t \cdot \sqrt{\frac{3k}{p+1}}\right) \left(\sum\nolimits_{j>k} \sigma_j^2\right)^{1/2} + ut \cdot \frac{\mathrm{e}\sqrt{k+p}}{p+1} \cdot \sigma_{k+1},$$

with failure probability at most $2t^{-p} + \mathrm{e}^{-u^2/2}$.

To parse this theorem, observe that the first term in the error bound corresponds with the expected approximation error in Theorem 10.5. The second term represents a deviation above the mean.

An analogous result holds for the spectral norm.

THEOREM 10.8 (deviation bounds for the spectral error). *Frame the hypotheses of Theorem* 10.5. *Assume further that* $p \geq 4$. *For all* $u, t \geq 1$,

$$\begin{aligned}&\|(\mathbf{I} - \boldsymbol{P}_{\boldsymbol{Y}})\boldsymbol{A}\| \\ &\leq \left[\left(1 + t \cdot \sqrt{\frac{3k}{p+1}}\right)\sigma_{k+1} + t \cdot \frac{\mathrm{e}\sqrt{k+p}}{p+1}\left(\sum\nolimits_{j>k} \sigma_j^2\right)^{1/2}\right] + ut \cdot \frac{\mathrm{e}\sqrt{k+p}}{p+1}\sigma_{k+1},\end{aligned}$$

with failure probability at most $2t^{-p} + \mathrm{e}^{-u^2/2}$.

The bracket corresponds with the expected spectral-norm error, while the remaining term represents a deviation above the mean. Neither the numerical constants nor the precise form of the bound is optimal because of the slackness in Proposition 10.4. Nevertheless, the theorem gives a fairly good picture of what is actually happening.

We acknowledge that the current form of Theorem 10.8 is complicated. To produce more transparent results, we make appropriate selections for the parameters u, t and bound the numerical constants.

COROLLARY 10.9 (simplified deviation bounds for the spectral error). *Frame the hypotheses of Theorem* 10.5, *and assume further that* $p \geq 4$. *Then*

$$\|(\mathbf{I} - \boldsymbol{P}_{\boldsymbol{Y}})\boldsymbol{A}\| \leq \left(1 + 16\sqrt{1 + \frac{k}{p+1}}\right)\sigma_{k+1} + \frac{8\sqrt{k+p}}{p+1}\left(\sum\nolimits_{j>k} \sigma_j^2\right)^{1/2},$$

with failure probability at most $3\mathrm{e}^{-p}$. *Moreover,*

$$\|(\mathbf{I} - \boldsymbol{P}_{\boldsymbol{Y}})\boldsymbol{A}\| \leq \left(1 + 6\sqrt{(k+p) \cdot p \log p}\right)\sigma_{k+1} + 3\sqrt{k+p}\left(\sum\nolimits_{j>k} \sigma_j^2\right)^{1/2},$$

with failure probability at most $3p^{-p}$.

Proof. The first part of the result follows from the choices $t = \mathrm{e}$ and $u = \sqrt{2p}$, and the second emerges when $t = p$ and $u = \sqrt{2p \log p}$. Another interesting parameter selection is $t = p^{c/p}$ and $u = \sqrt{2c \log p}$, which yields a failure probability $3p^{-c}$. ∎

Corollary 10.9 should be compared with [92, Obs. 4.4–4.5]. Although our result contains sharper error estimates, the failure probabilities are slightly worse. The error bound (1.9) presented in section 1.5 follows after further simplification of the second bound from Corollary 10.9.

We continue with a proof of Theorem 10.8. The same argument can be used to obtain a bound for the Frobenius-norm error, but we omit a detailed account.

Proof of Theorem 10.8. Since $\boldsymbol{\Omega}_1$ and $\boldsymbol{\Omega}_2$ are independent from each other, we can study how the error depends on the matrix $\boldsymbol{\Omega}_2$ by conditioning on the event that $\boldsymbol{\Omega}_1$ is not too irregular. To that end, we define a (parameterized) event on which the spectral and Frobenius norms of the matrix $\boldsymbol{\Omega}_1^\dagger$ are both controlled. For $t \geq 1$, let

$$E_t = \left\{ \boldsymbol{\Omega}_1 : \left\|\boldsymbol{\Omega}_1^\dagger\right\| \leq \frac{\mathrm{e}\sqrt{k+p}}{p+1} \cdot t \quad \text{and} \quad \left\|\boldsymbol{\Omega}_1^\dagger\right\|_{\mathrm{F}} \leq \sqrt{\frac{3k}{p+1}} \cdot t \right\}.$$

Invoking both parts of Proposition 10.4, we find that

$$\mathbb{P}\left(E_t^c\right) \leq t^{-(p+1)} + t^{-p} \leq 2t^{-p}.$$

Consider the function $h(\boldsymbol{X}) = \left\|\boldsymbol{\Sigma}_2 \boldsymbol{X} \boldsymbol{\Omega}_1^\dagger\right\|$. We quickly compute its Lipschitz constant L with the lower triangle inequality and some standard norm estimates:

$$|h(\boldsymbol{X}) - h(\boldsymbol{Y})| \leq \left\|\boldsymbol{\Sigma}_2(\boldsymbol{X}-\boldsymbol{Y})\boldsymbol{\Omega}_1^\dagger\right\| \leq \|\boldsymbol{\Sigma}_2\| \, \|\boldsymbol{X}-\boldsymbol{Y}\| \, \left\|\boldsymbol{\Omega}_1^\dagger\right\| \leq \|\boldsymbol{\Sigma}_2\| \, \left\|\boldsymbol{\Omega}_1^\dagger\right\| \, \|\boldsymbol{X}-\boldsymbol{Y}\|_{\mathrm{F}}.$$

Therefore, $L \leq \|\boldsymbol{\Sigma}_2\| \, \left\|\boldsymbol{\Omega}_1^\dagger\right\|$. Relation (10.2) of Proposition 10.1 implies that

$$\mathbb{E}[h(\boldsymbol{\Omega}_2) \mid \boldsymbol{\Omega}_1] \leq \|\boldsymbol{\Sigma}_2\| \, \left\|\boldsymbol{\Omega}_1^\dagger\right\|_{\mathrm{F}} + \|\boldsymbol{\Sigma}_2\|_{\mathrm{F}} \, \left\|\boldsymbol{\Omega}_1^\dagger\right\|.$$

Applying the concentration of measure inequality, Proposition 10.3, conditionally to the random variable $h(\boldsymbol{\Omega}_2) = \left\|\boldsymbol{\Sigma}_2\boldsymbol{\Omega}_2\boldsymbol{\Omega}_1^\dagger\right\|$ results in

$$\mathbb{P}\left\{ \left\|\boldsymbol{\Sigma}_2\boldsymbol{\Omega}_2\boldsymbol{\Omega}_1^\dagger\right\| > \|\boldsymbol{\Sigma}_2\| \, \left\|\boldsymbol{\Omega}_1^\dagger\right\|_{\mathrm{F}} + \|\boldsymbol{\Sigma}_2\|_{\mathrm{F}} \, \left\|\boldsymbol{\Omega}_1^\dagger\right\| + \|\boldsymbol{\Sigma}_2\| \, \left\|\boldsymbol{\Omega}_1^\dagger\right\| \cdot u \;\middle|\; E_t \right\} \leq \mathrm{e}^{-u^2/2}.$$

Under the event E_t, we have explicit bounds on the norms of $\boldsymbol{\Omega}_1^\dagger$, so

$$\mathbb{P}\left\{ \left\|\boldsymbol{\Sigma}_2\boldsymbol{\Omega}_2\boldsymbol{\Omega}_1^\dagger\right\| > \|\boldsymbol{\Sigma}_2\| \sqrt{\frac{3k}{p+1}} \cdot t + \|\boldsymbol{\Sigma}_2\|_{\mathrm{F}} \frac{\mathrm{e}\sqrt{k+p}}{p+1} \cdot t + \|\boldsymbol{\Sigma}_2\| \frac{\mathrm{e}\sqrt{k+p}}{p+1} \cdot ut \;\middle|\; E_t \right\} \leq \mathrm{e}^{-u^2/2}.$$

Use the fact $\mathbb{P}\left(E_t^c\right) \leq 2t^{-p}$ to remove the conditioning. Therefore,

$$\mathbb{P}\left\{ \left\|\boldsymbol{\Sigma}_2\boldsymbol{\Omega}_2\boldsymbol{\Omega}_1^\dagger\right\| > \|\boldsymbol{\Sigma}_2\| \sqrt{\frac{3k}{p+1}} \cdot t + \|\boldsymbol{\Sigma}_2\|_{\mathrm{F}} \frac{\mathrm{e}\sqrt{k+p}}{p+1} \cdot t + \|\boldsymbol{\Sigma}_2\| \frac{\mathrm{e}\sqrt{k+p}}{p+1} \cdot ut \right\} \leq 2t^{-p} + \mathrm{e}^{-u^2/2}.$$

Insert the expressions for the norms of $\boldsymbol{\Sigma}_2$ into this result to complete the probability bound. Finally, introduce this estimate into the error bound from Theorem 9.1. ∎

10.4. Analysis of the Power Scheme. Theorem 10.6 makes it clear that the performance of the randomized approximation scheme, Algorithm 4.1, depends heavily on the singular spectrum of the input matrix. The power scheme outlined in Algorithm 4.3 addresses this problem by enhancing the decay of spectrum. We can combine our analysis of Algorithm 4.1 with Theorem 9.2 to obtain a detailed report on the behavior of the performance of the power scheme using a Gaussian matrix.

COROLLARY 10.10 (average spectral error for the power scheme). *Frame the hypotheses of Theorem* 10.5. *Define* $\boldsymbol{B} = (\boldsymbol{A}\boldsymbol{A}^*)^q \boldsymbol{A}$ *for a nonnegative integer* q, *and construct the sample matrix* $\boldsymbol{Z} = \boldsymbol{B}\boldsymbol{\Omega}$. *Then*

$$\mathbb{E}\,\|(\mathbf{I} - \boldsymbol{P}_{\boldsymbol{Z}})\boldsymbol{A}\| \le \left[\left(1 + \sqrt{\frac{k}{p-1}}\right)\sigma_{k+1}^{2q+1} + \frac{\mathrm{e}\sqrt{k+p}}{p}\left(\sum\nolimits_{j>k}\sigma_j^{2(2q+1)}\right)^{1/2}\right]^{1/(2q+1)}.$$

Proof. By Hölder's inequality and Theorem 9.2,

$$\mathbb{E}\,\|(\mathbf{I} - \boldsymbol{P}_{\boldsymbol{Z}})\boldsymbol{A}\| \le \left(\mathbb{E}\,\|(\mathbf{I} - \boldsymbol{P}_{\boldsymbol{Z}})\boldsymbol{A}\|^{2q+1}\right)^{1/(2q+1)} \le \left(\mathbb{E}\,\|(\mathbf{I} - \boldsymbol{P}_{\boldsymbol{Z}})\boldsymbol{B}\|\right)^{1/(2q+1)}.$$

Invoke Theorem 10.6 to bound the right-hand side, noting that $\sigma_j(\boldsymbol{B}) = \sigma_j^{2q+1}$. ∎

The true message of Corollary 10.10 emerges if we bound the series using its largest term σ_{k+1}^{4q+2} and draw the factor σ_{k+1} out of the bracket:

$$\mathbb{E}\,\|(\mathbf{I} - \boldsymbol{P}_{\boldsymbol{Z}})\boldsymbol{A}\| \le \left[1 + \sqrt{\frac{k}{p-1}} + \frac{\mathrm{e}\sqrt{k+p}}{p}\cdot\sqrt{\min\{m,n\} - k}\right]^{1/(2q+1)}\sigma_{k+1}.$$

In other words, as we increase the exponent q, the power scheme drives the extra factor in the error to one exponentially fast. By the time $q \sim \log(\min\{m,n\})$,

$$\mathbb{E}\,\|(\mathbf{I} - \boldsymbol{P}_{\boldsymbol{Z}})\boldsymbol{A}\| \sim \sigma_{k+1},$$

which is the baseline for the spectral norm.

In most situations, the error bound given by Corollary 10.10 is substantially better than the estimates discussed in the last paragraph. For example, suppose that the tail singular values exhibit the decay profile

$$\sigma_j \lesssim j^{(1+\varepsilon)/(4q+2)} \quad \text{for } j > k \text{ and } \varepsilon > 0.$$

Then the series in Corollary 10.10 is comparable with its largest term, which allows us to remove the dimensional factor $\min\{m,n\}$ from the error bound.

To obtain large deviation bounds for the performance of the power scheme, simply combine Theorem 9.2 with Theorem 10.8. We omit a detailed statement.

Remark 10.1. We lack an analogous theory for the Frobenius norm because Theorem 9.2 depends on Proposition 8.6, which is not true for the Frobenius norm. It is possible to obtain some results by estimating the Frobenius norm in terms of the spectral norm.

11. SRFT Test Matrices. Another way to implement the proto-algorithm from section 1.3 is to use a structured random matrix so that the matrix product in step 2 can be performed quickly. One type of structured random matrix that has been proposed in the literature is the *subsampled random Fourier transform*, or SRFT, which we discussed in section 4.6. In this section, we present bounds on the performance of the proto-algorithm when it is implemented with an SRFT test matrix. In contrast to the results for Gaussian test matrices, the results in this section hold for both real and complex input matrices.

11.1. Construction and Properties. Recall from section 4.6 that an SRFT is a tall $n \times \ell$ matrix of the form $\boldsymbol{\Omega} = \sqrt{n/\ell} \cdot \boldsymbol{DFR}^*$, where

- $\boldsymbol{D}$ is a random $n \times n$ diagonal matrix whose entries are independent and uniformly distributed on the complex unit circle;
- $\boldsymbol{F}$ is the $n \times n$ unitary discrete Fourier transform; and
- $\boldsymbol{R}$ is a random $\ell \times n$ matrix that restricts an n-dimensional vector to ℓ coordinates, chosen uniformly at random.

Up to scaling, an SRFT is just a section of a unitary matrix, so it satisfies the norm identity $\|\boldsymbol{\Omega}\| = \sqrt{n/\ell}$. The critical fact is that an appropriately designed SRFT approximately preserves the geometry of an *entire subspace of vectors.*

THEOREM 11.1 (SRFT preserves geometry). *Fix an $n \times k$ orthonormal matrix $\boldsymbol{V}$, and draw an $n \times \ell$ SRFT matrix $\boldsymbol{\Omega}$, where the parameter ℓ satisfies*

$$4\left[\sqrt{k} + \sqrt{8\log(kn)}\right]^2 \log(k) \leq \ell \leq n.$$

Then

$$0.40 \leq \sigma_k(\boldsymbol{V}^*\boldsymbol{\Omega}) \quad \textit{and} \quad \sigma_1(\boldsymbol{V}^*\boldsymbol{\Omega}) \leq 1.48$$

with failure probability at most $\mathrm{O}(k^{-1})$.

In other words, the kernel of an SRFT of dimension $\ell \sim k\log(k)$ is unlikely to have a nontrivial intersection with a fixed k-dimensional subspace. In contrast with the Gaussian case, the logarithmic factor $\log(k)$ in the lower bound on ℓ cannot generally be removed (Remark 11.2).

Theorem 11.1 follows from a straightforward variation of the argument in [135], which establishes equivalent bounds for a real analogue of the SRFT, called the *subsampled randomized Hadamard transform* (SRHT). We omit further details.

Remark 11.1. For large problems, we can obtain better numerical constants [135, Thm. 3.2]. Fix a small, positive number ι. If $k \gg \log(n)$, then sampling

$$\ell \geq (1+\iota) \cdot k\log(k)$$

coordinates is sufficient to ensure that $\sigma_k(\boldsymbol{V}^*\boldsymbol{\Omega}) \geq \iota$ with failure probability at most $\mathrm{O}(k^{-c\iota})$. This sampling bound is essentially optimal because $(1-\iota)\cdot k\log(k)$ samples are not adequate in the worst case; see Remark 11.2.

Remark 11.2. The logarithmic factor in Theorem 11.1 is *necessary* when the orthonormal matrix $\boldsymbol{V}$ is particularly evil. Let us describe an infinite family of worst-case examples. Fix an integer k, and let $n = k^2$. Form an $n \times k$ orthonormal matrix $\boldsymbol{V}$ by regular decimation of the $n \times n$ identity matrix. More precisely, $\boldsymbol{V}$ is the matrix whose jth row has a unit entry in column $(j-1)/k$ when $j \equiv 1 \pmod{k}$ and is zero otherwise. To see why this type of matrix is nasty, it is helpful to consider the auxiliary matrix $\boldsymbol{W} = \boldsymbol{V}^*\boldsymbol{DF}$. Observe that, up to scaling and modulation of columns, $\boldsymbol{W}$ consists of k copies of a $k \times k$ DFT concatenated horizontally.

Suppose that we apply the SRFT $\boldsymbol{\Omega} = \boldsymbol{DFR}^*$ to the matrix $\boldsymbol{V}^*$. We obtain a matrix of the form $\boldsymbol{X} = \boldsymbol{V}^*\boldsymbol{\Omega} = \boldsymbol{WR}^*$, which consists of ℓ random columns sampled from $\boldsymbol{W}$. Theorem 11.1 certainly cannot hold unless $\sigma_k(\boldsymbol{X}) > 0$. To ensure the latter event occurs, we must pick at least one copy each of the k distinct columns of $\boldsymbol{W}$. This is the coupon collector's problem [99, sect. 3.6] in disguise. To obtain a complete set of k coupons (i.e., columns) with nonnegligible probability, we must draw at least $k\log(k)$ columns. The fact that we are sampling without replacement does not improve the analysis appreciably because the matrix has too many columns.

11.2. Performance Guarantees. We are now prepared to present detailed information on the performance of the proto-algorithm when the test matrix $\boldsymbol{\Omega}$ is an SRFT.

THEOREM 11.2 (error bounds for SRFT). *Fix an $m \times n$ matrix $\boldsymbol{A}$ with singular values $\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq \cdots$. Draw an $n \times \ell$ SRFT matrix $\boldsymbol{\Omega}$, where*

$$4\left[\sqrt{k} + \sqrt{8\log(kn)}\right]^2 \log(k) \leq \ell \leq n.$$

Construct the sample matrix $\boldsymbol{Y} = \boldsymbol{A}\boldsymbol{\Omega}$. Then

$$\|(\mathbf{I} - \boldsymbol{P}_{\boldsymbol{Y}})\boldsymbol{A}\| \leq \sqrt{1 + 7n/\ell} \cdot \sigma_{k+1} \quad \textit{and}$$
$$\|(\mathbf{I} - \boldsymbol{P}_{\boldsymbol{Y}})\boldsymbol{A}\|_{\mathrm{F}} \leq \sqrt{1 + 7n/\ell} \cdot \left(\sum\nolimits_{j>k} \sigma_j^2\right)^{1/2}$$

with failure probability at most $\mathrm{O}(k^{-1})$.

As we saw in section 10.2, the quantity σ_{k+1} is the minimal spectral-norm error possible when approximating $\boldsymbol{A}$ with a rank-k matrix. Similarly, the series in the second bound is the minimal Frobenius-norm error when approximating $\boldsymbol{A}$ with a rank-k matrix. We see that both error bounds lie within a polynomial factor of the baseline, and this factor decreases with the number ℓ of samples we retain.

The likelihood of error with an SRFT test matrix is substantially worse than in the Gaussian case. The failure probability here is roughly k^{-1}, while in the Gaussian case, the failure probability is roughly $\mathrm{e}^{-(\ell-k)}$. This qualitative difference is not an artifact of the analysis; discrete sampling techniques inherently fail with higher probability.

Matrix approximation schemes based on SRFTs often perform much better in practice than the error analysis here would indicate. While it is not generally possible to guarantee accuracy with a sampling parameter less than $\ell \sim k\log(k)$, we have found empirically that the choice $\ell = k + 20$ is adequate in almost all applications. Indeed, SRFTs sometimes perform even *better* than Gaussian matrices (see, e.g., Figure 7.8).

We complete the section with the proof of Theorem 11.2.

Proof of Theorem 11.2. Let $\boldsymbol{V}$ be the right unitary factor of matrix $\boldsymbol{A}$, and partition $\boldsymbol{V} = [\boldsymbol{V}_1 \mid \boldsymbol{V}_2]$ into blocks containing, respectively, k and $n - k$ columns. Recall that

$$\boldsymbol{\Omega}_1 = \boldsymbol{V}_1^*\boldsymbol{\Omega} \quad \text{and} \quad \boldsymbol{\Omega}_2 = \boldsymbol{V}_2^*\boldsymbol{\Omega},$$

where $\boldsymbol{\Omega}$ is the conjugate transpose of an SRFT. Theorem 11.1 ensures that the submatrix $\boldsymbol{\Omega}_1$ has full row rank, with failure probability at most $\mathrm{O}(k^{-1})$. Therefore, Theorem 9.1 implies that

$$|||(\mathbf{I} - \boldsymbol{P}_{\boldsymbol{Y}})\boldsymbol{A}||| \leq |||\boldsymbol{\Sigma}_2||| \left[1 + \left\|\boldsymbol{\Omega}_1^\dagger\right\|^2 \cdot \|\boldsymbol{\Omega}_2\|^2\right]^{1/2},$$

where $|||\cdot|||$ denotes either the spectral norm or the Frobenius norm. Our application of Theorem 11.1 also ensures that the spectral norm of $\boldsymbol{\Omega}_1^\dagger$ is under control.

$$\left\|\boldsymbol{\Omega}_1^\dagger\right\|^2 \leq \frac{1}{0.40^2} < 7.$$

We may bound the spectral norm of $\boldsymbol{\Omega}_2$ deterministically,

$$\|\boldsymbol{\Omega}_2\| = \|\boldsymbol{V}_2^*\boldsymbol{\Omega}\| \leq \|\boldsymbol{V}_2^*\| \, \|\boldsymbol{\Omega}\| = \sqrt{n/\ell},$$

since $\boldsymbol{V}_2$ and $\sqrt{\ell/n} \cdot \boldsymbol{\Omega}$ are both orthonormal matrices. Combine these estimates to complete the proof. ∎

Appendix A. On Gaussian Matrices. This appendix collects some of the properties of Gaussian matrices that we use in our analysis. Most of these results follow quickly from material that is readily available in the literature. We focus on the real case here; the complex case is similar but actually yields better results.

A.1. Expectation of Norms. We begin with the expected Frobenius norm of a scaled Gaussian matrix, which follows from an easy calculation.

PROPOSITION A.1. *Fix real matrices $\boldsymbol{S}, \boldsymbol{T}$, and draw a standard Gaussian matrix $\boldsymbol{G}$. Then*

$$\left(\mathbb{E}\,\|\boldsymbol{SGT}\|_{\mathrm{F}}^2\right)^{1/2} = \|\boldsymbol{S}\|_{\mathrm{F}}\,\|\boldsymbol{T}\|_{\mathrm{F}}\,.$$

Proof. The distribution of a Gaussian matrix is invariant under orthogonal transformations, and the Frobenius norm is also unitarily invariant. As a result, it represents no loss of generality to assume that $\boldsymbol{S}$ and $\boldsymbol{T}$ are diagonal. Therefore,

$$\mathbb{E}\,\|\boldsymbol{SGT}\|_{\mathrm{F}}^2 = \mathbb{E}\left[\sum\nolimits_{jk} |s_{jj} g_{jk} t_{kk}|^2\right] = \sum\nolimits_{jk} |s_{jj}|^2 |t_{kk}|^2 = \|\boldsymbol{S}\|_{\mathrm{F}}^2\,\|\boldsymbol{T}\|_{\mathrm{F}}^2\,.$$

Since the right-hand side is unitarily invariant, we have also identified the value of the expectation for general matrices $\boldsymbol{S}$ and $\boldsymbol{T}$. ∎

The literature contains an excellent bound for the expected spectral norm of a scaled Gaussian matrix. The result is due to Gordon [62, 63], who established the bound using a sharp version of Slepian's lemma. See [84, sect. 3.3] and [34, sect. 2.3] for additional discussion.

PROPOSITION A.2. *Fix real matrices $\boldsymbol{S}, \boldsymbol{T}$, and draw a standard Gaussian matrix $\boldsymbol{G}$. Then*

$$\mathbb{E}\,\|\boldsymbol{SGT}\| \le \|\boldsymbol{S}\|\,\|\boldsymbol{T}\|_{\mathrm{F}} + \|\boldsymbol{S}\|_{\mathrm{F}}\,\|\boldsymbol{T}\|\,.$$

A.2. Spectral Norm of Pseudoinverse. Now, we turn to the pseudoinverse of a Gaussian matrix. Recently, Chen and Dongarra developed a good bound on the probability that its spectral norm is large. The statement here follows from [25, Lem. 4.1] after an application of Stirling's approximation. See also [92, Lem. 2.14].

PROPOSITION A.3. *Let $\boldsymbol{G}$ be an $m \times n$ standard Gaussian matrix with $n \ge m \ge 2$. For each $t > 0$,*

$$\mathbb{P}\left\{\left\|\boldsymbol{G}^{\dagger}\right\| > t\right\} \le \frac{1}{\sqrt{2\pi(n-m+1)}} \left[\frac{\mathrm{e}\sqrt{n}}{n-m+1}\right]^{n-m+1} t^{-(n-m+1)}.$$

We can use Proposition A.3 to bound the expected spectral norm of a pseudoinverted Gaussian matrix.

PROPOSITION A.4. *Let $\boldsymbol{G}$ be an $m \times n$ standard Gaussian matrix with $n - m \ge 1$ and $m \ge 2$. Then*

$$\mathbb{E}\left\|\boldsymbol{G}^{\dagger}\right\| < \frac{\mathrm{e}\sqrt{n}}{n-m}.$$

Proof. Let us make the abbreviations $p = n - m$ and

$$C = \frac{1}{\sqrt{2\pi(p+1)}} \left[\frac{\mathrm{e}\sqrt{n}}{p+1}\right]^{p+1}.$$

We compute the expectation by way of a standard argument. The integral formula for the mean of a nonnegative random variable implies that, for all $E > 0$,

$$\begin{aligned}\mathbb{E}\left\|\boldsymbol{G}^{\dagger}\right\| &= \int_0^{\infty} \mathbb{P}\left\{\left\|\boldsymbol{G}^{\dagger}\right\| > t\right\} \,\mathrm{d}t \leq E + \int_E^{\infty} \mathbb{P}\left\{\left\|\boldsymbol{G}^{\dagger}\right\| > t\right\} \,\mathrm{d}t \\ &\leq E + C\int_E^{\infty} t^{-(p+1)} \,\mathrm{d}t = E + \frac{1}{p} C E^{-p},\end{aligned}$$

where the second inequality follows from Proposition A.3. The right-hand side is minimized when $E = C^{1/(p+1)}$. Substitute and simplify. ∎

A.3. Frobenius Norm of Pseudoinverse. The squared Frobenius norm of a pseudoinverted Gaussian matrix is closely connected with the trace of an inverted Wishart matrix. This observation leads to an exact expression for the mean, as well as simple bounds for the tail probabilities.

The key tool in this section is a classical result which asserts that the diagonal of an inverted Wishart matrix consists of inverted χ^2 variates.

PROPOSITION A.5. *Let $\boldsymbol{G}$ be an $m \times n$ standard Gaussian matrix with $n \geq m$. For each $j = 1, 2, \ldots, m$,*

$$\left[(\boldsymbol{G}\boldsymbol{G}^*)^{-1}\right]_{jj} \sim X_j^{-1}, \quad \text{where } X_j \sim \chi^2_{n-m+1}.$$

Let us caution that these diagonal entries are correlated, so it takes some care to study their joint statistics. See [100, p. 96f] for the proof of Proposition A.5, as well as some additional discussion.

An immediate consequence of the latter result is a formula for the expected Frobenius norm of a pseudoinverted Gaussian matrix.

PROPOSITION A.6. *Let $\boldsymbol{G}$ be an $m \times n$ standard Gaussian matrix with $n - m \geq 2$. Then*

$$\mathbb{E}\left\|\boldsymbol{G}^{\dagger}\right\|_{\mathrm{F}}^2 = \frac{m}{n-m-1}.$$

Proof. Observe that

$$\left\|\boldsymbol{G}^{\dagger}\right\|_{\mathrm{F}}^2 = \operatorname{trace}\left[(\boldsymbol{G}^{\dagger})^*\boldsymbol{G}^{\dagger}\right] = \operatorname{trace}\left[(\boldsymbol{G}\boldsymbol{G}^*)^{-1}\right].$$

The second identity holds almost surely because the Wishart matrix $\boldsymbol{G}\boldsymbol{G}^*$ is invertible with probability one. Proposition A.5 states that the diagonal entries of $(\boldsymbol{G}\boldsymbol{G}^*)^{-1}$ are inverted χ^2 random variables. Therefore,

$$\left\|\boldsymbol{G}^{\dagger}\right\|_{\mathrm{F}}^2 = \sum\nolimits_{j=1}^{m} X_j^{-1}, \quad \text{where } X_j \sim \chi^2_{n-m+1}. \tag{A.1}$$

But we can compute the expectation of an inverted χ^2 variate explicitly:

$$\mathbb{E}(X_j^{-1}) = \frac{1}{n-m-1}.$$

This identity follows from Proposition A.8 below, a more general result on the moments of a χ^2 variate. ∎

The following theorem provides an adequate bound on the probability of a large deviation above the mean.

THEOREM A.7. *Let $\boldsymbol{G}$ be an $m \times n$ standard Gaussian matrix with $n - m \geq 4$. For each $t \geq 1$,*

$$\mathbb{P}\left\{ \left\| \boldsymbol{G}^{\dagger} \right\|_{\mathrm{F}}^2 > \frac{3m}{n-m+1} \cdot t \right\} \leq t^{-(n-m)/2}.$$

Neither the precise form of Theorem A.7 nor the constants are ideal; we have focused instead on establishing a useful bound with minimal fuss. The rest of the section is devoted to the proof. Unfortunately, most of the standard methods for producing tail bounds fail for random variables that do not exhibit normal or exponential concentration. Our argument relies on Proposition A.5 and a dose of hard analysis.

A.3.1. Technical Background. We begin with a piece of notation. For any number $q \geq 1$, we define the L_q norm of a random variable Z by

$$\mathbb{E}^q(Z) = \left(\mathbb{E}\,|Z|^q\right)^{1/q}.$$

In particular, the L_q norm satisfies the triangle inequality.

We need to know the moments of an inverted chi-square variate, which are expressed in terms of special functions.

PROPOSITION A.8. *Let X be a χ^2 variate with k degrees of freedom. When $0 \leq q < k/2$,*

$$\mathbb{E}\left(X^{-q}\right) = \frac{\Gamma(k/2-q)}{2^q \Gamma(k/2)}.$$

In particular, the mean of an inverted χ^2 variate satisfies $\mathbb{E}(X^{-1}) = (k-2)^{-1}$.

Proof. Recall that a χ^2 variate with k degrees of freedom has the probability density function

$$f(t) = \frac{1}{2^{k/2}\Gamma(k/2)} t^{k/2-1} \mathrm{e}^{-t/2} \quad \text{for } t \geq 0.$$

By the integral formula for expectation,

$$\mathbb{E}(X^{-q}) = \int_0^{\infty} t^{-q} f(t)\,\mathrm{d}t = \frac{\Gamma(k/2-q)}{2^q\Gamma(k/2)},$$

where the second equality follows from Euler's integral expression for the gamma function after a change of variables. ∎

To streamline the proof, we eliminate the gamma functions from Proposition A.8. The next lemma delivers an adequate estimate for the negative moments of a chi-square variate.

LEMMA A.9. *Let X be a χ^2 variate with k degrees of freedom, where $k \geq 5$. When $2 \leq q \leq (k-1)/2$,*

$$\mathbb{E}^q\left(X^{-1}\right) < \frac{3}{k}.$$

Proof. We establish the bound for $q = (k-1)/2$. For smaller values of q, the result follows from Hölder's inequality. Proposition A.8 shows that

$$\mathbb{E}^q\left(X^{-1}\right) = \left[\frac{\Gamma(k/2-q)}{2^q\Gamma(k/2)}\right]^{1/q} = \left[\frac{\Gamma(1/2)}{2^q\Gamma(k/2)}\right]^{1/q}.$$

Stirling's approximation ensures that $\Gamma(k/2) \geq \sqrt{2\pi} \cdot (k/2)^{(k-1)/2} \cdot \mathrm{e}^{-k/2}$. We invoke the identity $\Gamma(1/2) = \sqrt{\pi}$ to see that

$$\mathbb{E}^q \left(X^{-1}\right) \leq \left[\frac{\sqrt{\pi}}{2^q \sqrt{2\pi} \cdot (k/2)^q \cdot \mathrm{e}^{-q-1/2}}\right]^{1/q} = \frac{\mathrm{e}}{k} \left[\frac{\mathrm{e}}{2}\right]^{1/2q} < \frac{3}{k},$$

where we used the assumption $q \geq 2$ to complete the numerical estimate. ∎

A.3.2. Proof of Theorem A.7. Let $\boldsymbol{G}$ be an $m \times n$ Gaussian matrix, where we assume that $n - m \geq 4$. Define the random variable

$$Z = \left\|\boldsymbol{G}^\dagger\right\|_{\mathrm{F}}^2 = \mathrm{trace}[(\boldsymbol{G}\boldsymbol{G}^*)^{-1}],$$

where the second equality holds almost surely. The argument depends on the representation $Z = \sum_{j=1}^m X_j^{-1}$, where each $X_j \sim \chi^2_{n-m+1}$. Using this expression, we can bound the qth moment of Z in terms of the moments of the summands.

$$\mathbb{E}^q(Z) = \mathbb{E}^q \left[\sum\nolimits_{j=1}^m X_j^{-1}\right] \leq \sum\nolimits_{j=1}^m \mathbb{E}^q(X_j^{-1}),$$

where the second relation is the triangle inequality for the L_q norm. For the exponent $q = (n-m)/2$, Lemma A.9 ensures that

$$\mathbb{E}^q(X_j^{-1}) < \frac{3}{n-m+1} \quad \text{for } j = 1, \ldots, m.$$

In summary,

$$[\mathbb{E}(Z^q)]^{1/q} = \mathbb{E}^q(Z) < \frac{3m}{n-m+1}.$$

An application of Markov's inequality delivers the bound

$$\mathbb{P}\left\{Z \geq \frac{3m}{n-m+1} \cdot t\right\} \leq \left[\frac{3m}{n-m+1} \cdot t\right]^{-q} \cdot \mathbb{E}(Z^q) = t^{-q}.$$

This estimate completes the proof of Theorem A.7.

Remark A.1. There was an error in the proof of Theorem A.7 that we presented in earlier versions of this paper. The current approach leads to a better estimate even though the argument is simpler. We believe it would be an interesting project to develop more conceptual and extensible methods that deliver accurate concentration results for inverse spectral functions of a Gaussian matrix.

Acknowledgments. The authors have benefited from valuable discussions with many researchers, among them Inderjit Dhillon, Petros Drineas, Ming Gu, Edo Liberty, Michael Mahoney, Vladimir Rokhlin, Yoel Shkolnisky, and Arthur Szlam. In particular, we would like to thank Mark Tygert for his insightful remarks on early drafts of this paper. The example in section 7.2 was provided by François Meyer of the University of Colorado at Boulder. The example in section 7.3 comes from the FERET database of facial images collected under the FERET program, sponsored by the DoD Counterdrug Technology Development Program Office. The work reported was initiated during the program *Mathematics of Knowledge and Search Engines* held at IPAM in the fall of 2007. Finally, we would like to thank the anonymous referees, whose thoughtful remarks have helped us to improve the manuscript dramatically.

REFERENCES

[1] D. Achlioptas, *Database-friendly random projections: Johnson–Lindenstrauss with binary coins*, J. Comput. System Sci., 66 (2003), pp. 671–687.

[2] D. Achlioptas and F. McSherry, *Fast computation of low-rank matrix approximations*, J. Assoc. Comput. Mach., 54 (2007), article 9.

[3] N. Ailon and B. Chazelle, *Approximate nearest neighbors and the fast Johnson–Lindenstrauss transform*, in Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC '06), 2006, pp. 557–563.

[4] N. Ailon and E. Liberty, *Fast dimension reduction using Rademacher series on dual BCH codes*, in Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC '08), 2008, pp. 1–9.

[5] N. Alon, P. Gibbons, Y. Matias, and M. Szegedy, *Tracking join and self-join sizes in limited storage*, in Proceedings of the 18th ACM Symposium on Principles of Database Systems (PODS), 1999, pp. 10–20.

[6] N. Alon, Y. Matias, and M. Szegedy, *The space complexity of approximating frequency moments*, in Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC '96), 1996, pp. 20–29.

[7] S. Arora, E. Hazan, and S. Kale, *A fast random sampling algorithm for sparsifying matrices*, in Approximation, Randomization and Combinatorial Optimization: Algorithms and Techniques, Springer, Berlin, 2006, pp. 272–279.

[8] A. R. Barron, *Universal approximation bounds for superpositions of a sigmoidal function*, IEEE Trans. Inform. Theory, 39 (1993), pp. 930–945.

[9] I. Beichl, *The Metropolis algorithm*, Comput. Sci. Eng., 2 (2000), pp. 65–69.

[10] M.-A. Bellabas and P. J. Wolfe, *On sparse representations of linear operators and the approximation of matrix products*, in Proceedings of the 42nd Annual Conference on Information Sciences and Systems (CISS), 2008, pp. 258–263.

[11] R. Bhatia, *Matrix Analysis*, Grad. Texts in Math. 169, Springer, Berlin, 1997.

[12] Å. Björck, *Numerics of Gram–Schmidt orthogonalization*, Linear Algebra Appl., 197–198 (1994), pp. 297–316.

[13] Å. Björck, *Numerical Methods for Least Squares Problems*, SIAM, Philadelphia, PA, 1996.

[14] V. Bogdanov, *Gaussian Measures*, AMS, Providence, RI, 1998.

[15] J. Bourgain, *On Lipschitz embedding of finite metric spaces in Hilbert space*, Israel J. Math., 52 (1985), pp. 46–52.

[16] C. Boutsidis and P. Drineas, *Random projections for nonnegative least squares*, Linear Algebra Appl., 431 (2009), pp. 760–771.

[17] C. Boutsidis, M. W. Mahoney, and P. Drineas, *An improved approximation algorithm for the column subset selection problem*, in Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2009, pp. 968–977.

[18] C. Boutsidis, M. W. Mahoney, and P. Drineas, *Unsupervised feature selection for principal components analysis*, in Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 2008, pp. 61–69.

[19] E. Candès and J. K. Romberg, *Sparsity and incoherence in compressive sampling*, Inverse Problems, 23 (2007), pp. 969–985.

[20] E. Candès, J. K. Romberg, and T. Tao, *Robust uncertainty principles: Exact signal reconstruction from highly incomplete Fourier information*, IEEE Trans. Inform. Theory, 52 (2006), pp. 489–509.

[21] E. J. Candès, *Compressive sampling*, in Proceedings of the 2006 International Congress of Mathematicians, Madrid, 2006.

[22] E. J. Candès and B. Recht, *Exact matrix completion via convex optimization*, Found. Comput. Math., 9 (2009), pp. 717–772.

[23] E. J. Candès and T. Tao, *The power of convex relaxation: Near-optimal matrix completion*, IEEE Trans. Inform. Theory, 56 (2010), pp. 2053–2080.

[24] B. Carl, *Inequalities of Bernstein–Jackson-type and the degree of compactness in Banach spaces*, Ann. Inst. Fourier (Grenoble), 35 (1985), pp. 79–118.

[25] Z. Chen and J. J. Dongarra, *Condition numbers of Gaussian random matrices*, SIAM J. Matrix Anal. Appl., 27 (2005), pp. 603–620.

[26] H. Cheng, Z. Gimbutas, P. G. Martinsson, and V. Rokhlin, *On the compression of low rank matrices*, SIAM J. Sci. Comput., 26 (2005), pp. 1389–1404.

[27] A. Çivril and M. Magdon-Ismail, *On selecting a maximum volume sub-matrix of a matrix and related problems*, Theoret. Comput. Sci., 410 (2009), pp. 4801–4811.

[28] K. L. Clarkson, *Subgradient and sampling algorithms for ℓ_1 regression*, in Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2005, pp. 257–266.

[29] K. L. Clarkson and D. P. Woodruff, *Numerical linear algebra in the streaming model*, in Proceedings of the 41st Annual ACM Symposium on Theory of Computing (STOC '09), 2009, pp. 205–214.
[30] R. R. Coifman, S. Lafon, A. B. Lee, M. Maggioni, B. Nadler, F. Warner, and S. W. Zucker, *Geometric diffusions as a tool for harmonic analysis and structure definition of data: Diffusion maps*, Proc. Natl. Acad. Sci. USA, 102 (2005), pp. 7426–7431.
[31] A. Dasgupta, P. Drineas, B. Harb, R. Kumar, and M. W. Mahoney, *Sampling algorithms and coresets for ℓ_p regression*, SIAM J. Comput., 38 (2009), pp. 2060–2078.
[32] S. Dasgupta and A. Gupta, *An Elementary Proof of the Johnson–Lindenstrauss Lemma*, Tech. Report 99-006, University of California at Berkeley, 1999.
[33] A. d'Aspremont, *Subsampling Algorithms for Semidefinite Programming*, preprint, 2009; available online from http://arxiv.org/abs/0803.1990.
[34] K. R. Davidson and S. J. Szarek, *Local operator theory, random matrices, and Banach spaces*, in Handbook of Banach Space Geometry, W. B. Johnson and J. Lindenstrauss, eds., Elsevier, 2002, pp. 317–366.
[35] J. Demmel, I. Dumitriu, and O. Holtz, *Fast linear algebra is stable*, Numer. Math., 108 (2007), pp. 59–91.
[36] A. Deshpande and L. Rademacher, *Efficient Volume Sampling for Row/Column Subset Selection*, preprint, 2010; available online from http://arxiv.org/abs/1004.4057.
[37] A. Deshpande, L. Rademacher, S. Vempala, and G. Wang, *Matrix approximation and projective clustering via volume sampling*, in Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2006, pp. 1117–1126.
[38] A. Deshpande and S. Vempala, *Adaptive sampling and fast low-rank matrix approximation*, in Approximation, Randomization and Combinatorial Optimization, J. Díaz, K. Jansen, J. D. P. Rolim, and U. Zwick, eds., Lecture Notes in Comput. Sci. 4110, Springer, Berlin, 2006, pp. 292–303.
[39] J. D. Dixon, *Estimating extremal eigenvalues and condition numbers of matrices*, SIAM J. Numer. Anal., 20 (1983), pp. 812–814.
[40] J. Dongarra and F. Sullivan, *The top* 10 *algorithms*, Comput. Sci. Eng., 2 (1) (2000), pp. 22–23.
[41] D. L. Donoho, *Compressed sensing*, IEEE Trans. Inform. Theory, 52 (2006), pp. 1289–1306.
[42] D. L. Donoho, M. Vetterli, R. A. DeVore, and I. Daubechies, *Data compression and harmonic analysis*, IEEE Trans. Inform. Theory, 44 (1998), pp. 2433–2452.
[43] P. Drineas, A. Frieza, R. Kannan, S. Vempala, and V. Vinay, *Clustering of large graphs via the singular value decomposition*, Machine Learning, 56 (2004), pp. 9–33.
[44] P. Drineas, A. Frieze, R. Kannan, S. Vempala, and V. Vinay, *Clustering in large graphs and matrices*, in Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 1999, pp. 291–299.
[45] P. Drineas, R. Kannan, and M. W. Mahoney, *Fast Monte Carlo algorithms for matrices.* I. *Approximating matrix multiplication*, SIAM J. Comput., 36 (2006), pp. 132–157.
[46] P. Drineas, R. Kannan, and M. W. Mahoney, *Fast Monte Carlo algorithms for matrices.* II. *Computing a low-rank approximation to a matrix*, SIAM J. Comput., 36 (2006), pp. 158–183.
[47] P. Drineas, R. Kannan, and M. W. Mahoney, *Fast Monte Carlo algorithms for matrices.* III. *Computing a compressed approximate matrix decomposition*, SIAM J. Comput., 36 (2006), pp. 184–206.
[48] P. Drineas and M. W. Mahoney, *On the Nyström method for approximating a Gram matrix for improved kernel-based learning*, J. Mach. Learn. Res., 6 (2005), pp. 2153–2175.
[49] P. Drineas and M. W. Mahoney, *A randomized algorithm for a tensor-based generalization of the singular value decomposition*, Linear Algebra Appl., 420 (2007), pp. 553–571.
[50] P. Drineas, M. W. Mahoney, and S. Muthukrishnan, *Subspace sampling and relative-error matrix approximation: Column-based methods*, in Approximation, Randomization and Combinatorial Optimization, J. Díaz, K. Jansen, J. D. P. Rolim, and U. Zwick, eds., Lecture Notes in Comput. Sci. 4110, Springer, Berlin, 2006, pp. 321–326.
[51] P. Drineas, M. W. Mahoney, and S. Muthukrishnan, *Relative-error CUR matrix decompositions*, SIAM J. Matrix Anal. Appl., 30 (2008), pp. 844–881.
[52] P. Drineas, M. W. Mahoney, S. Muthukrishnan, and T. Sarlós, *Faster least squares approximation*, Numer. Math., 117 (2011), pp 219–249.
[53] A. Dvoretzky, *Some results on convex bodies and Banach spaces*, in Proceedings of the International Symposium on Linear Spaces, Jerusalem Academic Press, Jerusalem, Pergamon, Oxford, 1961, pp. 123–160.
[54] C. Eckart and G. Young, *The approximation of one matrix by another of lower rank*, Psychometrika, 1 (1936), pp. 211–218.

[55] A. Edelman, *Eigenvalues and Condition Numbers of Random Matrices*, Ph.D. thesis, Massachusetts Institute of Technology, Boston, MA, 1989.
[56] B. Engquist and O. Runborg, *Wavelet-based numerical homogenization with applications*, in Multiscale and Multiresolution Methods: Theory and Applications, T. J. Barth, T. Chan, and R. Haimes, eds., Lect. Notes Comput. Sci. Eng. 20, Springer, Berlin, 2001, pp. 97–148.
[57] A. Frieze, R. Kannan, and S. Vempala, *Fast Monte Carlo algorithms for finding low-rank approximations*, in Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 1998, pp. 370–378.
[58] A. Frieze, R. Kannan, and S. Vempala, *Fast Monte Carlo algorithms for finding low-rank approximations*, J. Assoc. Comput. Mach., 51 (2004), pp. 1025–1041.
[59] A. Y. Garnaev and E. D. Gluskin, *The widths of a Euclidean ball*, Dokl. Akad. Nauk. SSSR, 277 (1984), pp. 1048–1052 (in Russian).
[60] A. Gittens and J. A. Tropp, *Error Bounds for Random Matrix Approximation Schemes*, preprint, 2009; available online from http://arxiv.org/abs/0911.4108.
[61] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed., Johns Hopkins University Press, Baltimore, MD, 1996.
[62] Y. Gordon, *Some inequalities for Gaussian processes and applications*, Israel J. Math., 50 (1985), pp. 265–289.
[63] Y. Gordon, *Gaussian processes and almost spherical sections of convex bodies*, Ann. Probab., 16 (1988), pp. 180–188.
[64] S. A. Goreinov, E. E. Tyrtyshnikov, and N. L. Zamarashkin, *Theory of pseudo-skeleton matrix approximations*, Linear Algebra Appl., 261 (1997), pp. 1–21.
[65] L. Grasedyck and W. Hackbusch, *Construction and arithmetics of $\mathcal{H}$-matrices*, Computing, 70 (2003), pp. 295–334.
[66] L. Greengard and V. Rokhlin, *A new version of the fast multipole method for the Laplace equation in three dimensions*, Acta Numer., 17 (1997), pp. 229–269.
[67] M. Gu, *Personal communication*, 2007.
[68] M. Gu and S. C. Eisenstat, *Efficient algorithms for computing a strong rank-revealing QR factorization*, SIAM J. Sci. Comput., 17 (1996), pp. 848–869.
[69] N. Halko, P.-G. Martinsson, Y. Shkolnisky, and M. Tygert, *An Algorithm for the Principal Component Analysis of Large Data Sets*, preprint, 2010; available online from http://arxiv.org/abs/1007.5510.
[70] S. Har-Peled, *Matrix Approximation in Linear Time*, manuscript, 2006; available online from http://valis.cs.uiuc.edu/~sariel/research/papers/05/lrank/.
[71] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed., Springer, Berlin, 2008.
[72] R. A. Horn and C. R. Johnson, *Matrix Analysis*, Cambridge University Press, Cambridge, UK, 1985.
[73] P. Indyk and R. Motwani, *Approximate nearest neighbors: Toward removing the curse of dimensionality*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC '98), 1998, pp. 604–613.
[74] W. B. Johnson and J. Lindenstrauss, *Extensions of Lipschitz mappings into a Hilbert space*, Contemp. Math., 26 (1984), pp. 189–206.
[75] P. W. Jones, A. Osipov, and V. Rokhlin, *A Randomized Approximate Nearest Neighbors Algorithm*, Tech. Report YALEU/DCS/RR-1434, Yale University, New Haven, CT, 2010.
[76] D. R. Karger, *Random sampling in cut, flow, and network design problems*, Math. Oper. Res., 24 (1999), pp. 383–413.
[77] D. R. Karger, *Minimum cuts in near-linear time*, J. ACM, 47 (2000), pp. 46–76.
[78] B. S. Kašin, *On the widths of certain finite-dimensional sets and classes of smooth functions*, Izv. Akad. Nauk. SSSR Ser. Mat., 41 (1977), pp. 334–351, 478 (in Russian).
[79] J. Kleinberg, *Two algorithms for nearest neighbor search in high dimensions*, in Proceedings of the 29th ACM Symposium on Theory of Computing (STOC '97), 1997, pp. 599–608.
[80] J. Kuczyński and H. Woźniakowski, *Estimating the largest eigenvalue by the power and Lanczos algorithms with a random start*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 1094–1122.
[81] E. Kushilevitz, R. Ostrovski, and Y. Rabani, *Efficient search for approximate nearest neighbor in high dimensional spaces*, SIAM J. Comput., 30 (2000), pp. 457–474.
[82] D. Le and D. S. Parker, *Using randomization to make recursive matrix algorithms practical*, J. Funct. Programming, 9 (1999), pp. 605–624.
[83] M. Ledoux, *The Concentration of Measure Phenomenon*, Math. Surveys Monogr. 89, AMS, Providence, RI, 2001.
[84] M. Ledoux and M. Talagrand, *Probability in Banach Spaces: Isoperimetry and Processes*, Springer, Berlin, 1991.

[85] W. S. Lee, P. L. Bartlett, and R. C. Williamson, *Efficient agnostic learning of neural networks with bounded fan-in*, IEEE Trans. Inform. Theory, 42 (1996), pp. 2118–2132.
[86] Z. Leyk and H. Woźniakowski, *Estimating the largest eigenvector by Lanczos and polynomial algorithms with a random start*, Numer. Linear Algebra Appl., 5 (1998), pp. 147–164.
[87] E. Liberty, *Accelerated Dense Random Projections*, Ph.D. thesis, Yale University, New Haven, CT, 2009.
[88] E. Liberty, N. Ailon, and A. Singer, *Dense fast random projections and lean Walsh transforms*, in Approximation, Randomization and Combinatorial Optimization, A. Goel, K. Jansen, J. Rolim, and R. Rubinfeld, eds., Lecture Notes in Comput. Sci. 5171, Springer, Berlin, 2008, pp. 512–522.
[89] E. Liberty, F. 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.
[90] M. W. Mahoney and P. Drineas, *CUR matrix decompositions for improved data analysis*, Proc. Natl. Acad. Sci. USA, 106 (2009), pp. 697–702.
[91] P.-G. Martinsson, V. Rokhlin, Y. Shkolnisky, and M. Tygert, *ID: A Software Package for Low-Rank Approximation of Matrices via Interpolative Decompositions*, Ver. 0.2, http://cims.nyu.edu/~tygert/software.html, 2008.
[92] 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.
[93] P.-G. Martinsson, A. Szlam, and M. Tygert, *Normalized Power Iterations for the Computation of SVD*, manuscript, 2010.
[94] J. Matoušek, *Lectures on Discrete Geometry*, Springer, Berlin, 2002.
[95] F. McSherry, *Spectral Methods in Data Analysis*, Ph.D. thesis, University of Washington, Seattle, WA, 2004.
[96] N. Metropolis and S. Ulam, *The Monte Carlo method*, J. Amer. Statist. Assoc., 44 (1949), pp. 335–341.
[97] V. D. Milman, *A new proof of A. Dvoretsky's theorem on cross-sections of convex bodies*, Funkcional. Anal. i Priložen., 5 (1971), pp. 28–37.
[98] L. Mirsky, *Symmetric gauge functions and unitarily invariant norms*, Quart. J. Math. Oxford Ser. (2), 11 (1960), pp. 50–59.
[99] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, Cambridge, UK, 1995.
[100] R. J. Muirhead, *Aspects of Multivariate Statistical Theory*, Wiley, New York, 1982.
[101] S. Muthukrishnan, *Data Streams: Algorithms and Applications*, now Publishers, Boston, MA, 2005.
[102] D. Needell, *Randomized Kaczmarz solver for noisy linear systems*, BIT, 50 (2010), pp. 395–403.
[103] N. H. Nguyen, T. T. Do, and T. D. Tran, *A fast and efficient algorithm for low-rank approximation of a matrix*, in Proceedings of the 41st Annual ACM Symposium on Theory of Computing (STOC '09), 2009, pp. 215–224.
[104] C.-T. Pan, *On the existence and computation of rank-revealing LU factorizations*, Linear Algebra Appl., 316 (2000), pp. 199–222.
[105] C. H. Papadimitriou, P. Raghavan, H. Tamaki, and S. Vempala, *Latent semantic indexing: A probabilistic analysis*, in Proceedings of the 17th ACM Symposium on Principles of Database Systems (PODS), 1998, pp. 159–168.
[106] C. H. Papadimitriou, P. Raghavan, H. Tamaki, and S. Vempala, *Latent semantic indexing: A probabilistic analysis*, J. Comput. System Sci., 61 (2000), pp. 217–235.
[107] D. S. Parker and B. Pierce, *The Randomizing FFT: An Alternative to Pivoting in Gaussian Elimination*, Tech. Report CSD 950037, University of California at Los Angeles, 1995.
[108] P. J. Phillips, H. Moon, S. Rizvi, and P. Rauss, *The FERET evaluation methodology for face recognition algorithms*, IEEE Trans. Pattern Anal. Mach. Intelligence, 22 (2000), pp. 1090–1104.
[109] P. J. Phillips, H. Wechsler, J. Huang, and P. Rauss, *The FERET database and evaluation procedure for face recognition algorithms*, Image Vision Comput., 16 (1998), pp. 295–306.
[110] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes: The Art of Scientific Computing*, 3rd ed., Cambridge University Press, Cambridge, UK, 2007.
[111] A. Rahimi and B. Recht, *Random features for large-scale kernel machines*, in Proceedings of the 21st Annual Conference on Advances in Neural Information Processing Systems (NIPS), 2007.
[112] B. Recht, M. Fazel, and P. A. Parillo, *Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization*, SIAM Rev., 52 (2010), pp. 471–501.

[113] V. Rokhlin, A. Szlam, and M. Tygert, *A randomized algorithm for principal component analysis*, SIAM J. Matrix Anal. Appl., 31 (2009), pp. 1100–1124.
[114] 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.
[115] S. Roweis, *EM algorithms for PCA and SPCA*, in Proceedings of the 10th Annual Conference on Advances in Neural Information Processing Systems (NIPS), MIT Press, Cambridge, MA, 1997, pp. 626–632.
[116] M. Rudelson, *Random vectors in the isotropic position*, J. Funct. Anal., 164 (1999), pp. 60–72.
[117] M. Rudelson and R. Vershynin, *Sampling from large matrices: An approach through geometric functional analysis*, J. Assoc. Comput. Mach., 54 (2007), article 21.
[118] A. F. Ruston, *Auerbach's theorem*, Math. Proc. Cambridge Philos. Soc., 56 (1964), pp. 476–480.
[119] T. Sarlós, *Improved approximation algorithms for large matrices via random projections*, in Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 2006, pp. 143–152.
[120] S. Shalev-Shwartz and N. Srebro, *Low ℓ_1-norm and guarantees on sparsifiability*, in ICML/COLT/UAI Sparse Optimization and Variable Selection Workshop, 2008.
[121] X. Shen and F. G. Meyer, *Low-dimensional embedding of fMRI datasets*, Neuroimage, 41 (2008), pp. 886–902.
[122] N. D. Shyamalkumar and K. Varadarajan, *Efficient subspace approximation algorithms*, in Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2007, pp. 532–540.
[123] L. Sirovich and M. Kirby, *Low-dimensional procedure for the characterization of human faces.*, J. Optical Soc. Amer. A, 4 (1987), pp. 519–524.
[124] D. Spielman and N. Srivastasa, *Graph sparsification by effective resistances*, in Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC '08), 2008, pp. 563–568.
[125] G. Stewart, *Accelerating the orthogonal iteration for the eigenvectors of a Hermitian matrix*, Numer. Math., 13 (1969), pp. 362–376.
[126] G. W. Stewart, *On the perturbation of pseudo-inverses, projections and linear least squares problems*, SIAM Rev., 19 (1977), pp. 634–662.
[127] G. W. Stewart, *Four algorithms for the efficient computation of truncated pivoted QR approximations to a sparse matrix*, Numer. Math., 83 (1999), pp. 313–323.
[128] G. W. Stewart, *The decompositional approach to matrix computation*, Comput. Sci. Eng., 2 (1) (2000), pp. 50–59.
[129] T. Strohmer and R. Vershynin, *A randomized Kaczmarz algorithm with exponential convergence*, J. Fourier Anal. Appl., 15 (2009), pp. 262–278.
[130] J. Sun, Y. Xie, H. Zhang, and C. Faloutsos, *Less is more: Compact matrix decomposition for large sparse graphs*, Stat. Anal. Data Min., 1 (2008), pp. 6–22.
[131] S. J. Szarek, *Spaces with large distance from ℓ_∞^n and random matrices*, Amer. J. Math., 112 (1990), pp. 899–942.
[132] A. Szlam, M. Maggioni, and R. R. Coifman, *Regularization on graphs with function-adapted diffusion processes*, J. Mach. Learn. Res., 9 (2008), pp. 1711–1739.
[133] L. N. Trefethen and D. Bau III, *Numerical Linear Algebra*, SIAM, Philadelphia, PA, 1997.
[134] J. A. Tropp, *On the conditioning of random subdictionaries*, Appl. Comput. Harmon. Anal., 25 (2008), pp. 1–24.
[135] J. A. Tropp, *Improved analysis of the subsampled randomized Hadamard transform*, Adv. Adaptive Data Anal., 3 (2011), to appear.
[136] J. von Neumann and H. H. Goldstine, *Numerical inverting of matrices of high order*, Bull. Amer. Math. Soc., 53 (1947), pp. 1021–1099.
[137] J. von Neumann and H. H. Goldstine, *Numerical inverting of matrices of high order.* II, Proc. Amer. Math. Soc., 2 (1952), pp. 188–202.
[138] 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.