Randomized Numerical Linear Algebra: Foundations & Algorithms

Per-Gunnar Martinsson, University of Texas at Austin
Joel A. Tropp, California Institute of Technology

Abstract: This survey describes probabilistic algorithms for linear algebra computations, such as factorizing matrices and solving linear systems. It focuses on techniques that have a proven track record for real-world problems. The paper treats both the theoretical foundations of the subject and practical computational issues.

Topics include norm estimation; matrix approximation by sampling; structured and unstructured random embeddings; linear regression problems; low-rank approximation; subspace iteration and Krylov methods; error estimation and adaptivity; interpolatory and CUR factorizations; Nyström approximation of positive semidefinite matrices; single-view ("streaming") algorithms; full rank-revealing factorizations; solvers for linear systems; and approximation of kernel matrices that arise in machine learning and in scientific computing.

CONTENTS

1. INTRODUCTION

Numerical linear algebra (NLA) is one of the great achievements of scientific computing. On most computational platforms, we can now routinely and automatically solve small- and medium-scale linear algebra problems to high precision. The purpose of this survey is to describe a set of probabilistic techniques that have joined the mainstream of NLA over the last decade. These new techniques have accelerated everyday computations for small- and medium-size problems, and they have enabled large-scale computations that were beyond the reach of classical methods.

1.1. **Classical numerical linear algebra.** NLA definitively treats several major classes of problems, including

- solution of dense and sparse linear systems;
- orthogonalization, least-squares, and Tikhonov regularization;
- determination of eigenvalues, eigenvectors, and invariant subspaces;
- singular value decomposition (SVD) and total least-squares.

In spite of this catalog of successes, important challenges remain. The sheer scale of certain datasets (terabytes and beyond) makes them impervious to classical NLA algorithms. Modern computing architectures (GPUs, multi-core CPUs, massively distributed systems) are powerful, but this power can only be unleashed by algorithms that minimize data movement and that are designed *ab initio* with parallel computation in mind. New ways to organize and present data (out-of-core, distributed, streaming) also demand alternative techniques.

Randomization offers novel tools for addressing all of these challenges. This paper surveys these new ideas, provides detailed descriptions of algorithms with a proven track record, and outlines the mathematical techniques used to analyze these methods.

1.2. **Randomized algorithms emerge.** Probabilistic algorithms have held a central place in scientific computing ever since Ulam and von Neumann's groundbreaking work on Monte Carlo methods in the 1940s. For instance, Monte Carlo algorithms are essential for high-dimensional integration and for solving PDEs set in high-dimensional spaces. They also play a major role in modern machine learning and uncertainty quantification.

For many decades, however, numerical analysts regarded randomized algorithms as a method of last resort—to be invoked only in the absence of an effective deterministic alternative. Indeed, probabilistic techniques have several undesirable features. First, Monte Carlo methods often produce output with low accuracy. This is a consequence of the central limit theorem, and in many situations it cannot be avoided. Second, many computational scientists have a strong attachment to the engineering principle that two successive runs of the same algorithm should produce identical results. This requirement aids with debugging, and it can be critical for applications where safety is paramount, for example simulation of infrastructure or control of aircraft. Randomized methods do not generally offer this guarantee. (Controlling the seed of the random number generator can provide a partial work-around, but this necessarily involves additional complexity.)

Nevertheless, in the 1980s, randomized algorithms started to make inroads into NLA. Some of the early work concerns spectral computations, where it was already traditional to use random initialization. Dixon (1983) recognized that (a variant of) the power method with a random start *provably* approximates the largest eigenvalue of a positive semidefinite (PSD) matrix, even without a gap between the first and second eigenvalue. Kuczyński and Woźniakowski (1992) provided a sharp analysis of this phenomenon for both the power method and the Lanczos algorithm. Around the same time, Girard (1989) and Hutchinson (1990) proposed Monte Carlo methods for estimating the trace of a large psd matrix. Soon after, Parker (1995) demonstrated that randomized transformations can be used to avoid pivoting steps in Gaussian elimination.

Starting in the late 1990s, researchers in theoretical computer science identified other ways to apply probabilistic algorithms in NLA. Alon, Matias and Szegedy (1999) and Alon,

Gibbons, Matias and Szegedy (2002) showed that randomized embeddings allow for computations on streaming data with limited storage. (Papadimitriou, Raghavan, Tamaki and Vempala 2000) and (Frieze, Kannan and Vempala 2004) proposed Monte Carlo methods for low-rank matrix approximation. (Drineas, Kannan and Mahoney 2006*a*), (Drineas, Kannan and Mahoney 2006*b*), and (Drineas, Kannan and Mahoney 2006*c*) wrote the first statement of theoretical principles for randomized NLA. Sarlós (2006) showed how subspace embeddings support linear algebra computations.

In the mid-2000s, numerical analysts introduced practical randomized algorithms for low-rank matrix approximation and least-squares problems. This work includes the first computational evidence that randomized algorithms outperform classical NLA algorithms for particular classes of problems. Early contributions include (Martinsson, Rokhlin and Tygert 2006*a*, Liberty, Woolfe, Martinsson, Rokhlin and Tygert 2007, Rokhlin and Tygert 2008, Woolfe, Liberty, Rokhlin and Tygert 2008). These papers inspired later work, such as (Avron, Maymounkov and Toledo 2010, Halko, Martinsson and Tropp 2011*a*, Halko, Martinsson, Shkolnisky and Tygert 2011*b*), that has made a direct impact in applications

Parallel with the advances in numerical analysis, a tide of enthusiasm for randomized algorithms has flooded into cognate fields. In particular, stochastic gradient descent (Bottou 2010) has become a standard algorithm for solving large optimization problems in machine learning.

At the time of writing, in late 2019, randomized algorithms have joined the mainstream of NLA. They now appear in major reference works and textbooks (Golub and Van Loan 2013, Strang 2019). Key methods are being incorporated into standard software libraries ((NAG) 2019, Xiao, Gu and Langou 2017, Ghysels, Li, Gorman and Rouet 2017).

1.3. **What does randomness accomplish?** Over the course of this survey we will explore a number of different ways that randomization can be used to design effective NLA algorithms. For the moment, let us just summarize the most important benefits.

Randomized methods can handle certain NLA problems faster than any classical algorithm. In Section 10, we describe a randomized algorithm that can solve a dense $m \times n$ least-squares problem with $m \gg n$ using about $O(mn + n^3)$ arithmetic operations (Rokhlin and Tygert 2008). Meanwhile, classical methods require $O(mn^2)$ operations. In Section 18, we present an algorithm called SPARSECHOLESKY that can solve the Poisson problem on a dense undirected graph in time that is roughly *quadratic* in the number of vertices (Kyng and Sachdeva 2016). Standard methods have cost that is *cubic* in the number of vertices. The improvements can be even larger for sparse graphs.

Randomization allows us to tackle problems that otherwise seem impossible. Section 15 contains an algorithm that can compute a rank-r truncated SVD of an $m \times n$ matrix in a single pass over the data using working storage $O(r(m + n))$. The first reference for this kind of algorithm is Woolfe et al. (2008). We know of no classical method with this computational profile.

From an engineering point of view, randomization has another crucial advantage: it allows us to restructure NLA computations in a fundamentally different way. In Section 11, we will introduce the randomized SVD algorithm (Martinsson et al. 2006*a*, Halko et al. 2011*a*). Essentially all the arithmetic in this procedure takes place in a short sequence of matrix–matrix multiplications. Matrix multiplication is a highly optimized primitive on most computer systems; it parallelizes easily; and it performs particularly well on modern hardware such as GPUs. In contrast, classical SVD algorithms require either random access to the data or sequential matrix–vector multiplications. As a consequence, the randomized SVD can process matrices that are beyond the reach of classical SVD algorithms.

1.4. **Algorithm design considerations.** Before we decide what algorithm to use for a linear algebra computation, we must ask how we are permitted to interact with the data. A recurring theme of this survey is that randomization allows us to reorganize algorithms so that they control whichever computational resource is the most scarce (flops, communication, matrix entry evaluation, etc.). Let us illustrate with some representative examples:

- *Streaming computations ("single-view"):* There is rising demand for algorithms that can treat matrices that are so large that they cannot be stored at all; other applications involve matrices that are presented dynamically. In the streaming setting, the input matrix $\mathbf{A}$ is given by a sequence of simple linear updates that can viewed only once:

$$\mathbf{A} = \mathbf{H}_1 + \mathbf{H}_2 + \mathbf{H}_3 + \dots. \tag{1.1}$$

  We must discard each innovation $\mathbf{H}_i$ after it has been processed. As it happens, the *only* type of algorithm that can handle the model (1.1) is one based on randomized linear dimension reduction (Li, Nguyen and Woodruff 2014*b*). Our survey describes a number of algorithms that can operate in the streaming setting; see Sections 4, 5, 14, and 15.

- *Dense matrices stored in RAM:* One traditional computational model for NLA assumes that the input matrix is stored in fast memory, so that any entry can quickly be read and/or overwritten as needed. The ability of CPUs to perform arithmetic operations keeps growing rapidly, but memory latency has not kept up. Thus, it has become essential to formulate blocked algorithms that operate on submatrices. Section 16 shows how randomization can help.

- *Large sparse matrices:* For sparse matrices, it is natural to search for techniques that interact with a matrix only through its application to vectors, such as Krylov methods or subspace iteration. Randomization expands the design space for these methods. When the iteration is initialized with a random matrix, we can reach provably correct and highly accurate results after a few iterations; see Sections 11.6 and 11.7.

  Another idea is to apply randomized sampling to control sparsity levels. This technique arises in Section 18, which contains a randomized algorithm that accelerates incomplete Cholesky preconditioning for sparse graph Laplacians.

- *Matrices for which entry evaluation is expensive:* In machine learning and computational physics, it is often desirable to solve linear systems where it is too expensive to evaluate the full coefficient matrix. Randomization offers a systematic way to extract data from the matrix and to compute approximations that serve for the downstream applications. See Sections 19 and 20.

1.5. **Overview.** This paper covers fundamental mathematical ideas, as well as algorithms that have proved to be effective in practice. The balance shifts from theory at the beginning toward computational practice at the end. With the practitioner in mind, we have attempted to make the algorithmic sections self-contained, so that they can be read with a minimum of references to other parts of the paper.

After introducing notation and covering preliminaries from linear algebra and probability in Sections 2–3, the survey covers the following topics.

- Sections 4–5 discuss algorithms for trace estimation and Schatten p-norm estimation based on randomized sampling (i.e., Monte Carlo methods). Section 6 shows how iteration can improve the quality of estimates for maximum eigenvalues, maximum singular values, and trace functions.

- Section 7 develops randomized sampling methods for approximating matrices, including applications to matrix multiplication and approximation of combinatorial graphs.

- Sections 8–9 introduce the notion of a randomized linear embedding. These maps are frequently used to reduce the dimension of a set of vectors, while preserving their geometry. In Section 10, we explore several ways to use randomized embeddings in the context of an overdetermined least-squares problem.

- Sections 11–12 demonstrate how randomized methods can be used to find a subspace that is aligned with the range of a matrix and to assess the quality of this subspace. Sections 13, 14, and 15 show how to use this subspace to compute a variety of low-rank matrix approximations.

- Section 16 develops randomized algorithms for computing a factorization of a full-rank matrix, such as a pivoted QR decomposition or a URV decomposition.
- Section 17 describes some general approaches to solving linear systems using randomized techniques. Section 18 presents the SPARSECHOLESKY algorithm for solving the Poisson problem on an undirected graph (i.e., a linear system in a graph Laplacian).
- Last, Sections 19 and 20 show how to use randomized methods to approximate kernel matrices that arise in machine learning, computational physics, and scientific computing.

1.6. **Omissions.** While randomized NLA was a niche topic 15 years ago, we have seen an explosion of research over the last decade. This survey can only cover a small subset of the many important and interesting ideas that have emerged.

Among many other omissions, we do not discuss spectral computations in detail. There have been interesting and very recent developments, especially for the challenging problem of computing a spectral decomposition of a nonnormal matrix (Banks, Vargas, Kulkarni and Srivastava 2019). We also had to leave out a treatment of tensors and the rapidly developing field of randomized multilinear algebra.

There is no better way to demonstrate the value of a numerical method than careful numerical experiments that measure its speed and accuracy against state-of-the-art implementations of competing methods. Our selection of topics to cover is heavily influenced by such comparisons; for reasons of space, we have often had to settle for citations to the literature, instead of including the numerical evidence.

The intersection between optimization and linear algebra is of crucial importance in applications, and it remains a fertile ground for theoretical work. Randomized algorithms are invaluable in this context, but we realized early on that the paper would double in length if we included just the essential facts about randomized optimization algorithms.

There is a complementary perspective on randomized numerical analysis algorithms, called *probabilistic numerics*. See the website (Hennig and Osborne 2019) for a comprehensive bibliography.

For the topics that we do cover, we have made every effort to include all essential citations. Nevertheless, the literature is vast, and we are sure to have overlooked important work; we apologize in advance for these oversights.

1.7. **Other surveys.** There are a number of other survey papers on randomized algorithms for randomized NLA and related topics.

- Halko et al. (2011*a*) develop and analyse computational methods for low-rank matrix approximation (including the "randomized SVD" algorithm) from the point of view of a numerical analyst. The main idea is that randomization can furnish a subspace that captures the action of a matrix, and this subspace can be used to build structured low-rank matrix approximations.
- Mahoney (2011) treats randomized methods for least-squares computations and for low-rank matrix approximation. He emphasizes the useful principle that we can often decouple the linear algebra and the probability when analyzing randomized NLA algorithms.
- Woodruff (2014) describes how to use subspace embeddings as a primitive for developing randomized linear algebra algorithms. A distinctive feature is the development of lower bounds.
- Tropp (2015) gives an introduction to matrix concentration inequalities, and includes some applications to randomized NLA algorithms.
- The survey of Kannan and Vempala (2017) appeared in a previous volume of *Acta Numerica*. A unique aspect is the discussion of randomized tensor computations.

- Drineas and Mahoney (2018) have updated the presentation in Mahoney (2011), and include an introduction to linear algebra and probability that is directed toward NLA applications.
- Martinsson (2018) focuses on computational aspects of randomized NLA. A distinctive feature is the discussion of efficient algorithms for factorizing matrices of full, or nearly full, rank.
- Tropp (2019) gives a mathematical treatment of how matrix concentration supports a few randomized NLA algorithms, and includes a complete proof of correctness for the SPARSECHOLESKY algorithm described in Section 18.

1.8. **Acknowledgments.** We are grateful to Arieh Iserles for proposing that we write this survey. Both authors have benefited greatly from our collaborations with Vladimir Rokhlin and Mark Tygert. Most of all, we would like to thank Richard Kueng for his critical reading of the entire manuscript, which has improved the presentation in many places. Madeleine Udell, Riley Murray, James Levitt, and Abinand Gopal also gave us useful feedback on parts of the paper. Lorenzo Rosasco offered invaluable assistance with the section on kernel methods for machine learning. Navid Azizan, Babak Hassibi, and Peter Richtárik helped with citations to the literature on SGD. Finally, we would like to thank our ONR programme managers, Reza Malek-Madani and John Tague, for supporting research on randomized numerical linear algebra.

JAT acknowledges support from the Office of Naval Research (awards N-00014-17-1-2146 and N-00014-18-1-2363). PGM acknowledges support from the Office of Naval Research (award N00014-18-1-2354), from the National Science Foundation (award DMS-1620472), and from Nvidia Corp.

2. LINEAR ALGEBRA PRELIMINARIES

This section contains an overview of the linear algebra tools that arise in this survey. It collects the basic notation, along with some standard and not-so-standard definitions. It also contains a discussion about the role of the spectral norm.

Background references for linear algebra and matrix analysis include Bhatia (1997) and Horn and Johnson (2013). For a comprehensive treatment of matrix computations, we refer to (Golub and Van Loan 2013, Trefethen and Bau III 1997, Stewart 1998, Stewart 2001).

2.1. **Basics.** We will work in the real field ($\mathbb{R}$) or the complex field ($\mathbb{C}$). The symbol $\mathbb{F}$ refers to either the real or complex field, in cases where the precise choice is unimportant. As usual, scalars are denoted by lowercase italic Roman (a, b, c) or Greek (α, β) letters.

Vectors are elements of $\mathbb{F}^n$, where n is a natural number. We always denote vectors with lowercase bold Roman ($\mathbf{a}, \mathbf{b}, \mathbf{u}, \mathbf{v}$) or Greek ($\boldsymbol{\alpha}, \boldsymbol{\beta}$) letters. We write $\mathbf{0}$ for the zero vector and $\mathbf{1}$ for the vector of ones. The standard basis vectors are denoted as $\boldsymbol{\delta}_1, \dots, \boldsymbol{\delta}_n$. The dimensions of these special vectors are determined by context.

A general matrix is an element of $\mathbb{F}^{m \times n}$, where m, n are natural numbers. We always denote matrices with uppercase bold Roman ($\mathbf{A}, \mathbf{B}, \mathbf{C}$) or Greek ($\boldsymbol{\Delta}, \boldsymbol{\Lambda}$) letters. We write $\mathbf{0}$ for the zero matrix and $\mathbf{I}$ for the identity matrix; their dimensions are determined by a subscript or by context.

The parenthesis notation is used for indexing into vectors and matrices: $(\mathbf{a})_i$ is the ith coordinate of vector $\mathbf{a}$, while $(\mathbf{A})_{ij}$ is the (i, j)th coordinate of matrix $\mathbf{A}$. In some cases it is more convenient to invoke the functional form of indexing. For example, $\mathbf{A}(i, j)$ also refers to the (i, j)th coordinate of the matrix $\mathbf{A}$.

The colon notation is used to specify ranges of coordinates. For example, $(\mathbf{a})_{1:i}$ and $\mathbf{a}(1 : i)$ refer to the vector comprising the first i coordinates of $\mathbf{a}$. The colon by itself refers to the entire range of coordinates. For instance, $(\mathbf{A})_{i:}$ denotes the ith row of $\mathbf{A}$, while $(\mathbf{A})_{:j}$ denotes the jth column.

The symbol * is reserved for the (conjugate) transpose of a matrix of vector. A matrix that satisfies $\mathbf{A} = \mathbf{A}^*$ is said to be self-adjoint. It is convenient to distinguish the space $\mathbb{H}_n$ of

self-adjoint $n \times n$ matrices over the scalar field. We may write $\mathbb{H}_n(\mathbb{F})$ if it is necessary to specify the field.

The operator † extracts the Moore–Penrose pseudoinverse of a matrix. More precisely, for $\mathbf{A} \in \mathbb{F}^{m \times n}$, the pseudoinverse $\mathbf{A}^\dagger \in \mathbb{F}^{n \times m}$ is the unique matrix that satisfies the following:

(1) $\mathbf{A}\mathbf{A}^\dagger$ is self-adjoint.
(2) $\mathbf{A}^\dagger\mathbf{A}$ is self-adjoint.
(3) $\mathbf{A}\mathbf{A}^\dagger\mathbf{A} = \mathbf{A}$.
(4) $\mathbf{A}^\dagger\mathbf{A}\mathbf{A}^\dagger = \mathbf{A}^\dagger$.

If $\mathbf{A}$ has full column rank, then $\mathbf{A}^\dagger = (\mathbf{A}^*\mathbf{A})^{-1}\mathbf{A}^*$, where $(\cdot)^{-1}$ denotes the ordinary matrix inverse.

2.2. **Eigenvalues and singular values.** A positive semidefinite matrix is a self-adjoint matrix with nonnegative eigenvalues. We will generally abbreviate positive semidefinite to PSD. Likewise, a positive definite (PD) matrix is a self-adjoint matrix with positive eigenvalues.

The symbol $\preccurlyeq$ denotes the semidefinite order on self-adjoint matrices. The relation $\mathbf{A} \preccurlyeq \mathbf{B}$ means that $\mathbf{B} - \mathbf{A}$ is psd.

We write $\lambda_1 \geq \lambda_2 \geq \dots$ for the eigenvalues of a self-adjoint matrix. We write $\sigma_1 \geq \sigma_2 \geq \dots$ for the singular values of a general matrix. If the matrix is not clear from context, we may include it in the notation so that $\sigma_j(\mathbf{A})$ is the jth singular value of $\mathbf{A}$.

Let $f : \mathbb{R} \to \mathbb{R}$ be a function on the real line. We can extend f to a spectral function $f : \mathbb{H}_n \to \mathbb{H}_n$ on (conjugate) symmetric matrices. Indeed, for a matrix $\mathbf{A} \in \mathbb{H}_n$ with eigenvalue decomposition

$$\mathbf{A} = \sum_{i=1}^n \lambda_i \, \mathbf{u}_i\mathbf{u}_i^*, \quad \text{we define} \quad f(\mathbf{A}) := \sum_{i=1}^n f(\lambda_i) \, \mathbf{u}_i\mathbf{u}_i^*.$$

The Pascal notation for definitions ($:=$ and $=:$) is used sparingly, when we need to emphasize that the definition is taking place.

2.3. **Inner product geometry.** We equip $\mathbb{F}^n$ with the standard inner product and the associated ℓ_2 norm. For all vectors $\mathbf{a}, \mathbf{b} \in \mathbb{F}^n$,

$$\langle \mathbf{a}, \mathbf{b} \rangle := \mathbf{a} \cdot \mathbf{b} := \sum_{i=1}^n (\mathbf{a})_i^* (\mathbf{b})_i \quad \text{and} \quad \|\mathbf{a}\|^2 := \langle \mathbf{a}, \mathbf{a} \rangle.$$

We write $\mathbb{S}^{n-1}$ for the set of vectors in $\mathbb{F}^n$ with unit ℓ_2 norm. If needed, we may specify the field: $\mathbb{S}^{n-1}(\mathbb{F})$.

The trace of a square matrix is the sum of its diagonal entries:

$$\operatorname{trace}(\mathbf{A}) := \sum_{i=1}^n (\mathbf{A})_{ii} \quad \text{for } \mathbf{A} \in \mathbb{F}^{n \times n}.$$

Nonlinear functions bind before the trace. We equip $\mathbb{F}^{m \times n}$ with the standard trace inner product and the Frobenius norm. For all matrices $\mathbf{A}, \mathbf{B} \in \mathbb{F}^{m \times n}$,

$$\langle \mathbf{A}, \mathbf{B} \rangle := \operatorname{trace}(\mathbf{A}^*\mathbf{B}) \quad \text{and} \quad \|\mathbf{A}\|_{\mathrm{F}}^2 := \langle \mathbf{A}, \mathbf{A} \rangle.$$

For vectors, these definitions coincide with the ones in the last paragraph.

We say that a matrix $\mathbf{U}$ is *orthonormal* when its columns are orthonormal with respect to the standard inner product. That is, $\mathbf{U}^*\mathbf{U} = \mathbf{I}$. If $\mathbf{U}$ is also square, we say instead that $\mathbf{U}$ is *orthogonal* ($\mathbb{F} = \mathbb{R}$) or *unitary* ($\mathbb{F} = \mathbb{C}$).

2.4. **Norms on matrices.** Several different norms on matrices arise during this survey. We use consistent notation for these norms.

- The unadorned norm $\|\cdot\|$ refers to the spectral norm of a matrix, also known as the ℓ_2 operator norm. It reports the maximum singular value of its argument. For vectors, it coincides with the ℓ_2 norm.
- The norm $\|\cdot\|_*$ is the nuclear norm of a matrix, which is the dual of the spectral norm. It reports the sum of the singular values of its argument.

- The symbol $\|\cdot\|_{\mathrm{F}}$ refers to the Frobenius norm, defined in the last subsection. The Frobenius norm coincides with the ℓ_2 norm of the singular values of its argument.
- The notation $\|\cdot\|_p$ denotes the Schatten p-norm for each $p \in [1, \infty]$. The Schatten p-norm is the ℓ_p norm of the singular values of its argument. Special cases with their own notation include the nuclear norm (Schatten 1), the Frobenius norm (Schatten 2), and the spectral norm (Schatten ∞).

Occasionally, other norms may arise, and we will define them explicitly when they do.

2.5. **Approximation in the spectral norm.** Throughout this survey, we will almost exclusively use the spectral norm to measure the error in matrix computations. Let us recall some of the implications that follow from spectral norm bounds.

Suppose that $\mathbf{A} \in \mathbb{F}^{m\times n}$ is a matrix, and $\widehat{\mathbf{A}} \in \mathbb{F}^{m\times n}$ is an approximation. If the approximation satisfies the spectral norm error bound

$$\|\mathbf{A} - \widehat{\mathbf{A}}\| \leq \varepsilon,$$

then we can transfer the following information:

- **Linear functionals:** $|\langle \mathbf{F}, \mathbf{A}\rangle - \langle \mathbf{F}, \widehat{\mathbf{A}}\rangle| \leq \varepsilon \|\mathbf{F}\|_*$ for every matrix $\mathbf{F} \in \mathbb{F}^{m\times n}$
- **Singular values:** $|\sigma_j(\mathbf{A}) - \sigma_j(\widehat{\mathbf{A}})| \leq \varepsilon$ for each index j.
- **Singular vectors:** if the jth singular value $\sigma_j(\mathbf{A})$ is well separated from the other singular values, then the jth right singular vector of $\mathbf{A}$ is well approximated by the jth right singular vector of $\widehat{\mathbf{A}}$; a similar statement holds for the left singular vectors.

Detailed statements about the singular vectors are complicated, so we refer the reader to Bhatia (1997, Chaps. VII, X) for his treatment of perturbation of spectral subspaces.

For one-pass and streaming data models, it may not be possible to obtain good error bounds in the spectral norm. In this case, we may retrench to Frobenius norm or nuclear norm error bounds. These estimates give weaker information about linear functionals, singular values, and singular vectors.

Remark 2.1 (Frobenius norm approximation)**.** *In the literature on randomized NLA, some authors prefer to bound errors with respect to the Frobenius norm because the arguments are technically simpler. In many instances, these bounds are less valuable because the error can have the same scale as the matrix that we wish to approximate.*

For example, let us consider a variant of the spiked covariance model that is common in statistics applications (Johnstone 2001). Suppose we need to approximate a rank-one matrix contaminated with additive noise: $\mathbf{A} = \mathbf{u}\mathbf{u}^* + \varepsilon\mathbf{G} \in \mathbb{R}^{n\times n}$*, where* $\|\mathbf{u}\| = 1$ *and* $\mathbf{G} \in \mathbb{R}^{n\times n}$ *has independent* $\mathrm{NORMAL}(0, n^{-1})$ *entries. It is well known that* $\|\mathbf{G}\| \approx 2$*, while* $\|\mathbf{G}\|_{\mathrm{F}} \approx \sqrt{n}$*. With respect to the Frobenius norm, the zero matrix is almost as good an approximation of* $\mathbf{A}$ *as the rank-one matrix* $\mathbf{u}\mathbf{u}^*$*:*

$$\mathbb{E}\,\|\mathbf{A} - \mathbf{u}\mathbf{u}^*\|_{\mathrm{F}}^2 = \varepsilon^2 n \quad \textit{and} \quad \mathbb{E}\,\|\mathbf{A} - \mathbf{0}\|_{\mathrm{F}}^2 = \varepsilon^2 n + 1.$$

The difference is visible only when the size of the perturbation $\varepsilon \approx n^{-1/2}$*. In contrast, the spectral norm error can easily distinguish between the good approximation* $\mathbf{u}\mathbf{u}^*$ *and the vacuous approximation* $\mathbf{0}$*, even when* $\varepsilon = O(1)$*.*

For additional discussion, see Tropp (2015, Sec. 6.2.3) and Li, Linderman, Szlam, Stanton, Kluger and Tygert (2017, App.).

2.6. **Intrinsic dimension and stable rank.** Let $\mathbf{A} \in \mathbb{H}_n$ be a psd matrix. We define its *intrinsic dimension*:

$$\mathrm{intdim}(\mathbf{A}) := \frac{\mathrm{trace}(\mathbf{A})}{\|\mathbf{A}\|}. \tag{2.1}$$

The intrinsic dimension of a nonzero matrix satisfies the inequalities $1 \leq \mathrm{intdim}(\mathbf{A}) \leq \mathrm{rank}(\mathbf{A})$; the upper bound is saturated when $\mathbf{A}$ is an orthogonal projector. We can interpret the intrinsic dimension as a continuous measure of the rank, or the number of energetic dimensions in the matrix.

Let $\mathbf{B} \in \mathbb{F}^{m \times n}$ be a rectangular matrix. Its *stable rank* is

$$\operatorname{srank}(\mathbf{B}) := \operatorname{intdim}(\mathbf{B}^*\mathbf{B}) = \frac{\|\mathbf{B}\|_{\mathrm{F}}^2}{\|\mathbf{B}\|^2}. \tag{2.2}$$

Similar to the intrinsic dimension, the stable rank provides a continuous measure of the rank of $\mathbf{B}$.

2.7. **Schur complements.** Schur complements arise from partial Gaussian elimination and partial least-squares. They also play a key role in several parts of randomized NLA. We give the basic definitions here, referring to Zhang (2005) for a more complete treatment.

Let $\mathbf{A} \in \mathbb{F}^{n \times n}$ be a psd matrix, and let $\mathbf{X} \in \mathbb{F}^{n \times k}$ be a fixed matrix. First, define the psd matrix

$$\mathbf{A}\langle\mathbf{X}\rangle := (\mathbf{A}\mathbf{X})(\mathbf{X}^*\mathbf{A}\mathbf{X})^\dagger(\mathbf{A}\mathbf{X})^*. \tag{2.3}$$

The *Schur complement* of $\mathbf{A}$ with respect to $\mathbf{X}$ is the psd matrix

$$\mathbf{A}/\mathbf{X} := \mathbf{A} - \mathbf{A}\langle\mathbf{X}\rangle. \tag{2.4}$$

The matrices $\mathbf{A}\langle\mathbf{X}\rangle$ and $\mathbf{A}/\mathbf{X}$ depend on $\mathbf{X}$ only through its range. They also enjoy geometric interpretations in terms of orthogonal projections with respect to the $\mathbf{A}$ semi-inner-product.

2.8. **Miscellaneous.** We use big-O notation following standard computer science convention. For instance, we say that a method has (arithmetic) complexity $O(n^\omega)$ if there is a finite C for which the number of floating point operations (flops) expended is bounded by Cn^ω as the problem size $n \to \infty$.

We use MATLAB-inspired syntax in summarizing algorithms. For instance, the task of computing an SVD $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*$ of a given matrix $\mathbf{A}$ is written as $[\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}] = \texttt{svd}(\mathbf{A})$. We have taken the liberty to modify the syntax when we believe that this improves clarity. For instance, we write $[\mathbf{Q}, \mathbf{R}] = \texttt{qr\_econ}(\mathbf{A})$ to denote the *economy-size* QR factorization where the matrix $\mathbf{Q}$ has size $m \times \min(m, n)$ for an input matrix $\mathbf{A} \in \mathbb{F}^{m \times n}$. Arguments that are not needed are replaced by "$\sim$", so that, for example, $[\mathbf{Q}, \sim] = \texttt{qr\_econ}(\mathbf{A})$ returns only the matrix $\mathbf{Q}$ whose columns form an ON basis for the range of $\mathbf{A}$.

3. Probability preliminaries

This section summarizes the key definitions and background from probability and high-dimensional probability. Later in the survey, we will present more complete statements of foundational results, as they are needed.

Grimmett and Stirzaker (2001) provide an accessible overview of applied probability. Vershynin (2018) introduces the field of high-dimensional probability. For more mathematical presentations, see the classic book of Ledoux and Talagrand (1991) or the lecture notes of Van Handel (2016).

3.1. **Basics.** We work in a master probability space that is rich enough to support all of the random variables that are defined. We will not comment further about the underlying model.

In this paper, the unqualified term *random variable* encompasses random scalars, vectors, and matrices. Scalar-valued random variables are usually (but not always) denoted by uppercase italic Roman letters (X, Y, Z). A random vector is denoted by a lowercase bold letter ($\mathbf{x}, \boldsymbol{\omega}$). A random matrix is denoted by an uppercase bold letter ($\mathbf{X}, \mathbf{Y}, \mathbf{\Gamma}, \mathbf{\Omega}$). This notation works in concert with the notation for deterministic vectors and matrices.

The map $\mathbb{P}(E)$ returns the probability of an event E. We usually specify the event using the compact set builder notation that is standard in probability. For example, $\mathbb{P}\{X > t\}$ is the probability that the scalar random variable X exceeds a level t.

The operator $\mathbb{E}$ returns the expectation of a random variable. For vectors and matrices, the expectation can be computed coordinate by coordinate. The expectation is linear, which justifies relations like

$$\mathbb{E}[\mathbf{A}\mathbf{X}] = \mathbf{A}\,\mathbb{E}[\mathbf{X}] \quad \text{when } \mathbf{A} \text{ is deterministic and } \mathbf{X} \text{ is random.}$$

We use the convention that nonlinear functions bind before the expectation; for instance, $\mathbb{E}\, X^2 = \mathbb{E}[X^2]$. The operator $\mathrm{Var}[\cdot]$ returns the variance of a scalar random variable.

We say that a random variable is *centered* when its expectation equals zero. A random vector $\mathbf{x}$ is *isotropic* when $\mathbb{E}[\mathbf{x}\mathbf{x}^*] = \mathbf{I}$. A random vector is *standardized* when it is both centered and isotropic. In particular, a scalar random variable is standardized when it has expectation zero and variance one.

When referring to independent random variables, we often include the qualification "statistically independent" to make a distinction with "linearly independent." We abbreviate the term (statistically) "independent and identically distributed" as i.i.d.

3.2. **Distributions.** To refer to a named distribution, we use small capitals. In this context, the symbol $\sim$ means "has the same distribution as."

We write UNIF for the uniform distribution over a finite set (with counting measure). In particular, a scalar Rademacher random variable has the distribution $\textsc{unif}\{\pm 1\}$. A Rademacher random vector has iid coordinates, each distributed as a scalar Rademacher random variable. We sometimes require the uniform distribution over a Borel subset of $\mathbb{F}^n$, equipped with Lebesgue measure.

We write $\textsc{normal}(\boldsymbol{\mu}, \mathbf{C})$ for the normal distribution on $\mathbb{F}^n$ with expectation $\boldsymbol{\mu} \in \mathbb{F}^n$ and psd covariance matrix $\mathbf{C} \in \mathbb{H}_n(\mathbb{F})$. A *standard normal* random variable or random vector has expectation zero and covariance matrix equal to the identity. We often use the term *Gaussian* to refer to normal distributions.

3.3. **Concentration inequalities.** Concentration inequalities provide bounds on the probability that a random variable is close to its expectation. A good reference for the scalar case is the book by Boucheron, Lugosi and Massart (2013). In the matrix setting, closeness is measured in the spectral norm. For an introduction to matrix concentration, see (Tropp 2015, Tropp 2019). These results play an important role in randomized linear algebra.

3.4. **Gaussian random matrix theory.** On several occasions, we use comparison principles to study the action of a random matrix with iid Gaussian entries. In particular, these methods can be used to control the largest and smallest singular values. The main classical comparison theorems are associated with the names Slepian, Chevet, and Gordon. For accounts of this, see Ledoux and Talagrand (1991, Sec. 3.3) or Davidson and Szarek (2001, Sec. 2.3) or Vershynin (2018, Secs. 7.2–7.3). More recently, it has been observed that Gordon's inequality can be reversed in certain settings (Thrampoulidis, Oymak and Hassibi 2014).

In several instances, we require more detailed information about Gaussian random matrices. General resources include Muirhead (1982) and Bai and Silverstein (2010). Most of the specific results we need are presented in Halko et al. (2011*a*).

4. Trace estimation by sampling

We commence with a treatment of matrix trace estimation problems. These questions stand among the simplest linear algebra problems because the desired result is just a scalar. Even so, the algorithms have a vast sweep of applications, ranging from computational statistics to quantum chemistry. They also serve as building blocks for more complicated randomized NLA algorithms. We have chosen to begin our presentation here because many of the techniques that drive algorithms for more difficult problems already appear in a nascent—and especially pellucid—form in this section.

Randomized methods for trace estimation depend on a natural technical idea: One may construct an unbiased estimator for the trace and then average independent copies to reduce the variance of the estimate. Algorithms of this type are often called *Monte Carlo methods*. We describe how to use standard methods from probability and statistics to develop *a priori* and *a posteriori* guarantees for Monte Carlo trace estimators. We show how to use structured random distributions to improve the computational profile of the estimators. Last, we demonstrate that trace estimators also yield approximations for the Frobenius norm and the Schatten 4-norm of a general matrix.

In Section 5, we present more involved Monte Carlo methods that are required to estimate Schatten p-norms for larger values of p, which give better approximations for the spectral norm. Section 6 describes iterative algorithms that lead to much higher accuracy than Monte Carlo methods. In Section 6.6, we touch on related probabilistic techniques for evaluating trace functions.

4.1. **Overview.** We will focus on the problem of estimating the trace of a nonzero psd matrix $\mathbf{A} \in \mathbb{H}_n$. Our goal is to to produce an approximation of $\operatorname{trace}(\mathbf{A})$, along with a measure of quality.

Trace estimation is easy in the case where we have inexpensive access to the entries of the matrix $\mathbf{A}$ because we can simply read off the n diagonal entries. But there are many environments where the primitive operation is the matrix–vector product $\mathbf{u} \mapsto \mathbf{A}\mathbf{u}$. For example, $\mathbf{u} \mapsto \mathbf{A}\mathbf{u}$ might be the solution of a (discretized) linear differential equation with initial condition $\mathbf{u}$, implemented by some computer program. In this case, we would really prefer to avoid n applications of the primitive. (We can obviously compute the trace by applying the primitive to each standard basis vector $\boldsymbol{\delta}_i$.)

The methods in this section all use linear information about the matrix $\mathbf{A}$. In other words, we will extract data from the input matrix by computing the product $\mathbf{Y} = \mathbf{A}\boldsymbol{\Omega}$, where $\boldsymbol{\Omega} \in \mathbb{F}^{n \times k}$ is a (random) test matrix. All subsequent operations involve only the sample matrix $\mathbf{Y}$ and the test matrix $\boldsymbol{\Omega}$. Since the data collection process is linear, we can apply randomized trace estimators in the one-pass or streaming environments. Moreover, parts of these algorithms are trivially parallelizable.

The original application of randomized trace estimation was to perform *a posteriori* error estimation for large least-squares computations. More specifically, it was used to accelerate cross-validation procedures for estimating the optimal regularization parameter in a smoothing spline (Girard 1989, Hutchinson 1990). See Fitzsimons, Osborne, Roberts and Fitzsimons (2018) for a list of contemporary applications in machine learning, uncertainty quantification, and other fields.

4.2. **Trace estimation by randomized sampling.** Randomized trace estimation is based on the insight that it is easy to construct a random variable whose expectation equals the trace of the input matrix.

Consider a random test vector $\boldsymbol{\omega} \in \mathbb{F}^n$ that is isotropic: $\mathbb{E}[\boldsymbol{\omega}\boldsymbol{\omega}^*] = \mathbf{I}$. By the cyclicity of the trace and by linearity,

$$X = \boldsymbol{\omega}^*(\mathbf{A}\boldsymbol{\omega}) \quad \text{satisfies} \quad \mathbb{E}\, X = \operatorname{trace}(\mathbf{A}). \tag{4.1}$$

In other words, the random variable X is an unbiased estimator of the trace. Note that the distribution of X depends on the unknown matrix $\mathbf{A}$.

A single sample of X is rarely adequate because its variance, $\operatorname{Var}[X]$, will be large. The most common mechanism for reducing the variance is to average k independent copies of X. For $k \in \mathbb{N}$, define

$$\bar{X}_k = \frac{1}{k} \sum\nolimits_{i=1}^{k} X_i \quad \text{where } X_i \sim X \text{ are iid.} \tag{4.2}$$

By linearity, $\bar{X}_k$ is also an unbiased estimator of the trace. The individual samples are statistically independent, so the variance decreases. Indeed,

$$\mathbb{E}[\bar{X}_k] = \operatorname{trace}(\mathbf{A}) \quad \text{and} \quad \operatorname{Var}[\bar{X}_k] = \frac{1}{k} \operatorname{Var}[X].$$

The estimator (4.2) can be regarded as the most elementary method in randomized linear algebra. See Algorithm 1.

To compute $\bar{X}_k$, we must simulate k independent copies of the random vector $\boldsymbol{\omega} \in \mathbb{F}^n$ and perform k matrix–vector products with $\mathbf{A}$, plus $O(kn)$ additional arithmetic.

Algorithm 1 *Trace estimation by random sampling.*
See Section 4.2.

Input: Psd input matrix $\mathbf{A} \in \mathbb{H}_n$, number k of samples
Output: Trace estimate $\bar{X}_k$ and sample variance S_k

function TRACEESTIMATE($\mathbf{A}$, k)
  for $i = 1, \dots, k$ **do** ▷ Compute trace samples
    Draw isotropic test vector $\boldsymbol{\omega}_i \in \mathbb{F}^n$
    Compute $X_i = \boldsymbol{\omega}_i^* (\mathbf{A}\boldsymbol{\omega}_i)$
  Form trace estimator: $\bar{X}_k = k^{-1} \sum_{i=1}^k X_i$
  Form sample variance: $S_k = (k-1)^{-1} \sum_{i=1}^k (X_i - \bar{X}_k)^2$
  ▷ Use compensated summation techniques for large k!

Example 4.1 (Girard (1989))**.** *Consider a standard normal random vector* $\boldsymbol{\omega} \sim \text{NORMAL}(\mathbf{0}, \mathbf{I})$. *The variance of the resulting trace estimator* (4.1)–(4.2) *satisfies*

$$\text{Var}[\bar{X}_k] = \frac{2}{k} \sum_{i,j=1}^n |(\mathbf{A})_{ij}|^2 = \frac{2}{k} \|\mathbf{A}\|_{\text{F}}^2 \leq \frac{2}{k} \|\mathbf{A}\| \, \text{trace}(\mathbf{A}). \tag{4.3}$$

The rotational invariance of the standard normal distribution allows us to characterize the behavior of this estimator in full detail.

Example 4.2 (Hutchinson (1990))**.** *Consider a Rademacher random vector* $\boldsymbol{\omega} \sim \text{UNIF}\{\pm 1\}^n$. *The variance of the resulting trace estimator* (4.1)–(4.2) *satisfies*

$$\text{Var}[\bar{X}_k] = \frac{4}{k} \sum_{1 \leq i < j \leq n} |(\mathbf{A})_{ij}|^2 < \frac{2}{k} \|\mathbf{A}\|_{\text{F}}^2 \leq \frac{2}{k} \|\mathbf{A}\| \, \text{trace}(\mathbf{A}).$$

This is the minimum variance trace estimator generated by an isotropic random vector $\boldsymbol{\omega}$ *with statistically independent coordinates. It also avoids the simulation of normal variables.*

Girard (1989) also studied the estimator obtained by drawing $\boldsymbol{\omega} \in \mathbb{F}^n$ uniformly at random from the sphere $\sqrt{n}\, \mathbb{S}^{n-1}(\mathbb{F})$ for $\mathbb{F} = \mathbb{R}$. When $\mathbb{F} = \mathbb{C}$, this approach has the minimax variance among all trace estimators of the form (4.1)–(4.2). We return to this example in Section 4.7.1.

Remark 4.3 (General matrices)**.** *The assumption that* $\mathbf{A}$ *is psd allows us to conclude that the standard deviation of the randomized trace estimate is smaller than the trace of the matrix. The same methods allow us to estimate the trace of a general square matrix, but the variance of the estimator may no longer be comparable with the trace.*

4.3. **A priori error estimates.** We can use theoretical analysis to obtain prior guarantees on the performance of the trace estimator. These results illuminate what features of the input matrix affect the quality of the trace estimate, and they tell us how many samples k suffice to achieve a given error tolerance. Note, however, that these bounds depend on properties of the input matrix that are often unknown to the user of the trace estimator.

Regardless of the distribution of the isotropic test vector $\boldsymbol{\omega}$, Chebyshev's inequality delivers a simple probability bound for the trace estimator:

$$\mathbb{P}\left\{ |\bar{X}_k - \text{trace}(\mathbf{A})| \geq t \right\} \leq \frac{\text{Var}[X]}{kt^2} \quad \text{for } t > 0. \tag{4.4}$$

We can specialize this result to specific trace estimators by inserting the variance.

Example 4.4 (Girard trace estimator)**.** *If the test vector* $\boldsymbol{\omega}$ *is standard normal, the trace estimator* $\bar{X}_k$ *satisfies*

$$\mathbb{P}\left\{ |\bar{X}_k - \text{trace}(\mathbf{A})| \geq t \cdot \text{trace}(\mathbf{A}) \right\} \leq \frac{2}{k \, \text{intdim}(\mathbf{A}) \, t^2}.$$

The bound follows from (4.3)*,* (4.4)*, and* (2.1)*. In words, the trace estimator achieves a relative error bound that is sharpest when the intrinsic dimension* (2.1) *of* $\mathbf{A}$ *is large.*

For specific distributions of the random test vector $\boldsymbol{\omega}$, we can obtain much stronger probability bounds for the resulting trace estimator using exponential concentration inequalities. Here is a recent analysis for Girard's estimator based on fine properties of the standard normal distribution.

Theorem 4.5 (Gratton and Titley-Peloquin (2018))**.** *Let* $\mathbf{A} \in \mathbb{H}_n(\mathbb{R})$ *be a nonzero psd matrix. Consider the trace estimator* (4.1)–(4.2) *obtained from a standard normal test vector* $\boldsymbol{\omega} \in \mathbb{R}^n$*. For* $\tau > 1$ *and* $k \leq n$*,*

$$\mathbb{P}\left\{\bar{X}_k \geq \tau \operatorname{trace}(\mathbf{A})\right\} \leq \exp\left(-\tfrac{1}{2}k \operatorname{intdim}(\mathbf{A})\big(\sqrt{\tau} - 1\big)^2\right);$$
$$\mathbb{P}\left\{\bar{X}_k \leq \tau^{-1} \operatorname{trace}(\mathbf{A})\right\} \leq \exp\left(-\tfrac{1}{4}k \operatorname{intdim}(\mathbf{A})\big(\tau^{-1} - 1\big)^2\right).$$

When $\mathbf{A} \in \mathbb{H}_n(\mathbb{C})$ *is psd and* $\boldsymbol{\omega} \in \mathbb{C}^n$ *is complex standard normal, the same bounds hold with an extra factor two in the exponent. (So the estimator works better in the complex setting.)*

Proof. (Sketch) Carefully estimate the moment generating function of the random variable X, and use the Cramér–Chernoff method to obtain the probability inequalities. □

4.4. **Universality.** Empirically, for a large sample, the performance of the trace estimator $\bar{X}_k$ only depends on the distribution of the test vector $\boldsymbol{\omega}$ through the variance of the resulting sample X. In a word, the estimator exhibits *universality*. As a consequence, we can select the distribution that is most convenient for computational purposes.

Classical probability theory furnishes justification for these claims. The strong law of large numbers tells us that

$$\bar{X}_k \to \operatorname{trace}(\mathbf{A}) \quad \text{almost surely as } k \to \infty.$$

Concentration inequalities (Boucheron et al. 2013) allow us to derive rates of convergence akin to Theorem 4.5.

To understand the sampling distribution of the estimator $\bar{X}_k$, we can invoke the central limit theorem:

$$\sqrt{k}(\bar{X}_k - \operatorname{trace}(\mathbf{A})) \to \textsc{normal}(0, \operatorname{Var}[X]) \quad \text{in distribution as } k \to \infty.$$

We can obtain estimates for the rate of convergence to normality using the Berry–Esséen theorem and its variants (Ross 2011, Chen, Goldstein and Shao 2011).

Owing to the universality phenomenon, we can formally use the normal limit to obtain heuristic error estimates and insights for trace estimators constructed with test vectors from any distribution. This strategy becomes even more valuable when the linear algebra problem is more complicated.

Warning 4.6 (CLT)**.** *Estimators based on averaging independent samples cannot overcome the central limit theorem. Their accuracy will always be limited by fluctuations on the scale of* $\sqrt{\operatorname{Var}[X]}$*. In other words, we must extract* ε^{-2} *samples to reduce the error to* $\varepsilon\sqrt{\operatorname{Var}[X]}$ *for small* $\varepsilon > 0$*. This is the curse of Monte Carlo.*

4.5. **A posteriori error estimates.** In practice, we rarely have access to all the information required to activate *a priori* error bounds. It is wiser to assess the quality of the estimate from the information that we actually collect. Since we have full knowledge of the random process that generates the trace estimate, we can confidently use approaches from classical statistics.

At the most basic level, the sample variance is an unbiased estimator for the variance of the individual samples:

$$S_k = \frac{1}{k-1}\sum_{i=1}^{k}(X_i - \bar{X}_k)^2 \quad \text{satisfies} \quad \mathbb{E}[S_k] = \operatorname{Var}[X].$$

Algorithm 2 *Bootstrap confidence interval for trace estimation.*
See Section 4.6.

Input: Psd input matrix $\mathbf{A} \in \mathbb{H}_n$, number k of trace samples, number B of bootstrap replicates, parameter α for level of confidence
Output: Confidence interval $[\bar{X}_k + q_\alpha, \bar{X}_k + q_{1-\alpha}]$ at level $1 - 2\alpha$

function BOOTSTRAPTRACEESTIMATE($\mathbf{A}$, k, b, α)
  for $i = 1, \dots, k$ **do** ▷ Compute trace estimators
    Draw isotropic test vector $\boldsymbol{\omega}_i \in \mathbb{F}^n$
    Form $X_i = \boldsymbol{\omega}_i^*(\mathbf{A}\boldsymbol{\omega}_i)$
  $\mathcal{X} = (X_1, \dots, X_k)$ ▷ Collate sample
  $\bar{X}_k = k^{-1} \sum_{i=1}^k X_i$ ▷ Trace estimate
  for $b = 1, \dots, B$ **do** ▷ Bootstrap replicates
    Draw $(X_1^*, \dots, X_k^*)$ from $\mathcal{X}$ with replacement
    Compute $e_b^* = (k^{-1} \sum_{i=1}^k X_i^*) - \bar{X}_k$
  Find q_α and $q_{1-\alpha}$ quantiles of errors $(e_1^*, \dots, e_b^*)$

The variance of S_k depends on the fourth moment of the random variable X. A standard estimate is

$$\operatorname{Var}[S_k] \le \frac{1}{k} \mathbb{E}[(X - \mathbb{E}\, X)^4].$$

Bounds and empirical estimates for the variance of S_k can also be obtained using the Efron–Stein inequality (Boucheron et al. 2013, Sec. 3.1).

For $\alpha \in (0, 1/2)$, we can construct the (symmetric, Student's t) confidence interval at level $1 - 2\alpha$:

$$\operatorname{trace}(\mathbf{A}) \in \bar{X}_k \pm t_{\alpha,k-1} \sqrt{S_k}$$

where $t_{\alpha,k-1}$ is the α quantile of the Student's t-distribution with $k - 1$ degrees of freedom. We interpret this result as saying that $\operatorname{trace}(\mathbf{A})$ lies in the specified interval with probability roughly $1-2\alpha$ (over the randomness in the trace estimator). The usual rule of thumb is that the sample size should be moderate (say, $k \ge 30$), while α cannot be too small (say, $\alpha \ge 0.025$).

4.6. **Bootstrapping the sampling distribution.** Miles Lopes has proposed a sweeping program that uses the bootstrap to construct data-driven confidence sets for randomized NLA algorithms (Lopes 2019). For trace estimation, this approach is straightforward to describe and implement; see Algorithm 2.

Let $\mathcal{X} = (X_1, \dots, X_k)$ be the empirical sample from (4.2). The bootstrap draws further random samples from $\mathcal{X}$ to elicit more information about the sampling distribution of the trace estimator X, such as confidence sets.

(1) For each $b = 1, \dots, B$,
  (a) Draw a bootstrap replicate $(X_1^*, \dots, X_k^*)$ uniformly from $\mathcal{X}$ with replacement.
  (b) Compute the error estimate $e_b^* = \bar{X}_k^* - \bar{X}_k$, where $\bar{X}_k^*$ is the sample average of the bootstrap replicate.
(2) Compute quantiles q_α and $q_{1-\alpha}$ of the error distribution $(e_1^*, \dots, e_B^*)$.
(3) Report the $1 - 2\alpha$ confidence set $[\bar{X}_k + q_\alpha, \bar{X}_k + q_{1-\alpha}]$.

Typical values are $k \ge 30$ samples and $B \ge 1000$ bootstrap replicates when $\alpha \ge 0.025$. This method is effective for a wide range of distributions on the test vector, and it extends to other problems. See Efron (1982) for an introduction to resampling methods.

4.7. **Structured distributions for test vectors.** As discussed, there is a lot of flexibility in designing the distribution of the test vector. We can exploit this freedom to achieve additional computational goals. For example, we might:

- minimize the variance $\operatorname{Var}[X]$ of each sample;
- minimize the number of random bits required to construct $\boldsymbol{\omega}$;

- design a test vector $\boldsymbol{\omega}$ that is "compatible" with the input matrix $\mathbf{A}$ to facilitate the matrix–vector product. For example, if $\mathbf{A}$ has a tensor product structure, we might require $\boldsymbol{\omega}$ to share the tensor structure.

Let us describe a general construction for test vectors that can help achieve these desiderata. The ideas come from frame theory and quantum information theory. The approach here extends the work in Fitzsimons et al. (2018).

4.7.1. *Optimal measurement systems.* In this section, we work in the complex field. Consider a discrete set $\mathcal{U} := \{\mathbf{u}_1, \dots, \mathbf{u}_m\} \subset \mathbb{C}^n$ of vectors, each with unit ℓ_2 norm. We say that the $\mathcal{U}$ is an *optimal measurement system* when

$$\frac{1}{m} \sum\nolimits_{i=1}^{m} (\mathbf{u}_i^* \mathbf{M} \mathbf{u}_i)\, \mathbf{u}_i \mathbf{u}_i^* = \frac{1}{(n+1)n} \left[\mathbf{M} + \operatorname{trace}(\mathbf{M})\, \mathbf{I} \right] \tag{4.5}$$

for all $\mathbf{M} \in \mathbb{H}_n(\mathbb{C})$. The reproducing property (4.5) shows that the system of vectors acquires enough information to reconstruct an arbitrary self-adjoint matrix. A similar definition is valid for an infinite system of unit vectors, provided we replace the sum in (4.5) with an integral.

Now, suppose that we draw a random test vector $\boldsymbol{\omega} = \sqrt{n}\, \mathbf{u}$, where $\mathbf{u}$ is drawn uniformly at random from an optimal measurement system $\mathcal{U}$. Then the resulting trace estimator is unbiased:

$$X = \boldsymbol{\omega}^* (\mathbf{A}\boldsymbol{\omega}) \quad \text{satisfies} \quad \mathbb{E}\, X = \operatorname{trace}(\mathbf{A}). \tag{4.6}$$

The variance of this trace estimator satisfies

$$\operatorname{Var}[X] = \frac{n}{n+1} \left[\|\mathbf{A}\|_{\mathrm{F}}^2 - \frac{1}{n} \operatorname{trace}(\mathbf{A})^2 \right]. \tag{4.7}$$

The identities (4.6) and (4.7) follow quickly from (4.5). As it happens, this is the minimax variance achievable for a best isotropic distribution on test vectors (Kueng 2019)

4.7.2. *Examples.* Optimal measurement systems arise in quantum information theory (as near-isotropic measurement systems), in approximation theory (as projective 2-designs), and in frame theory (as tight fusion frames). The core examples are as follows:

(1) A set of n^2 equiangular lines in $\mathbb{C}^n$, each spanned by a unit vector in $\{\mathbf{u}_1, \dots, \mathbf{u}_{n^2}\}$, gives an optimal measurement system. In this case, equiangularity means that $|\langle \mathbf{u}_i, \mathbf{u}_j \rangle|^2 = (d+1)^{-1}$ whenever $i \neq j$. It is conjectured that these systems exist for every natural number n.

(2) The columns of a family of $(n+1)$ mutually unbiased bases in $\mathbb{F}^n$ compose an optimal measurement system with $n(n+1)$ unit vectors. For reference, a pair $(\mathbf{U}, \mathbf{V})$ of $n \times n$ unitary matrices is called *mutually unbiased* if $\boldsymbol{\delta}_i^* (\mathbf{U}^* \mathbf{V}) \boldsymbol{\delta}_j = n^{-1}$ for all i, j. (For instance, consider the identity matrix and the discrete Fourier transform matrix.) These systems exist whenever n is a power of a prime number (Wootters and Fields 1989).

(3) The ℓ_2 unit sphere $\mathbb{S}^{n-1}(\mathbb{C})$ in $\mathbb{C}^n$, equipped with the uniform measure, is a continuous optimal measurement system. The real case was studied by Girard (1989), but the complex case is actually more natural.

See Tropp (2019, Lec. 3) for more discussion and an application to quantum state tomography. Waldron (2018) provides a good survey of what is currently known about finite optimal measurement systems.

4.8. **Extension: The Frobenius norm and the Schatten 4-norm.** The randomized trace estimators developed in this section can also be deployed to estimate a couple of matrix norms.

Consider a rectangular matrix $\mathbf{B} \in \mathbb{F}^{m \times n}$, accessed via the matrix–vector product $\mathbf{u} \mapsto \mathbf{B}\mathbf{u}$. Let us demonstrate how to estimate the Frobenius norm (i.e., Schatten 2-norm) and the Schatten 4-norm of $\mathbf{B}$.

For concreteness, suppose that we extract test vectors from the standard normal distribution. Draw a standard normal matrix $\mathbf{\Omega} \in \mathbb{F}^{n\times k}$ with columns $\boldsymbol{\omega}_i$. Construct the random variable

$$\bar{X}_k := \frac{1}{k}\|\mathbf{B\Omega}\|_{\mathrm{F}}^2 = \frac{1}{k}\sum_{i=1}^{k} \boldsymbol{\omega}_i^*(\mathbf{B}^*\mathbf{B})\boldsymbol{\omega}_i =: \frac{1}{k}\sum_{i=1}^{k} X_i.$$

We compute $\bar{X}_k$ by simulating nk standard normal variables, taking k matrix–vector products with $\mathbf{B}$, and performing $O(km)$ additional arithmetic.

To analyze $\bar{X}_k$, note that it is an instance of the randomized trace estimator, where $\mathbf{A} = \mathbf{B}^*\mathbf{B}$. In particular, its statistics are

$$\mathbb{E}[\bar{X}_k] = \|\mathbf{B}\|_{\mathrm{F}}^2 \quad \text{and} \quad \mathrm{Var}[\bar{X}_k] = \frac{2}{k}\|\mathbf{B}\|_4^4.$$

We see that $\bar{X}_k$ provides an unbiased estimate for the squared Frobenius norm of the matrix $\mathbf{B}$. Meanwhile, by rescaling the sample variance S_k^2 of the data $(X_1, \dots, X_k)$, we obtain an unbiased estimate for the fourth power of the Schatten 4-norm of $\mathbf{B}$.

Our discussion shows how we can obtain *a priori* guarantees and *a posteriori* error estimates for these norm computations; the results can be expressed in terms of the stable rank (2.2) of $\mathbf{B}$. We can also obtain norm estimators that are more computationally efficient using structured distributions for the test vectors, such as elements from an optimal measurement system.

5. Schatten p-norm estimation by sampling

As we saw in Section 4.8, we can easily construct unbiased estimators for the Schatten 2-norm and the Schatten 4-norm of a matrix by randomized sampling. What about the other Schatten norms? This type of computation can be used to obtain better approximations of the spectral norm, or it can be combined with the method of moments to approximate the spectral density of the input matrix (Kong and Valiant 2017).

In this section, we show that it is possible to use randomized sampling to construct unbiased estimators for the Schatten $2p$-norm for each natural number $p \in \mathbb{N}$. In contrast with the case $p \in \{1, 2\}$, estimators for $p \geq 3$ are combinatorial. They may also require a large number of samples to ensure that the variance of the estimator is controlled.

In the next section, we explain how to use iterative methods to approximate the spectral norm (i.e., the Schatten ∞-norm). Iterative algorithms also lead to much more reliable estimators for general Schatten norms.

5.1. Overview.

Consider a general matrix $\mathbf{B} \in \mathbb{F}^{m\times n}$, accessed via the matrix–vector product $\mathbf{u} \mapsto \mathbf{Bu}$. For a sample size k, let $\mathbf{\Omega} \in \mathbb{F}^{n\times k}$ be a (random) test matrix that does not depend on $\mathbf{B}$. For a natural number $p \geq 3$, we consider the problem of estimating the Schatten $2p$-norm $\|\mathbf{B}\|_{2p}$ from the sample matrix $\mathbf{Y} = \mathbf{B\Omega}$. The key idea is to cleverly process the sample matrix $\mathbf{Y}$ to form an unbiased estimator for the $2p$-th power of the norm. Since the methods in this section use linear information, they can be parallelized, and they are applicable in the one-pass and streaming environments.

Of course, if we are given a singular value decomposition (SVD) of $\mathbf{B}$, it is straightforward to extract the Schatten $2p$-norm. We are interested in methods that are much less expensive than the $O(\min\{mn^2, nm^2\})$ cost of computing an SVD with a classical direct algorithm. Randomized SVD or URV algorithms (Sections 11.2, 15, and 16) can also be used for Schatten norm estimation, but this approach is typically overkill.

5.2. Interlude: Lower bounds.

How large a sample size k is required to estimate $\|\mathbf{B}\|_{2p}$ up to a fixed constant factor with 75% probability over the randomness in $\mathbf{\Omega}$? The answer, unfortunately, turns out to be $k \gtrsim \min\{m, n\}^{1-2/p}$. In other words, for a general matrix, we cannot approximate the Schatten $2p$-norm for any $p > 2$ unless the sample size k grows polynomially with the dimension.

A more detailed version of this statement appears in Li, Nguyen and Woodruff (2014*a*, Thm. 3.2). The authors exhibit a particularly difficult type of input matrix (a standard normal matrix with a rank-one spike, as in Remark 2.1) to arrive at the negative conclusion.

If you have a more optimistic nature, you can also take the inspiration that other types of input matrices might be easier to handle. For example, it is possible to use a small random sample to compute the Schatten norm of a matrix that enjoys some decay in the singular value spectrum.

5.3. **Estimating Schatten norms the hard way.** First, let us describe a technique from classical statistics that leads to an unbiased estimator of $\|\mathbf{B}\|_{2p}^{2p}$. This estimator is both highly variable and computationally expensive, so we must proceed with caution.

For the rest of this section, we assume that the random test matrix $\mathbf{\Omega} \in \mathbb{F}^{n \times k}$ has isotropic columns $\boldsymbol{\omega}_i$ that are iid. Form the sample matrix $\mathbf{Y} = \mathbf{B}\mathbf{\Omega}$. Abbreviate $\mathbf{A} = \mathbf{B}^*\mathbf{B}$ and $\mathbf{X} = \mathbf{Y} * \mathbf{Y}$. Observe that

$$(\mathbf{X})_{ij} = (\mathbf{Y}^*\mathbf{Y})_{ij} = \boldsymbol{\omega}_i^* \mathbf{A} \boldsymbol{\omega}_j.$$

Therefore, for any natural numbers that satisfy $1 \le i_1, \dots, i_p \le k$,

$$(\mathbf{X})_{i_1 i_2} (\mathbf{X})_{i_2 i_3} \dots (\mathbf{X})_{i_p i_1} = \operatorname{trace}\left[\boldsymbol{\omega}_{i_1} \boldsymbol{\omega}_{i_1}^* \mathbf{A} \dots \boldsymbol{\omega}_{i_p} \boldsymbol{\omega}_{i_p}^* \mathbf{A}\right].$$

If we assume that $i_1, \dots, i_p$ are distinct, we can use independence and isotropy to compute the expectation:

$$\mathbb{E}[(\mathbf{X})_{i_1 i_2} (\mathbf{X})_{i_2 i_3} \dots (\mathbf{X})_{i_p i_1}] = \operatorname{trace}[\mathbf{A}^p] = \|\mathbf{B}\|_{2p}^{2p}$$

By averaging over all sequences of distinct indices, we obtain an unbiased estimator:

$$U_p = \frac{(k-p)!}{k!} \mathop{\sum\nolimits^{\circ}}_{1 \le i_1, \dots, i_p \le k} (\mathbf{X})_{i_1 i_2} (\mathbf{X})_{i_2 i_3} \dots (\mathbf{X})_{i_p i_1}.$$

The circle over the sum indicates that the indices must be distinct. Since $\mathbb{E}\, U_p = \|\mathbf{B}\|_{2p}^{2p}$, our hope is that

$$U_p^{1/(2p)} \approx \|\mathbf{B}\|_{2p}.$$

To ensure that the approximation is precise, the standard deviation of U_p should be somewhat smaller than the mean of U_p.

To understand the statistic U_p, we can use tools from the theory of U-statistics (Koroljuk and Borovskich 1994). For instance, when p is fixed, we have the limit

$$k \operatorname{Var}[U_p] \to p^2 \operatorname{Var}[\boldsymbol{\omega}^* \mathbf{A}^p \boldsymbol{\omega}] \quad \text{as } k \to \infty.$$

In particular, if the test vector $\boldsymbol{\omega} \sim \textsc{normal}(\mathbf{0}, \mathbf{I})$, then

$$k \operatorname{Var}[U_p] \to 2p^2 \|\mathbf{B}\|_{4p}^{4p} \quad \text{as } k \to \infty.$$

We can reduce the variance further by using test vectors from an optimal measurement system.

Unfortunately, it is quite expensive to compute the statistic U_p because it involves almost k^p summands. When p is a small constant (say, $p = 4$ or $p = 5$), it is not too onerous to form U_p. On the other hand, in the worst case, we cannot beat the lower bound $k \gtrsim \min\{m, n\}^{1-2/p}$, where computation of U_p requires $O(\min\{m, n\}^p)$ operations. Good (1977) proposes some small economies in this computation.

5.4. **Estimating Schatten norms the easy way.** Next, we describe a more recent approach that was proposed by Kong and Valiant (2017). This method suffers from even higher variance, but it is computationally efficient. As a consequence, we can target larger values of p and exploit a larger sample size k.

Superficially, the Kong & Valiant estimator appears similar to the statistic U_p. With the same notation, they restrict their attention to *increasing* sequences $i_1 < i_2 < \dots < i_p$ of indices. In other words,

$$V_p = \binom{k}{p}^{-1} \sum_{1 \le i_1 < \dots < i_p \le k} (\mathbf{X})_{i_1 i_2} (\mathbf{X})_{i_2 i_3} \dots (\mathbf{X})_{i_p i_1}.$$

Algorithm 3 *Schatten $2p$-norm estimation by random sampling.*
See Section 5.4.

Input: Input matrix $\mathbf{B} \in \mathbb{F}^{m \times n}$, order p of norm, number k of samples
Output: Schatten $2p$-norm estimate V_p

function SCHATTENESTIMATE($\mathbf{B}$, p, k)
  Draw test matrix $\mathbf{\Omega} \in \mathbb{F}^{n \times k}$ with iid isotropic columns
  Compute the sample matrix $\mathbf{Y} = \mathbf{B\Omega}$
  Form the Gram matrix $\mathbf{X} = \mathbf{Y}^*\mathbf{Y} \in \mathbb{F}^{k \times k}$
  Extract the strict upper triangle $\mathbf{T} = \mathcal{T}(\mathbf{X})$
  Compute $\mathbf{T}^{p-1}$ by repeated squaring
  Return $V_p = \operatorname{trace}(\mathbf{T}^{p-1}\mathbf{X})$

Much as before, V_p gives an unbiased estimator for $\|\mathbf{B}\|_{2p}^{2p}$.

Although V_p still appears to be combinatorial, the restriction to increasing sequences allows for a linear algebraic reformulation of the statistic. Let $\mathcal{T} : \mathbb{H}_k \to \mathbb{F}^{k \times k}$ be the linear map that reports the strict upper triangle of a (conjugate) symmetric matrix. Then

$$V_p = \binom{k}{p}^{-1} \operatorname{trace}[\mathcal{T}(\mathbf{X})^{p-1}\mathbf{X}].$$

The cost of computing V_p is usually dominated by the $O(k^2 n)$ arithmetic required to form $\mathbf{X}$ given $\mathbf{Y}$. See Algorithm 3 for the procedure.

Kong and Valiant (2017) obtain bounds for the variance of the estimator V_p to justify its employment when the number k of samples satisfies $k \gtrsim \min\{m, n\}^{1-2/p}$. This bound is probably substantially pessimistic for matrices that exhibit spectral decay, but these theoretical and computational questions remain open.

5.5. **Bootstrapping the sampling distribution.** Given the lack of prior guarantees for the estimators U_p and V_p described in this section, we recommend that users apply resampling methods to obtain empirical information about the sampling distribution. See Arcones and Gine (1992) for reliable bootstrap procedures for U-statistics.

5.6. **Extension: Estimating the spectral norm by random sampling.** Recall that the spectral norm of $\mathbf{B}$ is comparable with the Schatten $2p$-norm of $\mathbf{B}$ for an appropriate choice of p. Indeed,

$$\|\mathbf{B}\|_{2p} \geq \|\mathbf{B}\| \geq \min\{m, n\}^{-1/(2p)} \|\mathbf{B}\|_{2p} \quad \text{for } p \geq 1/2.$$

Thus, the Schatten $2p$-norm is equivalent to the spectral norm when $p \gtrsim \log \min\{m, n\}$. In fact, when the matrix $\mathbf{B}$ has a decaying singular value spectrum, the Schatten $2p$-norm may already be comparable with the spectral norm for much smaller values of p.

Thus, we can try to approximate the spectral norm by estimating the Schatten $2p$-norm for a sufficiently large value of p. Resampling techniques can help ensure that the estimate is reliable. Nevertheless, this method should be used with caution.

6. MAXIMUM EIGENVALUES AND TRACE FUNCTIONS

Our discussion of estimating the spectral norm from a random sample indicates that there is no straightforward way to construct an unbiased estimator for the maximum eigenvalue of a PSD matrix. Instead, we turn to Krylov methods, which repeatedly apply the matrix to appropriate vectors to extract more information.

The power method and the Lanczos method are two classic algorithms of this species. Historically, these algorithms have been initialized with a random vector to ensure that the starting vector has a component in the direction of a maximum eigenvector. Later, researchers recognized that the randomness has ancillary benefits. In particular, randomized algorithms can produce reliable estimates of the maximum eigenvalue, even when it is not separated from the rest of the spectrum (Dixon 1983, Kuczyński and Woźniakowski 1992).

Algorithm 4 *Randomized power method.*
See Section 6.2.

Input: Psd input matrix $\mathbf{A} \in \mathbb{H}_n$, maximum number q of iterations, stopping tolerance ε
Output: Estimate ξ of maximum eigenvalue of $\mathbf{A}$

```
function RandomizedPower(A, q, ε)
    ω = randn(n, 1)                                  ▷ Starting vector is random
    y_0 = ω/‖ω‖
    for i = 1, ..., q do
        y_i = A y_{i-1}
        ξ_{i-1} = y_{i-1}^* y_i
        y_i = y_i/‖y_i‖
        if |ξ_{i-1} − ξ_{i-2}| ≤ ε ξ_{i-1} then break   ▷ [opt] Stopping rule
    return ξ_{i-1}
```

In this section, we summarize theoretical results on the randomized power method and randomized Krylov methods for computing the maximum eigenvalue of a psd matrix. These results also have implications for estimating the minimum eigenvalue of a psd matrix and the spectral norm of a general matrix. Last, we explain how the Lanczos method leads to accurate estimates for trace functions.

6.1. **Overview.** Consider a psd matrix $\mathbf{A} \in \mathbb{H}_n$ with decreasingly ordered eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq 0$. We are interested in the problem of estimating the maximum eigenvalue λ_1. As in the last two sections, we assume access to $\mathbf{A}$ via the matrix–vector product $\mathbf{u} \mapsto \mathbf{A}\mathbf{u}$.

In contrast to the methods in Sections 4 and 5, the algorithms in this section require sequential applications of the matrix–vector product. In other words, we now demand nonlinear information about the input matrix $\mathbf{A}$. As a consequence, these algorithms resist parallelization, and they cannot be used in the one-pass or streaming environments.

The theoretical treatment in this section also covers the case of estimating the spectral norm $\|\mathbf{B}\|$ of a rectangular matrix $\mathbf{B} \in \mathbb{F}^{m \times n}$. Indeed, we can simply pass to the psd matrix $\mathbf{A} = \mathbf{B}^*\mathbf{B}$. From an applied point of view, however, it is important to develop separate algorithms that avoid squaring the matrix $\mathbf{B}$. For brevity, we omit a discussion about estimating spectral norms; see Golub and Van Loan (2013, Sec. 10.4).

6.2. **The randomized power method.** The randomized power method is a simple iterative algorithm for estimating the maximum eigenvalue.

6.2.1. *Procedure.* First, draw a random test vector $\boldsymbol{\omega} \in \mathbb{F}^n$. Since the maximum eigenvalue is unitarily invariant, it is most natural to draw the test vector $\boldsymbol{\omega}$ from a rotationally invariant distribution, such as $\boldsymbol{\omega} \sim \text{NORMAL}(\mathbf{0}, \mathbf{I})$. The power method iteratively constructs the sequence

$$\mathbf{y}_0 = \frac{\boldsymbol{\omega}}{\|\boldsymbol{\omega}\|} \quad \text{and} \quad \mathbf{y}_q = \frac{\mathbf{A}\mathbf{y}_{q-1}}{\|\mathbf{A}\mathbf{y}_{q-1}\|} \quad \text{for } q \geq 1.$$

At each step, we obtain an eigenvalue estimate

$$\xi_q = \mathbf{y}_q^* \mathbf{A} \mathbf{y}_q = \frac{\boldsymbol{\omega}^* \mathbf{A}^{2q+1} \boldsymbol{\omega}}{\boldsymbol{\omega}^* \mathbf{A}^{2q} \boldsymbol{\omega}} \quad \text{for } q \geq 0.$$

The randomized power method requires simulation of a single random test vector $\boldsymbol{\omega}$. To perform q iterations, it takes q sequential matrix–vector products with $\mathbf{A}$ and lower-order arithmetic. It operates with storage $O(n)$. See Algorithm 4 for pseudocode.

6.2.2. *Analysis.* The question is how many iterations q suffice to make ξ_q close to the maximum eigenvalue λ_1. More precisely, we aim to control the relative error e_q in the eigenvalue estimate ξ_q:

$$e_q = \frac{\lambda_1 - \xi_q}{\lambda_1}.$$

The error e_q is always nonnegative because of the Rayleigh theorem. Note that the computed vector $\mathbf{y}_q$ always has a substantial component in the invariant subspace associated with eigenvalues larger than ξ_q, but it may not be close to any maximum eigenvector, even when $\xi_q \approx \lambda_1$.

Kuczyński and Woźniakowski (1992) have established several remarkable results about the evolution of the error.

Theorem 6.1 (Randomized power method)**.** *Let $\mathbf{A} \in \mathbb{H}_n(\mathbb{R})$ be a real psd matrix. Draw a real test vector $\boldsymbol{\omega} \sim \text{NORMAL}(\mathbf{0}, \mathbf{I})$. After q iterations of the randomized power method, the error e_q in the maximum eigenvalue estimate ξ_q satisfies*

$$\mathbb{E}\, e_q \leq 0.871 \cdot \frac{\log n}{q-1} \quad \textit{for } q \geq 2. \tag{6.1}$$

Furthermore, if $\gamma = (\lambda_1 - \lambda_2)/\lambda_1$ is the relative spectral gap, then

$$\mathbb{E}\, e_q \leq 1.254 \cdot \sqrt{n}\, \gamma \mathrm{e}^{-q\gamma} \quad \textit{for } q \geq 1. \tag{6.2}$$

The second result (6.2) does not appear explicitly in Kuczyński and Woźniakowski (1992), but it follows from related ideas (Tropp 2018).

6.2.3. *Discussion.* Theorem 6.1 makes two different claims. First, the power method can exhibit a burn-in period of $q \approx \log n$ iterations before it produces a nontrivial estimate of the maximum eigenvalue; after this point, it always decreases the error in proportion to the number q of iterations. The second claim concerns the situation where the matrix has a spectral gap γ bounded away from zero. In the latter case, after the burn-in period, the error decreases at each iteration by a constant factor that depends on the spectral gap. The burn-in period of $q \approx \log n$ iterations is necessary for *any* algorithm that estimates the maximum eigenvalue of $\mathbf{A}$ from q matrix–vector products with the matrix (Simchowitz, Alaoui and Recht 2017).

Whereas classical analyses of the power method depend on the spectral gap γ, Theorem 6.1 comprehends that we can estimate the maximum eigenvalue even when $\gamma \approx 0$. On the other hand, it is generally not possible to obtain a reliable estimate of the maximum eigenvector in this extreme (Leyk and Woźniakowski 1998).

Theorem 6.1 can be improved in several respects. First, we can develop variants where the dimension n of the matrix $\mathbf{A}$ is replaced by its intrinsic dimension (2.1), or by smaller quantities that reflect spectral decay. Second, when the maximum eigenvalue has multiplicity greater than one, the power method estimates the maximum eigenvalue faster. Third, the result can be extended to the complex setting. See Tropp (2018) for further discussion.

Although the power method is often deprecated because it converges slowly, it is numerically stable, and it enjoys the (minimal) storage cost of $O(n)$.

6.3. **Randomized Krylov methods.** The power method uses the qth power of the matrix to estimate the maximum eigenvalue. A more sophisticated approach allows *any* polynomial with degree q. Algorithms based on this general technique are often referred to as *Krylov subspace methods*. The most famous instantiation is the *Lanczos method*, which is an efficient implementation of a Krylov subspace method for estimating the eigenvalues of a self-adjoint matrix.

6.3.1. *Abstract procedure.* Draw a random test vector $\boldsymbol{\omega} \in \mathbb{F}^n$. It is natural to use a rotationally invariant distribution, such as $\boldsymbol{\omega} \sim \text{NORMAL}(\mathbf{0}, \mathbf{I})$. For a depth parameter $q \in \mathbb{N}$, a randomized Krylov subspace method implicitly constructs the subspace

$$K_{q+1} := \operatorname{span}\{\boldsymbol{\omega}, \mathbf{A}\boldsymbol{\omega}, \dots, \mathbf{A}^q\boldsymbol{\omega}\}.$$

We can estimate the maximum eigenvalue of $\mathbf{A}$ as

$$\xi_q = \max_{\mathbf{u} \in K_{q+1}} \frac{\mathbf{u}^* \mathbf{A} \mathbf{u}}{\mathbf{u}^* \mathbf{u}} = \max_{\deg p \le q} \frac{\boldsymbol{\omega}^* \mathbf{A} p^2(\mathbf{A}) \boldsymbol{\omega}}{\|p(\mathbf{A})\boldsymbol{\omega}\|^2}.$$

The maximum occurs over polynomials p with coefficients in $\mathbb{F}$ and with degree at most q. The notation $p(\mathbf{A})$ refers to the spectral function induced by the polynomial p. We will discuss implementations of Krylov methods below in Section 6.3.4.

6.3.2. *Analysis.* Since the Krylov subspace is invariant to shifts in the spectrum of $\mathbf{A}$, it is more natural to compute the error relative to the spectral range of the matrix:

$$f_q = \frac{\lambda_1 - \xi_q}{\lambda_1 - \lambda_n}.$$

The error f_q is always nonnegative because it is a Rayleigh quotient.

Kuczyński and Woźniakowski (1992) have established striking results for the maximum eigenvalue estimate obtained via a randomized Krylov subspace method.

Theorem 6.2 (Randomized Krylov method)**.** *Let $\mathbf{A} \in \mathbb{H}_n(\mathbb{R})$ be a real psd matrix. Draw a real test vector $\boldsymbol{\omega} \sim \text{NORMAL}(\mathbf{0}, \mathbf{I})$. After q iterations of the randomized Krylov method, the error f_q in the maximum eigenvalue estimate ξ_q satisfies*

$$\mathbb{E}\, f_q \le 2.575 \cdot \left(\frac{\log n}{q-1} \right)^2 \quad \textit{for } q \ge 4. \tag{6.3}$$

Furthermore, if $\gamma = (\lambda_1 - \lambda_2)/\lambda_1$ is the relative spectral gap, then

$$\mathbb{E}\, f_q \le 2.589 \cdot \sqrt{n}\, \mathrm{e}^{-2(q-1)\sqrt{\gamma}} \quad \textit{for } q \ge 1. \tag{6.4}$$

The second result (6.4) is a direct consequence of a more detailed formula reported in Kuczyński and Woźniakowski (1992).

6.3.3. *Discussion.* Like the randomized power method, a randomized Krylov method can also exhibit a burn-in period of $q \approx \log n$ steps. Afterwards, the result (6.3) shows that the error decreases in proportion to $1/q^2$, which is much faster than the $1/q$ rate achieved by the power method. Furthermore, the result (6.4) shows that each iteration decreases the error by a constant factor $\mathrm{e}^{-2\sqrt{\gamma}}$ where γ is the spectral gap. In contrast, the power method only decreases the error by a constant factor $\mathrm{e}^{-\gamma}$.

Theorem 6.2 admits the same kind of refinements as Theorem 6.1. In particular, we can replace the dimension n with measures that reflect the spectral decay of the input matrix. See Tropp (2018) for details.

6.3.4. *Implementing Krylov methods.* What do we have to pay for the superior performance of the randomized Krylov method? If we only need an estimate of the maximum eigenvalue, without an associated eigenvector estimate, the cost is almost the same as for the randomized power method! On the other hand, if we desire the eigenvector estimate, it is common practice to store a basis for the Krylov subspace K_q. This is a classic example of a time–data tradeoff in computation.

We present pseudocode for the randomized Lanczos method, which is an efficient formulation of the Krylov method. Algorithm 5 is a direct implementation of the Lanczos recursion, but it exhibits complicated performance in floating-point arithmetic. Algorithm 5 includes the option to add full reorthogonalization; this step removes the numerical shortcomings at a substantial price in arithmetic (and storage).

If we use the Lanczos method without orthogonalization, then q iterations require q matrix–vector multiplies with $\mathbf{A}$ plus $O(qn)$ additional arithmetic. The orthogonalization step adds a total of $O(q^2 n)$ additional arithmetic. Computing the maximum eigenvalue (and eigenvector) of the tridiagonal matrix $\mathbf{T}$ can be performed with $O(q)$ arithmetic (Golub and Van Loan 2013, Sec. 8.4).

Algorithm 5 *Randomized Lanczos method (with full reorthogonalization).*
Use with caution! See Section 6.3.4.

Input: Psd input matrix $\mathbf{A} \in \mathbb{H}_n$, maximum number q of iterations
Output: Estimate $(\xi, \mathbf{y})$ for a maximum eigenpair of $\mathbf{A}$

function RANDOMIZEDLANCZOS($\mathbf{A}$, q)
  $q = \min(q, n-1)$
  $\mathbf{Q}(:,1) = \texttt{randn}(n,1)$ ▷ Starting vector $\boldsymbol{\omega}$ is random
  $\mathbf{Q}(:,1) = \mathbf{Q}(:,1)/\|\mathbf{Q}(:,1)\|$
  for $i = 1, \ldots, q$ **do**
    $\mathbf{Q}(:,i+1) = \mathbf{A}\mathbf{Q}(:,i)$
    $\alpha_i = \text{real}(\mathbf{Q}(:,i)^*\mathbf{Q}(:,i+1))$
    if $i = 1$ **then**
      $\mathbf{Q}(:,i+1) = \mathbf{Q}(:,i+1) - \alpha_i\mathbf{Q}(:,i)$
    else
      $\mathbf{Q}(:,i+1) = \mathbf{Q}(:,i+1) - \alpha_i\mathbf{Q}(:,i) - \beta_{i-1}\mathbf{Q}(:,i-1)$
    ▷ [opt] Reorthogonalize via double Gram–Schmidt
    $\mathbf{Q}(:,i+1) = \mathbf{Q}(:,i+1) - \mathbf{Q}(:,1:i)(\mathbf{Q}(:,1:i)^*\mathbf{Q}(:,i+1))$
    $\mathbf{Q}(:,i+1) = \mathbf{Q}(:,i+1) - \mathbf{Q}(:,1:i)(\mathbf{Q}(:,1:i)^*\mathbf{Q}(:,i+1))$
    $\beta_i = \|\mathbf{Q}(:,i+1)\|$
    if $\beta_i < \mu\sqrt{n}$ **then break** ▷ μ is machine precision
    $\mathbf{Q}(:,i+1) = \mathbf{Q}(:,i+1)/\beta_i$
  $\mathbf{T} = \texttt{tridiag}(\beta(1:i-1), \alpha(1:i), \beta(1:i-1))$
  $[\mathbf{V}, \mathbf{D}] = \texttt{eig}(\mathbf{T})$
  $[\xi, \texttt{ind}] = \min(\text{diag}(\mathbf{D}))$
  $\mathbf{y} = \mathbf{Q}(:,1:i)\,\mathbf{V}(:,\texttt{ind})$ ▷ [opt] Estimate max eigenvector

If we do not require the maximum eigenvector, the Lanczos method without orthogonalization operates with storage $O(n)$. If we need the maximum eigenvector or we add orthogonalization, the storage cost grows to $O(qn)$. It is possible to avoid the extra storage by recomputing the Lanczos vectors, but this approach requires great care. One of the main thrusts in the literature on Krylov methods is to reduce these storage costs while maintaining rapid convergence.

Warning 6.3 (Lanczos method)**.** *Algorithm 5 should be used with caution. For a proper discussion about designing Krylov methods, we recommend the books (Parlett 1998, Bai, Demmel, Dongarra, Ruhe and van der Vorst 1987, Golub and Van Loan 2013). There is also some recent theoretical work on the numerical stability of Lanczos methods (Musco, Musco and Sidford 2018, Carmon, Duchi, Sidford and Tian 2019).*

6.4. **The minimum eigenvalue.** The randomized power method and the randomized Krylov method can be used to estimate the minimum eigenvalue λ_n of the psd matrix $\mathbf{A} \in \mathbb{H}_n$.

The first approach is to apply the randomized power method to the shifted matrix $\nu\mathbf{I} - \mathbf{A}$, where the shift is chosen so that $\nu \geq \lambda_1$. In this case, the algorithm produces an approximation for $\nu - \lambda_n$. Note that the error in Theorem 6.1 is relative to $\nu - \lambda_n$, rather than λ_n.

The second approach begins with the computation of the Krylov subspace K_{q+1}. Instead of maximizing the Rayleigh quotient over the Krylov subspace, we minimize it:

$$\zeta_q = \min_{\mathbf{u} \in K_{q+1}} \frac{\mathbf{u}^*\mathbf{A}\mathbf{u}}{\mathbf{u}^*\mathbf{u}}.$$

This approach directly produces an estimate ζ_q for λ_n. The Krylov subspace is invariant under affine transformations of the spectrum of $\mathbf{A}$, so we can obtain an error bound for ζ_q by applying Theorem 6.2 formally to $\lambda_n \mathbf{I} - \mathbf{A}$.

Remark 6.4 (Inverses)**.** *If we can apply the matrix inverse $\mathbf{A}^{-1}$ to vectors, then we gain access to a wider class of algorithms for computing the minimum eigenvalue, including (shifted) inverse iteration and the Rayleigh quotient iteration. See (Parlett 1998) and (Golub and Van Loan 2013).*

6.5. **Block methods.** The basic power method and Krylov method can be extended by applying the iteration simultaneously to a larger number of (random) test vectors. The resulting algorithms are called *subspace iteration* and *block Krylov methods*, respectively. Historically, the reason for developing block methods was to resolve repeated or clustered eigenvalues.

In randomized linear algebra, we discover additional motivations for developing block methods. When the test vectors are drawn at random, block methods may converge slightly faster, and they succeed with much higher probability. On modern computer architectures, the cost of a block method may be comparable with the cost of a simple vector iteration, which makes this modification appealing. We will treat this class of algorithm more thoroughly in Section 11, so we postpone a full discussion. See also (Tropp 2018).

6.6. **Estimating trace functions.** Finally, we turn to the problem of estimating the trace of a spectral function of a psd matrix.

6.6.1. *Overview.* Consider a psd matrix $\mathbf{A} \in \mathbb{H}_n$ with eigenpairs $(\lambda_j, \mathbf{u}_j)$ for $j = 1, \dots, n$. Let $f : \mathbb{R}_+ \to \mathbb{R}$ be a function, and suppose that we wish to approximate

$$\operatorname{trace} f(\mathbf{A}) = \sum\nolimits_{j=1}^{n} f(\lambda_j).$$

We outline an incredible approach to this problem, called *stochastic Lanczos quadrature* (SLQ), that marries the randomized trace estimator (Section 4) to the Lanczos iteration (Algorithm 5).

This algorithm was devised by (Golub and Meurant 1994). Our presentation is based on Golub and Van Loan (2013, Sec. 10.2) and Ubaru, Chen and Saad (2017). Golub and Meurant (2010) provide a more complete treatment, and Musco et al. (2018) give a theoretical discussion about stability.

Related ideas can be used to estimate the trace of a spectral function of a rectangular matrix; that is, the sum of a function of each singular value of the matrix. For brevity, we omit all details on the rectangular case.

6.6.2. *Examples.* Computing the trace of a spectral function is a ubiquitous problem with a huge number of applications. Let us mention some of the key examples.

(1) For $f(\lambda) = \lambda^{-1}$, the resulting trace function is the trace of the matrix inverse. This computation arises in electronic structure calculations.

(2) For $f(t) = \log t$, the associated trace function is the log-determinant. This computation arises in Gaussian process regression.

(3) For $f(t) = t^p$ with $p \geq 1$, the trace function is the pth power of the Schatten p-norm. SLQ offers a more powerful alternative to the methods in Section 5.

We refer to Ubaru et al. (2017) for additional discussion.

6.6.3. *Procedure.* Let us summarize the mathematical ideas that lead to SLQ. As usual, we draw an isotropic random vector $\boldsymbol{\omega} \in \mathbb{F}^n$. Then the random variable

$$X = \boldsymbol{\omega}^* f(\mathbf{A}) \boldsymbol{\omega} \quad \text{satisfies} \quad \mathbb{E}\, X = \operatorname{trace} f(\mathbf{A}).$$

Using the spectral resolution of $\mathbf{A}$, we can rewrite X in the form

$$X = \sum\nolimits_{j=1}^{n} f(\lambda_j) \, |\mathbf{u}_j^* \boldsymbol{\omega}|^2 = \int_{\mathbb{R}_+} f(\lambda) \, \nu(\mathrm{d}\lambda)$$

Algorithm 6 *Stochastic Lanczos quadrature.*
See Section 6.6.

Input: Psd input matrix $\mathbf{A} \in \mathbb{H}_n$, function f, number k of samples, number q of Lanczos iterations
Output: Estimate $\bar{Z}_k$ for $\operatorname{trace} f(\mathbf{A})$.

function StochasticLanczosQuadrature($\mathbf{A}$, f, k, q)
  for $i = 1, \dots, k$ **do** ▷ Extract k independent samples
    Draw a random isotropic vector $\boldsymbol{\omega}_i \in \mathbb{F}^n$
    Form $\mathbf{T} = \text{RandomizedLanczos}(\mathbf{A}, \boldsymbol{\omega}_i, q)$
      ▷ Apply q steps of Lanczos with starting vector $\boldsymbol{\omega}_i$
    $[\mathbf{V}, \boldsymbol{\Theta}] = \texttt{eig}(\mathbf{T})$ ▷ Tridiagonal eigenproblem
    Extract nodes $\boldsymbol{\Theta} = \operatorname{diag}(\theta_1, \dots, \theta_{q+1})$
    Extract weights $\boldsymbol{\delta}_1^* \mathbf{V} = (\tau_1, \dots, \tau_{q+1})$
    Form the approximation $Z_i = \sum_{\ell=1}^{q+1} \tau_\ell^2 f(\theta_\ell)$
  Return $\bar{Z}_k = k^{-1} \sum_{i=1}^{k} Z_i$

for an appropriate measure ν on $\mathbb{R}_+$ that depends on $\mathbf{A}$ and $\boldsymbol{\omega}$. Although we cannot generally compute the integral directly, we can approximate it by using a numerical quadrature rule:

$$X \approx \sum\nolimits_{\ell=1}^{q+1} \tau_\ell^2 f(\theta_\ell) =: Z.$$

What is truly amazing is that the weights τ_ℓ^2 and the nodes θ_ℓ for the quadrature rule can be extracted from the tridiagonal matrix $\mathbf{T} \in \mathbb{H}_{q+1}$ produced by q iterations of the Lanczos iteration with starting vector $\boldsymbol{\omega}$. This point is not obvious, but a full explanation exceeds our scope.

SLQ approximates the trace function by averaging independent copies of the simple approximation:

$$\operatorname{trace} f(\mathbf{A}) \approx \frac{1}{k} \sum\nolimits_{i=1}^{k} Z_i \quad \text{where} \quad Z_i \sim Z \text{ are iid.}$$

The analysis of the SLQ approximation requires heavy machinery from approximation theory. See Golub and Meurant (2010) and Ubaru et al. (2017) for more details.

Algorithm 6 contains pseudocode for SLQ. The dominant cost is $O(kq)$ matrix–vector multiplies with $\mathbf{A}$, plus $O(kq^2)$ additional arithmetic. We recommend using structured random test vectors to reduce the variance of the resulting approximation. The storage cost is $O(qn)$ numbers.

7. Matrix approximation by sampling

In Section 4, we have seen that it is easy to form an unbiased estimator for the trace of a matrix. By averaging multiple copies of the simple estimator, we can improve the quality of the estimate. The same idea applies in the context of matrix approximation. In this setting, the goal is to produce a matrix approximation that has more "structure" than the target matrix. The basic approach is to find a structured unbiased estimator for the matrix and to average multiple copies of the simple estimator to improve the approximation quality.

This section outlines two instances of this methodology. First, we develop a toy algorithm for approximate multiplication of matrices. Second, we show how to approximate a dense graph Laplacian by a sparse graph Laplacian; this construction plays a role in the Sparse-Cholesky algorithm presented in Section 18. Section 19 contains another example, the method of random features in kernel learning.

The material in this section is summarized from the treatments of matrix concentration in (Tropp 2015, Tropp 2019).

7.1. **Empirical approximation.** We begin with a high-level discussion of the method of empirical approximation of a matrix. The examples in this section are all instances of this general idea.

Let $\mathbf{B} \in \mathbb{F}^{m \times n}$ be a target matrix that we wish to approximate by a more "structured" matrix. Suppose that we can express the matrix $\mathbf{B}$ as a sum of "simple" matrices:

$$\mathbf{B} = \sum\nolimits_{i=1}^{I} \mathbf{B}_i. \tag{7.1}$$

In the cases we will study, the summands $\mathbf{B}_i$ will be sparse or low-rank.

Next, consider a probability distribution $\{p_i : i = 1, \dots, I\}$ over the indices in the sum (7.1). For now, we treat this distribution as given. Construct a random matrix $\mathbf{X} \in \mathbb{F}^{m \times n}$ that takes values

$$\mathbf{X} = p_i^{-1} \mathbf{B}_i \quad \text{with probability } p_i \text{ for each } i = 1, \dots, I.$$

(Enforce the convention that $0/0 = 0$.) It is clear that $\mathbf{X}$ is an unbiased estimator for $\mathbf{B}$:

$$\mathbb{E}\, \mathbf{X} = \sum\nolimits_{i=1}^{I} (p_i^{-1} \mathbf{B}_i) p_i = \mathbf{B}.$$

The random matrix $\mathbf{X}$ enjoys the same kind of structure as the summands $\mathbf{B}_i$. On the other hand, a single draw of the matrix $\mathbf{X}$ is rarely a good approximation of the matrix $\mathbf{B}$.

To obtain a better estimator for $\mathbf{B}$, we average independent copies of the initial estimator:

$$\bar{\mathbf{X}}_k = \frac{1}{k} \sum\nolimits_{i=1}^{k} \mathbf{X}_i \quad \text{where } \mathbf{X}_i \sim \mathbf{X} \text{ are iid.}$$

By linearity of expectation, $\bar{\mathbf{X}}_k$ is also unbiased:

$$\mathbb{E}\, \bar{\mathbf{X}}_k = \mathbf{B}.$$

If the parameter k remains small, then $\bar{\mathbf{X}}_k$ inherits some of the structure from the $\mathbf{B}_i$. The question is how many samples k we need to ensure that $\bar{\mathbf{X}}_k$ also approximates $\mathbf{B}$ well.

Remark 7.1 (History)**.** *Empirical approximation was first developed by Maurey in unpublished work from the late 1970s on the geometry of Banach spaces. The idea was first broadcast by Carl (1985), in a paper on approximation theory. Applications to randomized linear algebra were proposed by Frieze et al. (2004) and by (Drineas et al. 2006*a*, Drineas et al. 2006*b*, Drineas et al. 2006*c*). More refined analyses of empirical matrix approximation were obtained in (Rudelson and Vershynin 2007, Tropp 2015). Many other papers consider specific applications of the same methodology.*

7.2. **The matrix Bernstein inequality.** The main tool for analyzing the sample average estimator $\bar{\mathbf{X}}_k$ from the last section is a variant of the matrix Bernstein inequality. The following result is drawn from Tropp (2015).

Theorem 7.2 (Matrix Monte Carlo)**.** *Let $\mathbf{B} \in \mathbb{F}^{m \times n}$ be a fixed matrix. Construct a random matrix $\mathbf{X} \in \mathbb{F}^{m \times n}$ that satisfies*

$$\mathbb{E}\, \mathbf{X} = \mathbf{B} \quad \textit{and} \quad \|\mathbf{X}\| \le R.$$

Define the per-sample second moment:

$$v(\mathbf{X}) := \max\{\| \mathbb{E}[\mathbf{X}\mathbf{X}^*]\|, \| \mathbb{E}[\mathbf{X}^*\mathbf{X}]\|\}.$$

Form the matrix sampling estimator

$$\bar{\mathbf{X}}_k = \frac{1}{k} \sum\nolimits_{i=1}^{k} \mathbf{X}_i \quad \textit{where } \mathbf{X}_i \sim \mathbf{X} \textit{ are iid.}$$

Then

$$\mathbb{E}\, \|\bar{\mathbf{X}}_k - \mathbf{B}\| \le \sqrt{\frac{2v(\mathbf{X}) \log(m+n)}{k}} + \frac{2R \log(m+n)}{3k}.$$

Furthermore, for all $t \ge 0$,

$$\mathbb{P}\left\{ \|\bar{\mathbf{X}}_k - \mathbf{B}\| \ge t \right\} \le (m+n) \exp\left(\frac{-kt^2/2}{v(\mathbf{X}) + 2Rt/3} \right).$$

To explain the meaning of this result, let us determine how large k should be to ensure that the expected approximation error lies below a positive threshold ε. The bound

$$k \geq \max\left\{ \frac{2v(\mathbf{X})\log(m+n)}{\varepsilon^2},\ \frac{2R\log(m+n)}{3\varepsilon} \right\}$$

implies that $\mathbb{E}\,\|\bar{\mathbf{X}}_k - \mathbf{B}\| \leq \varepsilon + \varepsilon^2$. In other words, the number k of samples should be proportional to the larger of the second moment $v(\mathbf{X})$ and the upper bound R.

This fact points toward a disappointing feature of empirical matrix approximation: To make ε small, the number k of samples must increase with ε^{-2} and often with $\log(m+n)$ as well. This phenomenon is an unavoidable consequence of the central limit theorem and the geometry induced by the spectral norm. It means that matrix sampling estimators are not suitable for achieving high-precision approximations. See (Tropp 2015, Sec. 6.2.3) for further discussion. The logarithmic terms are also necessary in the worst case.

Remark 7.3 (History)**.** *The matrix Bernstein inequality has a long history, outlined in Tropp (2015). The earliest related results concern uniform smoothness estimates (Tomczak-Jaegermann 1974) and Khintchine inequalities (Lust-Piquard 1986) for the Schatten classes. A first application to statistics appears in Rudelson (1999). Modern approaches are based on the matrix Laplace transform method (Ahlswede and Winter 2002, Oliveira 2009*a*, Tropp 2012*b*) or on the method of exchangeable pairs (Mackey, Jordan, Chen, Farrell and Tropp 2014, Tropp 2016).*

7.3. **Approximate matrix multiplication.** A first application of empirical matrix approximation is to approximate the product of two matrices: $\mathbf{M} = \mathbf{BC}$ where $\mathbf{B} \in \mathbb{F}^{m\times I}$ and $\mathbf{C} \in \mathbb{F}^{I\times n}$. Computing the product by direct matrix–matrix multiplication requires $O(mnI)$ arithmetic operations. When the inner dimension I is very large as compared with m and n, we might try to reduce the cost by sampling.

In our own work, we have not encountered situations where approximate matrix multiplication is practical because the quality of the approximation is very low. Nevertheless, the theory serves a dual purpose as the foundation for subspace embedding by discrete sampling (Section 9.6).

7.3.1. *Matrix multiplication by sampling.* To simplify the analysis, let us pre-scale the matrices $\mathbf{B}$ and $\mathbf{C}$ so that each one has spectral norm equal to one:

$$\|\mathbf{B}\| = \|\mathbf{C}\| = 1.$$

This step can be performed efficiently using the spectral norm estimators outlined in Section 6. This normalization will remain in force for the rest of Section 7.3.

Observe that the matrix–matrix product satisfies

$$\mathbf{BC} = \sum\nolimits_{i=1}^{I} (\mathbf{B})_{:i}(\mathbf{C})_{i:},$$

where $(\mathbf{B})_{:i}$ is the ith column of $\mathbf{B}$ and $(\mathbf{C})_{i:}$ is the ith row of $\mathbf{C}$. This expression motivates us to approximate the product by sampling terms at random from the sum.

Let $\{p_i : i = 1, \dots, I\}$ be a sampling distribution, to be specified later. Form a random rank-one unbiased estimator for the product:

$$\mathbf{X} = p_i^{-1} \cdot (\mathbf{B})_{:i}(\mathbf{C})_{i:} \quad \text{with probability } p_i \text{ for each } i = 1, \dots, I.$$

By construction, $\mathbb{E}\,\mathbf{X} = \mathbf{BC}$. We can average k independent copies of $\mathbf{X}$ to obtain a better approximation $\bar{\mathbf{X}}_k$ to the matrix product. The cost of computing the estimator $\bar{\mathbf{X}}_k$ of the matrix product explicitly is only $O(mnk)$ operations, so it is more efficient than the full multiplication when $k \ll I$.

Theorem 7.2 yields an easy analysis of this approach. The conclusion depends heavily on the choice of sampling distribution. Regardless, we can never expect to attain a high-accuracy approximation of the product by sampling because the number k of samples must scale proportionally with the inverse square ε^{-2} of the accuracy parameter ε.

7.3.2. *Uniform sampling.* The easiest way to approximate matrix multiplication is to choose uniform sampling probabilities: $p_i = 1/I$ for each $i = 1, \dots, I$. To analyze this case, we introduce the *coherence* parameter:

$$\mu(\mathbf{B}) := I \cdot \max_{i=1,\dots,I} \|(\mathbf{B})_{:i}\|^2.$$

Up to scaling, this is the maximum squared norm of a column of $\mathbf{B}$. Since $\mathbf{B} \in \mathbb{F}^{m \times I}$ and $\|\mathbf{B}\| = 1$, the coherence lies in the range $[m, I]$. The difficulty of approximating matrix multiplication by uniform sampling increases with the coherence of $\mathbf{B}$ and $\mathbf{C}^*$.

To invoke Theorem 7.2, observe that the per-sample second moment and the spectral norm of the estimator $\mathbf{X}$ satisfy the bound

$$\max\{v(\mathbf{X}), \|\mathbf{X}\|\} \le \max\{\mu(\mathbf{B}), \mu(\mathbf{C}^*)\}.$$

Let $\varepsilon \in (0, 1]$ be an accuracy parameter. If

$$k \ge 2\varepsilon^{-2} \max\{\mu(\mathbf{B}), \mu(\mathbf{C}^*)\} \log(m + n),$$

then

$$\mathbb{E} \|\bar{\mathbf{X}}_k - \mathbf{BC}\| \le 2\varepsilon.$$

In other words, the number k of rank-one factors we need to obtain a relative approximation of the matrix product is proportional to the maximum coherence of $\mathbf{B}$ and $\mathbf{C}^*$. In the best scenario, the number of samples is proportional to $\max\{m, n\} \log(m + n)$; in the worst case, the sample complexity can be as large as I.

7.3.3. *Importance sampling.* If the norms of the columns of the factors $\mathbf{B}$ and $\mathbf{C}^*$ vary wildly, we may need to use importance sampling to obtain a nontrivial approximation bound. Define the sampling probabilities

$$p_i = \frac{\|(\mathbf{B})_{:i}\|^2 + \|(\mathbf{C})_{:i}\|^2}{\|\mathbf{B}\|_{\mathrm{F}}^2 + \|\mathbf{C}\|_{\mathrm{F}}^2} \quad \text{for } i = 1, \dots, I.$$

These probabilities are designed to balance terms arising from Theorem 7.2. In most cases, we compute the sampling distribution by directly evaluating the formula, at the cost of $O((m + n)I)$ operations.

With the importance sampling distribution, the per-sample second moment and the spectral norm of the estimator $\mathbf{X}$ satisfy

$$\max\{v(\mathbf{X}), \|\mathbf{X}\|\} \le \frac{1}{2}(\operatorname{srank}(\mathbf{B}) + \operatorname{srank}(\mathbf{C})).$$

The stable rank is defined in (2.2). Let $\varepsilon \in (0, 1]$ be an accuracy parameter. If

$$k \ge \varepsilon^{-2} \left(\operatorname{srank}(\mathbf{B}) + \operatorname{srank}(\mathbf{C})\right) \log(m + n),$$

then

$$\mathbb{E} \|\bar{\mathbf{X}}_k - \mathbf{BC}\| \le 2\varepsilon.$$

In other words, the number of rank-one factors we need to approximate the matrix product by importance sampling is $\log(m + n)$ times the total stable rank of the matrices.

The sample complexity bound for importance sampling always improves over the bound for uniform sampling. Indeed, $\operatorname{srank}(\mathbf{B}) \le \mu(\mathbf{B})$ under the normalization $\|\mathbf{B}\| = 1$. There are also many situations where the stable rank is smaller than either dimension of the matrix. These are the cases where one might consider using approximate matrix multiplication by importance sampling.

7.3.4. *History.* Randomized matrix multiplication was proposed in Cohen and Lewis (1999). It is implicit in Frieze et al. (2004), while Drineas et al. (2006*a*) give a more explicit treatment. The analysis here has its origins in Rudelson and Vershynin (2007), and the detailed presentation is adapted from Zouzias (2013). Another interesting approach to randomized matrix multiplication appears in (Pagh 2013). See Tropp (2015, Chap. 6 Notes) for further references.

7.4. **Approximating a graph by a sparse graph.** As a second application of empirical matrix approximation, we will show how to take a dense graph and find a sparse graph that serves as a proxy. This procedure operates by replacing the Laplacian matrix of the graph with a sparse Laplacian matrix. Beyond its intrinsic interest as a fact about graphs, this technique plays a central role in randomized solvers for Laplacian linear systems (Section 18).

7.4.1. *Graphs and Laplacians.* We will consider weighted, loop-free, undirected graphs on the vertex set $V = \{1, \dots, n\}$. We can specify a weighted graph G on V by means of a nonnegative weight function $w : V \times V \to \mathbb{R}_+$. The graph is *loop-free* when $w_{ii} = 0$ for each vertex $i \in V$. The graph is *undirected* if and only if $w_{ij} = w_{ji}$ for each pair (i, j). The *sparsity* of an undirected graph is the number of strictly positive weights w_{ij} with $i \leq j$.

Alternatively, we can work with graph Laplacians. The *elementary Laplacian* $\mathbf{\Delta}_{ij}$ on a vertex pair $(i, j) \in V \times V$ is the psd matrix

$$\mathbf{\Delta}_{ij} = (\boldsymbol{\delta}_i - \boldsymbol{\delta}_j)(\boldsymbol{\delta}_i - \boldsymbol{\delta}_j)^* \in \mathbb{H}_n(\mathbb{R}).$$

Let w be the weight function of a loop-free, undirected graph G. The Laplacian associated with the graph G is the matrix

$$\mathbf{L}_G = \sum\nolimits_{1 \leq i < j \leq n} w_{ij} \mathbf{\Delta}_{ij} \in \mathbb{H}_n(\mathbb{R}). \tag{7.2}$$

The Laplacian $\mathbf{L}_G$ is psd because it is a nonnegative linear combination of psd matrices.

The Laplacian of a graph is analogous to the Laplacian differential operator. You can think about $\mathbf{L}_G$ as an analog of the heat kernel that models the diffusion of a particle on the graph. The Poisson problem $\mathbf{L}_G \mathbf{x} = \mathbf{f}$ serves as a primitive for answering a wide range of questions involving undirected graphs (Teng 2010). Applications include clustering and partitioning data, studying random walks on graphs, and solving finite-element discretizations of elliptic PDEs.

7.4.2. *Spectral approximation.* Fix a parameter $\varepsilon \in (0, 1)$. We say that a graph H is an ε-spectral approximation of a graph G if their Laplacians are comparable in the psd order:

$$(1 - \varepsilon)\, \mathbf{L}_G \preccurlyeq \mathbf{L}_H \preccurlyeq (1 + \varepsilon)\, \mathbf{L}_G. \tag{7.3}$$

The relation (7.3) ensures that the graph H and the graph G are close cousins.

In particular, under (7.3), the matrix $\mathbf{L}_H$ serves as an excellent preconditioner for the Laplacian $\mathbf{L}_G$. In other words, if we can easily solve (consistent) linear systems of the form $\mathbf{L}_H \mathbf{y} = \mathbf{b}$, then we can just as easily solve the Poisson problem $\mathbf{L}_G \mathbf{x} = \mathbf{f}$.

For an arbitrary input graph G, we will demonstrate that there is a *sparse* graph H that is a good spectral approximation of G. For several reasons, this construction does not immediately lead to effective methods for designing preconditioners. Nevertheless, related ideas have resulted in practical, fast solvers for Poisson problems on undirected graphs. We will make this connection in Section 18.

7.4.3. *The normalizing map.* It is convenient to present a few more concepts from spectral graph theory. Let us introduce the *normalizing map*

$$\mathbf{K}_G(\mathbf{A}) := (\mathbf{L}_G^\dagger)^{1/2} \mathbf{A} (\mathbf{L}_G^\dagger)^{1/2} \quad \text{for } \mathbf{A} \in \mathbb{H}_n(\mathbb{R}).$$

As usual, $(\cdot)^\dagger$ is the pseudoinverse and $(\cdot)^{1/2}$ is the psd square root of a psd matrix. We use the normalizing map to compare graph Laplacians. Indeed,

$$\|\mathbf{K}_G(\mathbf{L}_H - \mathbf{L}_G)\| \leq \varepsilon \tag{7.4}$$

implies that the graph H is an ε-spectral approximation of the graph G. This claim follows easily from (7.3).

7.4.4. *Effective resistance.* Next, define the *effective resistance* ϱ_{ij} of a vertex pair (i, j) in the graph G as

$$\varrho_{ij} := \operatorname{trace}[\mathbf{K}_G(\mathbf{\Delta}_{ij})] \geq 0 \quad \text{for each } 1 \leq i < j \leq n. \tag{7.5}$$

We can compute the family of effective resistances in time $O(n^3)$ by means of a Cholesky factorization of $\mathbf{L}_G$, although faster algorithms are now available (Kyng 2017).

To understand the terminology, let us regard the graph G as an electrical network where w_{ij} is the conductivity of the wire connecting the vertex pair (i, j). The effective resistance ϱ_{ij} is the resistance of the entire electrical network G against passing a unit of current from vertex i to vertex j.

7.4.5. *Sparsification by sampling.* We are now prepared to construct a sparse approximation of a loop-free, undirected graph G specified by the weight function w. The representation (7.2) of the graph Laplacian immediately suggests that we can apply the empirical approximation paradigm. To do so, we must design an appropriate sampling distribution.

Define the sampling probabilities

$$p_{ij} = \frac{w_{ij}\, \varrho_{ij}}{\operatorname{rank}(\mathbf{L}_G)} \quad \text{for each } 1 \leq i < j \leq n.$$

It is clear that $p_{ij} \geq 0$. With some basic matrix algebra, we can also confirm that $\sum_{i<j} p_{ij} = 1$.

Following our standard approach, we construct the random matrix

$$\mathbf{X} = \frac{w_{ij}}{p_{ij}}\, \mathbf{\Delta}_{ij} \quad \text{with probability } p_{ij} \text{ for each } i < j.$$

Next, we average k independent copies of the estimator:

$$\bar{\mathbf{X}}_k = \frac{1}{k} \sum\nolimits_{i=1}^{k} \mathbf{X}_i \quad \text{where } \mathbf{X}_i \sim \mathbf{X} \text{ are iid.}$$

By construction, the estimator is unbiased: $\mathbb{E}\, \bar{\mathbf{X}}_k = \mathbf{L}_G$. Moreover, $\bar{\mathbf{X}}_k$ is itself the graph Laplacian of a (random) graph H with sparsity at most k. We just need to determine the number k of samples that are sufficient to make H a good spectral approximation of G.

Given the effective resistances, computation of the sampling probabilities requires $O(s)$ time, where s is the sparsity of the graph. The cost of sampling k copies of $\mathbf{X}$ is $\tilde{O}(s + k)$. It is natural to represent $\bar{\mathbf{X}}_k$ using a sparse matrix data structure, with storage cost $O(k \log n)$.

7.4.6. *Analysis.* The analysis involves a small twist. In view of (7.4), we need to demonstrate that $\mathbf{K}_G(\mathbf{L}_H) \approx \mathbf{K}_G(\mathbf{L}_G)$. Therefore, instead of considering the random matrix $\bar{\mathbf{X}}_k$, we pass to the random matrix $\mathbf{K}_G(\bar{\mathbf{X}}_k)$, which is an unbiased estimator for $\mathbf{K}_G(\mathbf{L}_G)$.

Note that the random matrix $\mathbf{K}_G(\mathbf{X})$ satisfies

$$\max\{v(\mathbf{K}_G(\mathbf{X})), \|\mathbf{K}_G(\mathbf{X})\|_2\} \leq \operatorname{rank}(\mathbf{L}_G) < n.$$

Suppose that we choose

$$k \geq 3\varepsilon^{-2} n \log(2n) \quad \text{where } \varepsilon \in (0, 1).$$

Then Theorem 7.2 implies

$$\mathbb{E}\, \|\mathbf{K}_G(\bar{\mathbf{X}}_k - \mathbf{L}_G)\|_2 \leq \varepsilon.$$

We conclude that the random graph H with Laplacian $\mathbf{L}_H = \bar{\mathbf{X}}_k$ has at most k nonzero weights, and it is an ε-spectral approximation to the graph G.

In other words, every graph on n vertices has a $(1/2)$-spectral approximation with at most $12n \log n$ nonzero weights. Modulo the precise constant, this is the tightest result that can be obtained if we form the graph H via random sampling.

7.4.7. *History.* The idea of approximating a matrix in the spectral norm by means of random sampling of entries was proposed by Achlioptas and McSherry (2001) and Achlioptas and McSherry (2007). This work initiated a line of literature on matrix sparsification in the randomized NLA community; see Tropp (2015) for more references. Let us emphasize that these general approaches achieve much weaker approximation guarantees than (7.3).

The idea of sparsifying a graph by randomly sampling edges in proportion to the effective resistances was developed by Spielman and Srivastava (2011); the analysis above is drawn from Tropp (2019). A deterministic method for graph sparsification, with superior guarantees, appears in Batson, Spielman and Srivastava (2014).

8. Randomized embeddings

One of the core tools in randomized linear algebra is *randomized linear embedding*, often assigned the misnomer *random projection*. The application of randomized embeddings is often referred to as *sketching*.

This section begins with a formal definition of a randomized embedding. Then we introduce the Gaussian embedding, which is the simplest construction, and we summarize its analysis. Randomized embeddings have a wide range of applications in randomized linear algebra. Some implications of this theory include the Johnson–Lindenstrauss lemma and a simple construction of a subspace embedding. We also explain why results for Gaussians transfer to a wider setting. Last, we give a short description of random partial isometries, a close cousin of Gaussian embeddings.

8.1. **What is a random embedding?** Let $E \subseteq \mathbb{F}^n$ be a set, and let $\varepsilon \in (0, 1)$ be a distortion parameter. We say that a linear map $\mathsf{S} : \mathbb{F}^n \to \mathbb{F}^d$ is an (ℓ_2) *embedding* of E with distortion ε when

$$(1 - \varepsilon) \|\mathsf{x}\| \leq \|\mathsf{S}\mathsf{x}\| \leq (1 + \varepsilon) \|\mathsf{x}\| \quad \text{for all } \mathsf{x} \in E. \tag{8.1}$$

It is sometimes convenient to abbreviate this kind of two-sided inequality as $\|\mathsf{S}\mathsf{x}\| = (1 \pm \varepsilon)\|\mathsf{x}\|$.

We usually think about the case where $d \ll n$, so the map S enacts a dimension reduction. In other words, S transfers data from the high-dimensional space $\mathbb{F}^n$ to the low-dimensional space $\mathbb{F}^d$. As we will discuss, the low-dimensional representation of the data can be used to obtain fast, approximate solutions to computational problems.

The relation (8.1) expresses the idea that the embedding S should preserve the geometry of the set E. Unfortunately, we do not always know the set E in advance. Moreover, we would like the map S to be easy to construct, and it should be computationally efficient to apply S to the data. These goals may be in tension.

We can resolve this dilemma by drawing the embedding S from a probability distribution. Many types of probability distributions serve. In particular, we can use highly structured random matrices (Section 9) that are easy to build, to store, and to apply to vectors. Section 10 presents a case study about how random embeddings can be applied to solve overdetermined least-squares problems.

8.2. **Restricted singular values.** Our initial goal is to understand something about the theoretical behavior of randomized embeddings. To that end, let us introduce quantities that measure how much an embedding distorts a set. Let $\mathsf{S} \in \mathbb{F}^{d \times n}$ be a linear map, and let $E \subseteq \mathbb{S}^{n-1}(\mathbb{F})$ be an arbitrary subset of the unit sphere in $\mathbb{F}^n$. The *minimum* and *maximum restricted singular value* are, respectively, defined as

$$\sigma_{\min}(\mathsf{S}; E) := \min_{\mathsf{x} \in E} \|\mathsf{S}\mathsf{x}\| \quad \text{and} \quad \sigma_{\max}(\mathsf{S}; E) := \max_{\mathsf{x} \in E} \|\mathsf{S}\mathsf{x}\|. \tag{8.2}$$

If E composes the entire unit sphere, then these quantities coincide with the ordinary minimum and maximum singular value of S. More generally, the restricted singular values describe how much the linear map S can contract or expand a point in E.

Remark 8.1 (General sets)**.** *In this treatment, we require E to be a subset of the unit sphere. Related, but more involved, results hold when E is a general set. See Thrampoulidis et al. (2014) and Oymak and Tropp (2018) for more results and applications.*

8.3. **Gaussian embeddings.** Our theoretical treatment of random embeddings focuses on the most highly structured case. A Gaussian embedding is a random matrix of the form

$$\mathbf{\Gamma} \in \mathbb{F}^{d \times n} \quad \text{with iid entries } (\mathbf{\Gamma})_{ij} \sim \text{NORMAL}(0, d^{-1}).$$

The cost of explicitly storing a Gaussian embedding is $O(dn)$, and the cost of applying it to a vector is $O(dn)$.

The scaling of the matrix ensures that

$$\mathbb{E} \, \|\mathbf{\Gamma x}\|^2 = \|\mathbf{x}\|^2 \quad \text{for each } \mathbf{x} \in \mathbb{F}^n.$$

We wish to understand how large to choose the embedding dimension d so that the map $\mathbf{\Gamma}$ approximately preserves the norms of all points in a given set E. We can do so by obtaining bounds for the restricted singular values. For a Gaussian embedding, we will see that $\sigma_{\min}(\mathbf{\Gamma}; E)$ and $\sigma_{\max}(\mathbf{\Gamma}; E)$ are controlled by the geometry of the set E.

Remark 8.2 (Why Gaussians?)**.** *Gaussian embeddings admit a simple and beautiful analysis. In our computational experience, many other embeddings exhibit the same (universal) behavior as a Gaussian map. In spite of that, the rigorous analysis of other types of embeddings tends to be difficult, even while it yields rather imprecise results. The confluence of these facts motivates us to argue that the Gaussian analysis provides enough insight for many practical purposes.*

8.4. **The Gaussian width.** For the remainder of Section 8, we will work in the real field ($\mathbb{F} = \mathbb{R}$). Given a set $E \subseteq \mathbb{S}^{n-1}(\mathbb{R})$, define the *Gaussian width* $w(E)$ via

$$w(E) := \mathbb{E} \sup_{\mathbf{x} \in E} \langle \mathbf{g}, \mathbf{x} \rangle \quad \text{where } \mathbf{g} \in \mathbb{R}^n \text{ is standard normal.}$$

The Gaussian width is a measure of the content of the set E. It plays a fundamental role in the performance of randomized embeddings.

Here are some basic properties of the Gaussian width.

- The width is invariant under rotations: $w(\mathbf{Q}E) = w(E)$ for each orthogonal matrix $\mathbf{Q}$.
- The width is increasing with respect to set inclusion: $E \subseteq F$ implies that $w(E) \leq w(F)$.
- The width lies in the range $0 \leq w(E) \leq \mathbb{E} \, \|\mathbf{g}\| < \sqrt{n}$.

The width can be calculated accurately for many sets of interest. In particular, if L is an arbitrary k-dimensional subspace of $\mathbb{R}^n$, then

$$\sqrt{k-1} < w(L \cap \mathbb{S}^{n-1}) < \sqrt{k}. \tag{8.3}$$

Indeed, it is productive to think about the *squared* width $w^2(E)$ as a measure of the "dimension" of the set E.

Remark 8.3 (Statistical dimension)**.** *The* statistical dimension *is another measure of content that is closely related to the squared Gaussian width. The statistical dimension has additional geometric properties that make it easier to work with in some contexts. See (Amelunxen, Lotz, McCoy and Tropp 2014, McCoy and Tropp 2013, McCoy and Tropp 2014) and (Goldstein, Nourdin and Peccati 2017) for more information.*

8.5. **Restricted singular values of Gaussian matrices.** In a classic work on Banach space geometry, Gordon (1988) showed that the Gaussian width controls both the minimum and maximum restricted singular values of a subset of the sphere.

Theorem 8.4 (Restricted singular values: Gaussian matrix)**.** *Fix a subset $E \subseteq \mathbb{S}^{n-1}(\mathbb{R})$ of the unit sphere. Draw a Gaussian matrix $\boldsymbol{\Gamma} \in \mathbb{R}^{d\times n}$ whose entries are iid* NORMAL$(0, d^{-1})$*. For all $t > 0$,*

$$\mathbb{P}\left\{\sigma_{\min}(\boldsymbol{\Gamma}; E) \le 1 - \frac{w(E)+1}{\sqrt{d}} - t\right\} \le \mathrm{e}^{-dt^2/2};$$
$$\mathbb{P}\left\{\sigma_{\max}(\boldsymbol{\Gamma}; E) \ge 1 + \frac{w(E)}{\sqrt{d}} + t\right\} \le \mathrm{e}^{-dt^2/2}.$$

Proof. (Sketch) The first inequality is a consequence of Gordon's minimax theorem and Gaussian concentration. The second inequality is essentially Chevet's theorem, which follows from Slepian's lemma. See Ledoux and Talagrand (1991, Chap. 3.3) for an overview of these ideas. □

Theorem 8.4 yields the relations

$$1 - \frac{w(E)+1}{\sqrt{d}} \lessapprox \sigma_{\min}(\boldsymbol{\Gamma}; E) \le \sigma_{\max}(\boldsymbol{\Gamma}; E) \lessapprox 1 + \frac{w(E)}{\sqrt{d}}.$$

In other words, the embedding dimension should satisfy $d > (w(E)+1)^2$ to ensure that the map $\boldsymbol{\Gamma}$ is unlikely to annihilate any point in E. For this choice of d, the random embedding is unlikely to dilate any point in E by more than a factor of two.

As a consequence, we have reduced the problem of computing embedding dimensions for Gaussian maps to the problem of computing Gaussian widths. In the next two subsections, we work out two important examples.

Remark 8.5 (Optimality)**.** *The statements in Theorem 8.4 are nearly optimal. One way to see this is to consider the set $E = \mathbb{S}^{n-1}(\mathbb{R})$, for which the theorem implies that*

$$1 - \sqrt{n/d} \lessapprox \mathbb{E}\,\sigma_{\min}(\boldsymbol{\Gamma}) \le \mathbb{E}\,\sigma_{\max}(\boldsymbol{\Gamma}) \le 1 + \sqrt{n/d}.$$

The Bai–Yin law (Bai and Silverstein 2010, Sec. 5.2) confirms that the first and last inequality are sharp as $n, d \to \infty$ with $n/d \to \text{const} \in [0, 1]$.

Moreover, if E is spherically convex (i.e., the intersection of a convex cone with the unit sphere), then the minimum restricted singular value satisfies the reverse inequality

$$\mathbb{P}\left\{\sigma_{\min}(\boldsymbol{\Gamma}; E) \ge 1 - \frac{w(E)}{\sqrt{d}} + t\right\} \le 2\mathrm{e}^{-dt^2/2}.$$

This result is adapted from (Thrampoulidis et al. 2014).

In addition, fifteen years of computational experiments have also shown that the predictions from Theorem 8.4 are frequently sharp. See Oymak and Tropp (2018) for some examples and references.

Remark 8.6 (History)**.** *The application of Gaussian comparison theorems in numerical analysis can be traced to work in mathematical signal processing. Rudelson and Vershynin (2008) used a corollary of Gordon's minimax theorem to study ℓ_1 minimization problems. Significant extensions and improvements of this argument were made by Stojnic (2010) and Chandrasekaran, Recht, Parrilo and Willsky (2012). Amelunxen et al. (2014, Rem. 2.9) seem to have been the first to recognize that Gordon's minimax theorem can be reversed in the presence of convexity. A substantial refinement of this observation appeared in Thrampoulidis et al. (2014). There is a long series of follow-up works by Babak Hassibi's group that apply this insight to other problems in signal processing and communications.*

8.6. **Example: Johnson–Lindenstrauss.** As a first application of Theorem 8.4, let us explain how it implies the classic dimension reduction result of Johnson and Lindenstrauss (1984).

8.6.1. *Overview.* Let $\{\mathbf{a}_1, \dots, \mathbf{a}_N\} \subset \mathbb{R}^n$ be a discrete point set. We would like to know when a Gaussian embedding $\mathbf{\Gamma} \in \mathbb{R}^{d \times n}$ approximately preserves all the pairwise distances between these points:

$$1 - \varepsilon \le \frac{\|\mathbf{\Gamma}(\mathbf{a}_i - \mathbf{a}_j)\|}{\|\mathbf{a}_i - \mathbf{a}_j\|} \le 1 + \varepsilon \quad \text{for all } i \ne j. \tag{8.4}$$

The question is how large we must set the embedding dimension d to achieve distortion $\varepsilon \in (0, 1)$.

8.6.2. *Analysis.* We can solve this problem using the machinery described in Section 8.5. Consider the set E of normalized chords:

$$E = \left\{ \frac{\mathbf{a}_i - \mathbf{a}_j}{\|\mathbf{a}_i - \mathbf{a}_j\|} : 1 \le i < j \le N \right\}.$$

By the definition (8.2) of the restricted singular values,

$$\sigma_{\min}(\mathbf{\Gamma}; E) \le \frac{\|\mathbf{\Gamma}(\mathbf{a}_i - \mathbf{a}_j)\|}{\|\mathbf{a}_i - \mathbf{a}_j\|} \le \sigma_{\max}(\mathbf{\Gamma}; E).$$

Therefore, we can invoke Theorem 8.4 to determine how the embedding dimension controls the distortion.

Let us summarize the argument. First, observe that the Gaussian width of the set E satisfies

$$w(E) = \mathbb{E} \max_{\mathbf{x} \in E} \langle \mathbf{g}, \mathbf{x} \rangle \le \sqrt{2 \log \#E} < 2\sqrt{\log(N/2)}.$$

As a consequence,

$$\begin{aligned} \mathbb{P}\left\{ \sigma_{\min}(\mathbf{\Gamma}; E) \le 1 - (1 + 2\sqrt{\log(N/2)})/\sqrt{d} - t \right\} &\le \mathrm{e}^{-dt^2/2}; \\ \mathbb{P}\left\{ \sigma_{\max}(\mathbf{\Gamma}; E) \ge 1 + 2\sqrt{\log(N/2)}/\sqrt{d} + t \right\} &\le \mathrm{e}^{-dt^2/2}. \end{aligned}$$

To achieve distortion ε with high probability, it is sufficient to choose

$$d \ge 8\varepsilon^{-2} \log N.$$

In other words, the embedding dimension only needs to be *logarithmic* in the cardinality N of the point set. With some additional calculation, we can also extract precise failure probabilities from this analysis.

8.6.3. *Discussion.* Let us close this example with a few comments. In spite of its prominence, the Johnson–Lindenstrauss embedding lemma is somewhat impractical. Indeed, since the embedding dimension d is proportional to ε^{-2}, it is a challenge to achieve small distortions. Even if we consider the setting where $\varepsilon \approx 1$, the uniform bound (8.4) may require the embedding dimension to be prohibitively large.

As a step toward more applicable results, note that the bound on the *minimum* restricted singular value is more crucial than the bound on the maximum restricted singular value, because the former ensures that no two points coalesce after the random embedding. Similarly, it is often more valuable to preserve the distances between nearby points than between far-flung points. This observation is the starting point for the theory of locality sensitive hashing (Gionis, Indyk and Motwani 1999).

8.6.4. *History.* Johnson and Lindenstrauss (1984) were concerned with a problem in Banach space geometry, namely the prospect of extending a Lipschitz function from a finite metric space into a Hilbert space. The famous lemma from their paper took on a life of its own when Linial, London and Rabinovich (1995) used it to design efficient approximation algorithms for some graph problems. Indyk and Motwani (1999) used random embeddings to develop new algorithms for the approximate nearest neighbor problem. (Alon et al. 1999, Alon et al. 2002) introduced the term *sketching*, and they showed how to use sketches to track streaming data. Soon afterwards, Papadimitriou et al. (2000) and Frieze et al. (2004) proposed using random embeddings and matrix sampling for low-rank matrix approximation, bringing these ideas into the realm of computational linear algebra.

8.7. **Example: Subspace embedding.** Next, we consider a question at the heart of randomized linear algebra. Can we embed an unknown subspace into a lower-dimensional space?

8.7.1. *Overview.* Suppose that L is a k-dimensional subspace in $\mathbb{R}^n$. We say that a dimension reduction map $\mathbf{S}$ is a *subspace embedding* for L with distortion $\varepsilon \in (0, 1)$ if

$$(1 - \varepsilon) \|\mathbf{x}\| \leq \|\mathbf{S}\mathbf{x}\| \leq (1 + \varepsilon) \|\mathbf{x}\| \quad \text{for every } \mathbf{x} \in L. \tag{8.5}$$

We say that $\mathbf{S}$ is *oblivious* if it can be constructed without knowledge of the subspace L, except for its dimension.

Two questions arise. First, what types of dimension reduction maps yield (oblivious) subspace embeddings? Second, how large must we choose the embedding dimension to achieve this outcome?

8.7.2. *Analysis.* Gaussian dimension reduction maps yield very good oblivious subspace embeddings. Theorem 8.4 easily furnishes the justification. Consider the unit sphere in the subspace: $E = L \cap \mathbb{S}^{n-1}(\mathbb{R})$. Then construct the Gaussian dimension reduction map $\mathbf{\Gamma} \in \mathbb{R}^{d \times n}$. In view of (8.3), we have

$$\begin{aligned} \mathbb{P}\left\{ \sigma_{\min}(\mathbf{\Gamma}; E) \leq 1 - (1 + \sqrt{k})/\sqrt{d} - t \right\} &\leq \mathrm{e}^{-dt^2/2}; \\ \mathbb{P}\left\{ \sigma_{\max}(\mathbf{\Gamma}; E) \geq 1 + \sqrt{k}/\sqrt{d} + t \right\} &\leq \mathrm{e}^{-dt^2/2}. \end{aligned}$$

As a specific example, we can set the embedding dimension $d = 2k$ to ensure that $\|\mathbf{\Gamma}\mathbf{x}\| = (1 \pm 0.8)\|\mathbf{x}\|$ simultaneously for all points $\mathbf{x} \in L$, except with probability e^{-ck}. In some applications of subspace embeddings, we can even choose the dimension as small as $d = k+5$ or $d = k + 10$.

Many theoretical papers on randomized NLA use subspace embeddings as a primitive for designing algorithms for other linear algebra problems. For example, Section 10 describes several ways to use subspace embeddings to solve overdetermined least-squares problems.

8.7.3. *History.* Subspace embeddings were explicitly introduced by Sarlós (2006); see also Drineas, Mahoney and Muthukrishnan (2006*d*). As work on randomized NLA accelerated, researchers became interested in more structured types of subspace embeddings; an early reference is Woolfe et al. (2008). Section 9 covers these extensions. See Woodruff (2014) for a theoretical perspective on randomized NLA where subspace embeddings take pride of place.

8.8. **Universality of the minimum restricted singular value.** We have seen how to apply Gaussian dimension reduction for embedding discrete point sets and for embedding subspaces. Theorem 8.4 contains precise theoretical results on the behavior of Gaussian maps in terms of the Gaussian width. To what extent can we transfer this analysis to other types of random embeddings?

The following theorem (Oymak and Tropp 2018, Thm. 9.1) shows that the bound on the *minimum* restricted singular value in Theorem 8.4 is universal for a large class of random embeddings. In particular, this class includes sparse random matrices, whose nonzero entries compose a vanishing proportion of the total.

Theorem 8.7 (Universality)**.** *Fix a set $E \subseteq \mathbb{S}^{n-1}$. Let $\mathbf{S} \in \mathbb{R}^{d \times n}$ be a random matrix whose entries are independent random variables that satisfy*

$$\mathbb{E}[(\mathbf{S})_{ij}] = 0, \quad \mathbb{E}[(\mathbf{S})_{ij}^2] = d^{-1}, \quad \mathbb{E}[(\mathbf{S})_{ij}^5] \leq R.$$

When $d \leq n$, with high probability,

$$\sigma_{\min}(\mathbf{S}; E) \geq 1 - \frac{w(E)}{\sqrt{d}} - o(\sqrt{n/d}).$$

The constant in $o(\sqrt{n/d})$ depends only on R. A matching lower bound for $\sigma_{\min}(\mathbf{S}; E)$ holds when E is spherically convex.

In other words, if E is a moderately large set, the distribution of the entries of the random map $\mathbf{S}$ does not have an impact on the embedding dimension d sufficient to ensure no point in E is annihilated.

Theorem 8.7 is confirmed by extensive numerical experiments (Oymak and Tropp 2018), which demonstrate that dimension reduction maps with independent, standardized entries have identical performance for a wide range of examples.

It is perhaps surprising that the bound on the *maximum* restricted singular value from Theorem 8.4 is not universal. For some sets E, the quantity $\sigma_{\max}(\mathbf{S}; E)$ depends heavily on the distribution of the entries of $\mathbf{S}$.

Remark 8.8 (Universality for least-squares)**.** *Dobriban and Liu (2018) give some asymptotic universality results for random embeddings in the context of least-squares problems.*

8.9. **Random partial isometries.** Last, we consider a variant of Gaussian embedding that is more suitable when the embedding dimension d is close to the ambient dimension n. In this section, we allow the field $\mathbb{F}$ to be real or complex.

First, suppose that $d \leq n$, and let $\mathbf{\Gamma} \in \mathbb{F}^{d\times n}$ be a Gaussian embedding. Almost surely, the co-range of $\mathbf{\Gamma}$ is a uniformly random d-dimensional subspace of $\mathbb{F}^n$. Construct an embedding $\mathbf{S} \in \mathbb{F}^{d\times n}$ with orthonormal rows that span the co-range of $\mathbf{\Gamma}$, for example by QR factorization.

Similarly, we can consider a Gaussian embedding $\mathbf{\Gamma} \in \mathbb{F}^{d\times n}$ with $d \geq n$. In this case, the range of $\mathbf{\Gamma}$ is almost surely a uniformly random n-dimensional subspace of $\mathbb{F}^d$. Construct an embedding $\mathbf{S} \in \mathbb{F}^{d\times n}$ with orthonormal columns that span the range of $\mathbf{\Gamma}$, for example by QR factorization.

In each case, we call $\mathbf{S}$ a *random partial isometry*. The cost of storing a random partial isometry is $O(dn)$, and the cost of applying it to a vector is $O(dn)$. (We should warn the punctilious reader that QR factorization of $\mathbf{\Gamma}$ may not produce a matrix $\mathbf{S}$ that is Haar-distributed on the Stiefel manifold. To achieve this guarantee, use the algorithms from Mezzadri (2007).)

When $d \approx n$, random partial isometries are better embeddings than Gaussian maps (because the nonzero singular values of a partial isometry are all equal). When d and n are significantly different, the two models are quite similar to each other.

Thrampoulidis and Hassibi (2015) have established some theoretical results on the embedding behavior of real partial isometries ($\mathbb{F} = \mathbb{R}$). Unfortunately, the situation is more complicated than in Theorem 8.4. More relations between Gaussian matrices and partial isometries follow from the Marcus–Pisier comparison theorem (Marcus and Pisier 1981); see also Tropp (2012*a*).

9. Structured random embeddings

Gaussian embeddings and random partial isometries work extremely well. But they are not suitable for all practical applications because they are expensive to construct, to store, and to apply to vectors. Instead, we may prefer to implement more structured embedding matrices that alleviate these burdens.

This section summarizes a number of constructions that have appeared in the literature, with a focus on methods that have been useful in applications. Except as noted, these approaches have the same practical performance as either a Gaussian embedding or a random partial isometry.

Although many of these approaches are supported by theoretical analysis, the results are far less precise than for Gaussian embeddings. As such, we will not give detailed mathematical statements about structured embeddings. See Section 9.7 for a short discussion about how to manage the lack of theoretical guarantees.

9.1. **General techniques.** Many types of structured random embeddings operate on the same principle, articulated in (Ailon and Chazelle 2009). When we apply the random embedding to a fixed vector, it should homogenize ("mix") the coordinates so that each one carries about the same amount of energy. Then the embedding can sample coordinates at random to extract a lower-dimensional vector whose norm is proportional to the norm of the original vector and

has low variance. Random embeddings differ in how they perform the initial mixing step. Regardless of how it is done, mixing is very important for obtaining embeddings that work well in practice.

With this intuition at hand, let us introduce a number of pre- and post-processing transforms that help us design effective random embeddings. These approaches are used in many of the constructions below.

We say that a random variable is a *random sign* if it is UNIFORM$\{z \in \mathbb{F} : |z| = 1\}$. A random matrix $\mathbf{E} \in \mathbb{F}^{n \times n}$ is called a *random sign flip* if it is diagonal, and the diagonal entries are iid random sign variables.

A *random permutation* $\mathbf{\Pi} \in \mathbb{F}^{n \times n}$ is a matrix drawn uniformly at random from the set of permutation matrices. That is, each row and column of $\mathbf{\Pi}$ has a single nonzero entry, which equals one, and all such matrices are equally likely.

For $d \leq n$, a random matrix $\mathbf{R} \in \mathbb{F}^{d \times n}$ is called a *random restriction* if it selects d uniformly random entries from its input. With an abuse of terminology, we extend the definition to the case $d \geq n$ by making $\mathbf{R} \in \mathbb{F}^{d \times n}$ the matrix that embeds its input into the first n coordinates of the output. That is, $(\mathbf{R})_{ij} = 1$ when $i = j$ and zero otherwise.

Random sign flips and permutations are useful for preconditioning the input to a random embedding. Random restrictions are useful for reducing the dimension of a vector that has already been homogenized.

9.2. **Sparse sign matrices.** Among the earliest proposals for non-Gaussian embedding is to use a sparse random matrix whose entries are random signs.

Here is an effective construction of a sparse sign matrix $\mathbf{S} \in \mathbb{F}^{d \times n}$. Fix a sparsity parameter ζ in the range $2 \leq \zeta \leq d$. The random embedding takes the form

$$\mathbf{S} = \sqrt{\frac{n}{\zeta}} \begin{bmatrix} \mathbf{s}_1 & \dots & \mathbf{s}_n \end{bmatrix} \in \mathbb{F}^{d \times n}.$$

The columns $\mathbf{s}_i \in \mathbb{F}^d$ are iid random vectors. To construct each column, we draw ζ iid random signs, and we situate them in ζ uniformly random coordinates. Tropp, Yurtsever, Udell and Cevher (2019) recommend choosing $\zeta = \min\{d, 8\}$ in practice.

We can store a sparse embedding using about $O(\zeta n \log d)$ numbers. We can apply it to a vector in $\mathbb{F}^n$ with $O(\zeta n)$ arithmetic operations. The main disadvantage is that we must use sparse data structures and arithmetic to achieve these benefits. Sparse sign matrices have similar performance to Gaussian embeddings.

Cohen (2016) has shown that a sparse sign matrix serves as an oblivious subspace embedding with constant distortion for an arbitrary k-dimensional subspace of $\mathbb{R}^n$ when the embedding dimension $d = O(k \log k)$ and the per-column sparsity $\zeta = O(\log k)$. It is conjectured that improvements are still possible.

Remark 9.1 (History)**.** *Sparse random embeddings emerged from the work of Achlioptas (2003) and Charikar, Chen and Farach-Colton (2004). For randomized linear algebra applications, sparse embeddings were promoted in (Clarkson and Woodruff 2013, Meng and Mahoney 2013, Nelson and Nguyen 2013) and (Urano 2013). Analyses of the embedding behavior of a sparse map appear in (Bourgain, Dirksen and Nelson 2015) and (Cohen 2016).*

9.3. **Subsampled trigonometric transforms.** Another type of structured randomized embeddings is designed to mimic the performance of a random partial isometry. One important class of examples consists of the subsampled randomized trigonometric transforms (SRTTs).

To construct a random embedding $\mathbf{S} \in \mathbb{F}^{d \times n}$ with $d \leq n$, we select a unitary trigonometric transform $\mathbf{F} \in \mathbb{F}^{n \times n}$. Then we form

$$\mathbf{S} = \sqrt{\frac{n}{d}} \mathbf{R}\mathbf{F}\mathbf{E}\mathbf{\Pi},$$

where $\mathbf{R} \in \mathbb{F}^{d \times n}$ is a random restriction, $\mathbf{E} \in \mathbb{F}^{n \times n}$ is a random sign flip, and $\mathbf{\Pi} \in \mathbb{F}^{n \times n}$ is a random permutation. Note that $\mathbf{S}$ is a partial isometry.

The trigonometric transform F can be any one of the usual suspects. In the complex case ($\mathbb{F} = \mathbb{C}$), we often use a discrete Fourier transform. In the real case ($\mathbb{F} = \mathbb{R}$), common choices are the discrete cosine transform (DCT2) or the discrete Hartley transform (DHT). When n is a power of two, we can consider a Walsh–Hadamard transform (WHT). The paper (Avron et al. 2010) reports that the DHT is the best option in the real case.

The cost of storing a SRTT is $O(n \log n)$, and it can be applied to a vector in $O(n \log d)$ operations using a fast subsampled trigonometric transform algorithm. The main disadvantage is that it requires a good implementation of the fast transform.

Tropp (2011*b*) has shown that an SRTT serves as an oblivious subspace embedding with constant distortion for an arbitrary k-dimensional subspace of $\mathbb{F}^n$ provided that $d = O(k \log k)$. This paper focuses on the Walsh–Hadamard transform, but the analysis extends to other SRTTs. In practice, it often suffices to choose $d = O(k)$, but no rigorous justification is available.

Remark 9.2 (Rerandomization)**.** *It is also common to repeat the randomization and trigonometric transformations:*

$$\mathsf{S} = \sqrt{\frac{n}{d}}\mathsf{RFE}'\mathsf{\Pi}'\mathsf{FE}\mathsf{\Pi},$$

with an independent sign flip E' and an independent permutation $\mathsf{\Pi}'$. This enhancement can make the embedding more robust, although it is not always necessary.

Remark 9.3 (History)**.** *Parker (1995) proposed the use of randomized trigonometric transforms to precondition linear systems. The idea of applying an SRTT for dimension reduction appears in (Ailon and Chazelle 2006) and (Ailon and Chazelle 2009). Woolfe et al. (2008) develop algorithms for low-rank matrix approximation based on SRTTs. Embedding properties of an SRTT for general sets follow from (Rudelson and Vershynin 2008) and (Krahmer and Ward 2011); see Foucart and Rauhut (2013, Chap. 12) or Pilanci and Wainwright (2015).*

9.4. **Tensor random projections.** Next, we describe a class of random embeddings that are useful for very large linear algebra and multilinear algebra problems. This approach invokes tensor products to form a random embedding for a high-dimensional space from a family of random embeddings for lower-dimensional spaces.

Let $\mathsf{S}_1 \in \mathbb{F}^{d \times m_1}$ and $\mathsf{S}_2 \in \mathbb{F}^{d \times m_2}$ be statistically independent random embeddings. We define the *tensor random embedding*

$$\mathsf{S} := \mathsf{S}_1 \odot \mathsf{S}_2 \in \mathbb{F}^{d \times n} \quad \text{where} \quad n = m_1 m_2$$

to be the Khatri–Rao product of S_1 and S_2. That is, the ith row of S is

$$(\mathsf{S})_{i:} = \big[(\mathsf{S}_1)_{i1}(\mathsf{S}_2)_{i:} \quad \dots \quad (\mathsf{S}_1)_{im}(\mathsf{S}_2)_{i:}\big] \quad \text{for } i = 1, \dots, d.$$

Under moderate assumptions on the component embeddings S_1 and S_2, the tensor random embedding S preserves the squared Euclidean norm of an arbitrary vector in $\mathbb{F}^n$.

A natural extension of this idea is to draw many component embeddings $\mathsf{S}_i \in \mathbb{F}^{d \times m_i}$ for $i = 1, \dots, k$ and to form the tensor random embedding

$$\mathsf{S} := \mathsf{S}_1 \odot \mathsf{S}_2 \odot \cdots \odot \mathsf{S}_k \in \mathbb{F}^{d \times n} \quad \text{where} \quad n = \prod\nolimits_{i=1}^{k} m_i.$$

This embedding also inherits nice properties from its components.

The striking thing about this construction is that the tensor product embedding operates on a *much* larger space than the component embeddings. The storage cost for the component embeddings is $O(d(\sum_{i=1}^{k} m_i))$, or less. We can apply the tensor random embedding to a vector directly with $O(dn)$ arithmetic. We can accelerate the process by using component embeddings that have fast transforms, and we can obtain improvements for vectors that have a compatible tensor product structure. Some theoretical analysis is available, but results are not yet complete.

Remark 9.4 (History)**.** *Tensor random embeddings were introduced by Kasiviswanathan, Rudelson, Smith and Ullman (2010) on differential privacy. They were first analyzed by Rudelson (2012). The paper (Sun, Guo, Tropp and Udell 2018) proposed the application of tensor random embeddings for randomized linear algebra; some extensions appear in (Jin, Kolda and Ward 2019) and (Malik and Becker 2019). See (Baldi and Vershynin 2019) and (Vershynin 2019) for related theoretical results.*

9.5. **Other types of structured random embeddings.** We have described the random embeddings that have received the most attention in the NLA literature. Yet there are other types of random embeddings that may be useful in special circumstances. Some examples include random filters (Tropp, Wakin, Duarte, Baron and Baraniuk 2006, Krahmer and Ward 2011, Rauhut, Romberg and Tropp 2012, Mendelson, Rauhut and Ward 2018), the Kac random walk (Kac 1956, Rosenthal 1994, Oliveira 2009*b*, Pillai and Smith 2017), and sequences of random reflections (Sloane 1983, Porod 1996). Liberty (2009) discusses a number of other instances.

9.6. **Coordinate sampling.** So far, we have discussed random embeddings that mix up the coordinates of a vector. It is sometimes possible to construct embeddings just by sampling coordinates at random. Coordinate sampling can be appealing in specialized situations (e.g., kernel computations), where we only have access to individual entries of the data. On the other hand, this approach requires strong assumptions, and it is far less reliable than random embeddings that mix coordinates. In this section, we summarize the basic facts about subspace embedding via random coordinate sampling.

A note on terminology: we will use the term *coordinate sampling* to distinguish these maps from random embeddings that mix coordinates.

9.6.1. *Coherence and leverage.* Let $L \subset \mathbb{F}^n$ be a k-dimensional subspace. The *coherence* $\mu(L)$ of the subspace with respect to the standard basis is

$$\mu(L) := n \cdot \max_{i=1,\dots,n} \|\mathbf{P}_L \boldsymbol{\delta}_i\|^2,$$

where $\mathbf{P}_L \in \mathbb{H}_n$ is the orthogonal projector onto L and $\boldsymbol{\delta}_i$ is the ith standard basis vector. The coherence $\mu(L)$ lies in the range $[k, n]$. The behavior of coordinate sampling methods degrades as the coherence increases.

Next, define the (subspace) *leverage score* distribution with respect to the standard coordinate basis:

$$p_i = \frac{1}{k}\|\mathbf{P}_L \boldsymbol{\delta}_i\|^2 \quad \text{for } i = 1, \dots, n.$$

It is straightforward to verify that $(p_1, \dots, p_n)$ is a probability distribution. In most applications, it is expensive to compute or estimate subspace leverage scores because we typically do not have a basis for the subspace L at hand.

9.6.2. *Uniform sampling.* We can construct an embedding $\mathbf{S} \in \mathbb{F}^{d \times n}$ by sampling each output coordinate uniformly at random. That is, the rows of $\mathbf{S}$ are iid, and each row takes values $\boldsymbol{\delta}_i/\sqrt{d}$, each with probability $1/n$. (We can also sample coordinates uniformly *without* replacement; this approach performs slightly better but requires more work to analyze.)

The embedding dimension d must be chosen to ensure that

$$\|\mathbf{S}\mathbf{x}\|^2 = (1 \pm \varepsilon)\|\mathbf{x}\|^2 \quad \text{for all } \mathbf{x} \in L. \tag{9.1}$$

To achieve this goal, it suffices that

$$d \geq 2\varepsilon^{-2}\mu(L)\log(2k).$$

In other words, the embedding dimension is proportional to the coherence of the subspace, up to a logarithmic factor. We expect uniform sampling to work well precisely when the coherence is small ($\mu(L) \approx k$).

To prove this result, let $\mathbf{U} \in \mathbb{F}^{n \times k}$ be an orthonormal basis for the subspace L. We can approximate the product $\mathbf{I}_k = \mathbf{U}^*\mathbf{U}$ by sampling columns of $\mathbf{U}$ uniformly at random. The analysis in Section 7.3.2 furnishes the conclusion.

9.6.3. *Leverage score sampling.* When the coherence is large, it seems more natural to sample with respect to the leverage score distribution $(p_1, \dots, p_n)$ described above. That is, the embedding $\mathbf{S} \in \mathbb{F}^{d \times n}$ has iid rows, and each row takes value $\boldsymbol{\delta}_i / \sqrt{d}$ with probability p_i.

To achieve the embedding guarantee (9.1) with this sampling distribution, we should choose the embedding dimension

$$d \geq 2\varepsilon^{-2} k \log(2k).$$

In other words, it suffices that the embedding dimension is proportional to the dimension k of the subspace, up to the logarithmic factor. This result follows from the analysis of matrix multiplication by importance sampling (Section 7.3.3).

9.6.4. *Discussion.* Uniform sampling leads to an oblivious subspace embedding, although it is not obvious how to select the embedding dimension in advance because the coherence is usually not available. Leverage score sampling is definitely not oblivious, because we need to compute the sampling probabilities (potentially at great cost).

In practice, uniform sampling works better than one might anticipate, and it has an appealing computational profile. As a consequence, it has become a workhorse for large-scale kernel computation; see (Kumar, Mohri and Talwalkar 2012, Bach 2013) and (Rudi, Carratino and Rosasco 2017).

Our experience suggests that leverage score sampling is rarely a competitive method for constructing subspace embeddings, especially once we take account of the effort required to compute the sampling probabilities. We recommend using other types of random embeddings (Gaussians, sparse maps, SRTTs) in lieu of coordinate sampling whenever possible.

Although coordinate sampling may seem like a natural approach to construct matrix approximations involving rows or columns, we can obtain better algorithms for this problem by using (mixing) random embeddings. See Section 13 for details.

Coordinate sampling can be a compelling choice in situations where other types of random embeddings are simply unaffordable. For instance, a variant of leverage score sampling leads to effective algorithms for kernel ridge regression; see Rudi, Calandriello, Carratino and Rosasco (2018) for evidence. Indeed, in the context of kernel computations, coordinate sampling and random features may be the only tractable methods for extracting information from the kernel matrix. We discuss these ideas in Section 19.

An emerging research direction uses coordinate sampling for solving certain kinds of continuous problems, such as function interpolation. In this setting, sampling corresponds to function evaluation, while mixing embeddings may lead to operations that are impossible to implement in the continuous space. For some examples, see (Rauhut and Ward 2012, Cohen, Davenport and Leviatan 2013, Hampton and Doostan 2015, Rauhut and Ward 2016, Cohen and Migliorati 2017, Arras, Bachmayr and Cohen 2019, Avron, Kapralov, Musco, Musco, Velingker and Zandieh 2019) and (Chen and Price 2019).

9.6.5. *History.* Most of the early theoretical computer science papers on randomized NLA rely on coordinate sampling methods. These approaches typically construct an importance sampling distribution using the norms of the rows or columns of a matrix. For examples, see (Frieze et al. 2004, Drineas and Mahoney 2005) and (Drineas et al. 2006*a*, Drineas et al. 2006*b*, Drineas et al. 2006*c*). These papers measure errors in the Frobenius norm. The first spectral norm analysis of coordinate sampling appears in Rudelson and Vershynin (2007).

Leverage scores are a classical tool in statistical regression, used to identify influential data points. Drineas, Mahoney and Muthukrishnan (2008) proposed the subspace leverage scores as a sampling distribution for constructing low-rank matrix approximations. Mahoney and Drineas (2009) identified the connection with regression. Mahoney (2011) made a theoretical case for using leverage scores as the basis for randomized NLA algorithms. Alaoui and Mahoney (2015) introduced an alternative definition of leverage scores for kernel ridge regression.

It is somewhat harder to trace the application of uniform sampling in randomized NLA. Several authors have studied the behavior of uniform sampling in the context of Nyström approximation; see (Williams and Seeger 2001, Kumar et al. 2012) and (Gittens 2013). An

analysis of uniform coordinate sampling is implicit in the theory on SRTTs; see Tropp (2011*b*, Lemma 3.4). See Kannan and Vempala (2017) for more discussion about sampling methods in NLA.

9.7. **But how does it work in theory?** Structured random embeddings and random coordinate sampling lack the precise guarantees that we can attribute to Gaussian embedding matrices. So how can we apply them with confidence?

First, we advocate using *a posteriori* error estimators to assess the quality of the output of a randomized linear algebra computation. These error estimators are often quite cheap, yet they can give (statistical) evidence that the computation was performed correctly. We also recommend adaptive algorithms that can detect when the accuracy is insufficient and make refinements. With this approach, it is not pressing to produce theory that justifies all of the internal choices (e.g., the specific type of random embedding) in the NLA algorithm. See Section 12 for further discussion.

Even so, we would like to have *a priori* predictions about how our algorithms will behave. Beyond that, we need reliable methods for selecting algorithm parameters, especially in the streaming setting where we cannot review the data and repeat the computation.

Here is one answer to these concerns. As a practical matter, we can simply invoke the lessons from the Gaussian theory, even when we are using a different type of random embedding. The universality result, Theorem 8.7, gives a rationale for this approach in one special case. We also recommend undertaking computational experiments to verify that the Gaussian theory gives an adequate description of the observed behavior of an algorithm.

Warning 9.5 (Coordinate sampling)**.** *Mixing random embeddings perform similarly to Gaussian embeddings, but coordinate sampling methods typically exhibit behavior that is markedly worse.*

10. How to use random embeddings

Algorithm designers have employed random embeddings for many tasks in linear algebra, optimization, and related areas. Methods based on random embedding fall into three rough categories: (1) sketch and solve, (2) iterative sketching, and (3) sketch and precondition. To draw distinctions among these paradigms, we use each one to derive an algorithm for solving an overdetermined least-squares problem.

10.1. **Overdetermined least-squares.** Overdetermined least-squares problems sometimes arise in statistics and data-analysis applications. We may imagine that some of the data in these problems is redundant. As such, it seems plausible that we could reduce the size of the problem to accelerate computation without too much loss in accuracy.

Consider a matrix $\mathbf{A} \in \mathbb{F}^{m \times n}$ with $m \gg n$ and a vector $\mathbf{b} \in \mathbb{F}^m$. An overdetermined least-squares problem has the form

$$\operatorname*{minimize}_{\mathbf{x} \in \mathbb{F}^n} \frac{1}{2} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|^2. \tag{10.1}$$

Following Pilanci and Wainwright (2015), we can also rewrite the least-squares problem to emphasize the role of the matrix:

$$\operatorname*{minimize}_{\mathbf{x} \in \mathbb{F}^n} \frac{1}{2} \|\mathbf{A}\mathbf{x}\|^2 - \langle \mathbf{x}, \mathbf{A}^*\mathbf{b} \rangle. \tag{10.2}$$

Write $\mathbf{x}_\star$ for an arbitrary solution to the problem (10.1).

To make a clear comparison among algorithm design templates, we will assume that $\mathbf{A}$ is dense and unstructured. In this case, the classical approach to solving (10.1) is based on factorization of the coefficient matrix (such as QR or SVD) at a cost of $O(mn^2)$ arithmetic operations.

When $\mathbf{A}$ is sparse, we would typically use iterative methods (such as CG), which have a different computational profile. For sparse matrices, we would also make different design choices in a sketch-based algorithm. Nevertheless, for simplicity, we will not discuss the sparse case.

10.2. **Subspace embeddings for least-squares.** To design a sketching algorithm for the overdetermined least-squares problem (10.1), we need to construct a subspace embedding $\mathbf{S} \in \mathbb{F}^{d \times m}$ that preserves the geometry of the range of the matrix $\mathbf{A} \in \mathbb{F}^{m \times n}$. In some cases, we may also need the embedding to preserve the range of the bordered matrix $\begin{bmatrix} \mathbf{A} & \mathbf{b} \end{bmatrix} \in \mathbb{F}^{m \times (n+1)}$.

Since the matrix $\mathbf{A}$ is dense and unstructured, we will work with a structured subspace embedding, such as an SRTT (Section 9.3). More precisely, we will assume that evaluating the product $\mathbf{SA}$ costs only $O(mn \log d)$ arithmetic operations. The best theoretical results for these structured sketches require that the embedding dimension $d \sim n \log(n)/\varepsilon^2$ to achieve distortion ε, although the logarithmic factor seems to be unnecessary in practice.

Throughout this section, we use the heuristic notation $\sim$ to indicate quantities that are proportional. We also write $\ll$ to mean "much smaller than."

10.3. **Sketch and solve.** The sketch-and-solve paradigm maps the overdetermined least-squares problem (10.1) into a smaller space. Then it uses the solution to the reduced problem as a proxy for the solution to the original problem. This approach can be very fast, and we only need one view of the matrix $\mathbf{A}$. On the other hand, the results tend to be very inaccurate.

Let $\mathbf{S} \in \mathbb{F}^{d \times m}$ be a subspace embedding for the range of $\begin{bmatrix} \mathbf{A} & \mathbf{b} \end{bmatrix}$ with distortion ε. Consider the compressed least-squares problem

$$\operatorname*{minimize}_{\mathbf{x} \in \mathbb{R}^n} \frac{1}{2} \|\mathbf{S}(\mathbf{A}\mathbf{x} - \mathbf{b})\|^2. \tag{10.3}$$

Since $\mathbf{S}$ preserves geometry, we may hope that the solution $\widehat{\mathbf{x}}$ to the sketched problem (10.3) can replace the solution $\mathbf{x}_\star$ to the original problem (10.2). A typical theoretical bound is

$$\|\mathbf{A}\widehat{\mathbf{x}} - \mathbf{b}\| \le (1 + \varepsilon)\|\mathbf{A}\mathbf{x}_\star - \mathbf{b}\| \quad \text{when } d \sim n \log(n)/\varepsilon^2.$$

See Sarlós (2006). Although the residuals are comparable, it need *not* be the case that $\widehat{\mathbf{x}} \approx \mathbf{x}_\star$, even when the solution to (10.1) is unique.

The sketch-and-solve paradigm requires us to form the matrix $\mathbf{SA}$ at a cost of $O(mn \log d)$ operations. We would typically solve the (dense) reduced problem with a direct method, using $O(dn^2)$ operations. Assuming $d \sim n \log(n)/\varepsilon^2$, the total arithmetic cost is $O(mn \log(n/\varepsilon^2) + n^3 \log(n)/\varepsilon^2)$.

In summary, we witness an improvement in computational cost over classical methods if $\log n \ll n \ll m/\log n$ and ε is constant. But we must also be willing to accept large errors, because we cannot make ε small.

Remark 10.1 (History)**.** *The sketch-and-solve paradigm is attributed to Sarlós (2006). It plays a major role in the theoretical algorithms literature; see Woodruff (2014) for advocacy. It has also been proposed for enormous problems that might otherwise be entirely hopeless (Lim and Weare 2017).*

10.4. **Iterative sketching.** Iterative sketching attempts to remediate the poor accuracy of the sketch-and-solve paradigm by applying it repeatedly to reduce the residual error.

First, we construct an initial solution $\mathbf{x}_0 \in \mathbb{F}^n$ using the sketch-and-solve paradigm with a constant distortion embedding. For each iteration i, draw a fresh random subspace embedding $\mathbf{S}_i \in \mathbb{F}^{d \times m}$ for $\operatorname{range}(\mathbf{A})$, with constant distortion. We can solve a sequence of least-squares problems

$$\operatorname*{minimize}_{\mathbf{x} \in \mathbb{R}^n} \frac{1}{2} \|\mathbf{S}_i\mathbf{A}(\mathbf{x} - \mathbf{x}_{i-1})\|^2 + \langle \mathbf{x} - \mathbf{x}_{i-1},\, \mathbf{A}^*(\mathbf{b} - \mathbf{A}\mathbf{x}_{i-1}) \rangle. \tag{10.4}$$

The solution $\mathbf{x}_i$ to this subproblem is fed into the next subproblem. Without the sketch, each subproblem is equivalent to solving (10.2) with $\mathbf{b}$ replaced by the residual $\mathbf{r}_{i-1} = \mathbf{b} - \mathbf{A}\mathbf{x}_{i-1}$. The sketch $\mathbf{S}_i$ preserves the geometry while reducing the problem size. A typical theoretical error bound would be

$$\|\mathbf{A}\mathbf{x}_j - \mathbf{b}\| \le (1 + \varepsilon)\|\mathbf{A}\mathbf{x}_\star - \mathbf{b}\| \quad \text{when } j \sim \log(1/\varepsilon) \text{ and } d \sim n \log n.$$

See (Pilanci and Wainwright 2016) for related results.

In each iteration, the iterative sketching approach requires us to form $\mathbf{S}_i\mathbf{A}$ at a cost of $O(mn \log d)$. We compute $\mathbf{A}^*\mathbf{r}_{i-1}$ at a cost of $O(mn)$. Although it is unnecessary to solve each subproblem accurately, we cannot obtain reliable behavior without using a dense method at a cost of $O(dn^2)$ per iteration. With the theoretical parameter choices, the total arithmetic is $O((mn + n^3) \log(n) \log(1/\varepsilon))$ to achieve relative error ε.

The interesting parameter regime is $\log n \ll n \ll m/\log n$, but we can now allow ε to be tiny. In this setting, iterative sketching costs slightly more than the sketch-and-solve paradigm to achieve constant relative error, while it is faster than the classical approach. At the same time, it can produce errors as small as traditional least-squares algorithms. A shortcoming is that this method requires repeated sketches of the matrix $\mathbf{A}$.

For overdetermined least-squares, we can short-circuit the iterative sketching approach. In this setting, we can sketch the input matrix just once and factorize it. We can use the same factorized sketch in each iteration to solve the subproblems faster. For problems more general than least-squares, it may be necessary to extract a fresh sketch at each iteration, as we have done here.

Remark 10.2 (History)**.** *Iterative sketching can be viewed as an extension of stochastic approximation methods from optimization, for example stochastic gradient descent (Bottou 2010). In the context of randomized NLA, these algorithms first appeared in the guise of the randomized Kaczmarz iteration (Strohmer and Vershynin 2009); see Section 17.4. Gower and Richtárik (2015*b*) reinterpreted randomized Kaczmarz as an iterative sketching method and developed generalizations. Pilanci and Wainwright (2016) proposed a similar method for solving overdetermined least-squares problems with constraints; they observed that better numerical performance is obtained by sketching* (10.2) *instead of* (10.1)*.*

10.5. **Sketch and precondition.** The sketch-and-precondition paradigm uses random embedding to find a proxy for the input matrix. We can use this proxy to precondition a classical iterative algorithm so it converges in a minimal number of iterations.

Let $\mathbf{S} \in \mathbb{F}^{d\times m}$ be a subspace embedding for $\mathrm{range}(\mathbf{A})$ with constant distortion. Compress the input matrix $\mathbf{A}$, and then compute a (pivoted) QR factorization:

$$\mathbf{Y} = \mathbf{S}\mathbf{A} \quad \text{and} \quad \mathbf{Y} = \mathbf{Q}\mathbf{R}.$$

Since $\mathbf{S}$ preserves the range of $\mathbf{A}$ when $d \sim n \log n$, we anticipate that $\mathbf{Y}^*\mathbf{Y} \approx \mathbf{A}^*\mathbf{A}$. As a consequence, $\mathbf{A}\mathbf{R}^\dagger$ should be close to an isometry. Thus, we can pass to the preconditioned problem

$$\underset{\mathbf{x}\in\mathbb{R}^n}{\text{minimize}} \ \frac{1}{2}\|(\mathbf{A}\mathbf{R}^\dagger)(\mathbf{R}\mathbf{x}) - \mathbf{b}\|^2. \tag{10.5}$$

Construct an initial solution $\mathbf{x}_0 \in \mathbb{F}^n$ using the sketch-and-solve paradigm with the embedding $\mathbf{S}$. From this starting point, we solve (10.5) using preconditioned LQSR. The jth iterate satisfies

$$\|\mathbf{A}\mathbf{x}_j - \mathbf{b}\| \le (1+\varepsilon)\|\mathbf{A}\mathbf{x}_\star - \mathbf{b}\| \quad \text{when } j \sim \log(1/\varepsilon).$$

This statement is a reinterpretation of the theory in Rokhlin and Tygert (2008).

The cost of sketching the input matrix and performing the QR decomposition is $O(mn \log d + dn^2)$. Afterwards, we pay $O(mn)$ for each iteration of PCG. With the theoretical parameter settings, the total cost is $O(mn \log(nB/\varepsilon) + n^3 \log n)$ operations.

Once again, the interesting regime is $\log n \ll n \ll m/\log n$, and the value of ε can be very small. For overdetermined least-squares, this approach is faster than both the sketch-and-solve paradigm and the iterative sketching paradigm. The sketch-and-precondition approach leads to errors that are comparable with classical linear algebra algorithms, but it may be a factor of $n/\log(n)$ faster. On the other hand, it requires repeated applications of the matrix $\mathbf{A}$.

Remark 10.3 (History)**.** *The randomized preconditioning idea was proposed by Rokhlin and Tygert (2008). Avron et al. (2010) demonstrate that least-squares algorithms based on randomized preconditioning can beat the highly engineered software in LAPACK. The same*

Algorithm 7 *The randomized rangefinder.*
Implements the procedure from Section 11.1.1.

Input: Input matrix $\mathbf{B} \in \mathbb{F}^{m \times n}$, subspace dimension ℓ
Output: Orthonormal matrix $\mathbf{Q} \in \mathbb{F}^{m \times \ell}$

```
function RandomRangefinder(B, ℓ)
    Draw a random matrix Ω ∈ F^{n×ℓ}
    Form Y = BΩ
    Compute [Q, ∼] = qr_econ(Y)
```

method drives the algorithms in Meng, Saunders and Mahoney (2014). Avron (2018) contains a recent summary of existing randomized preconditioning methods.

10.6. **Comparisons.** If we seek a high-precision solution to a dense, unstructured, overdetermined least-squares problem, randomized preconditioning leads to the most efficient existing algorithm. For the same problem, if we can only view the input matrix once, then the sketch-and-solve paradigm still allows us to obtain a low-accuracy solution. Although iterative sketching is less efficient than its competitors in this setting, it remains useful for solving constrained least-squares problems, and it has further connections with optimization.

10.7. **Summary.** From the perspective of a numerical analyst, randomized preconditioning and iterative sketching should be the preferred methods for designing sketching algorithms because they allow for high precision. The sketch-and-solve approach is appropriate only when data access is severely constrained.

In spite of this fact, a majority of the literature on randomized NLA develops algorithms based on the sketch-and-solve paradigm. There are far fewer works on randomized preconditioning or iterative sketching. This discrepancy points to an opportunity for further research.

11. The randomized rangefinder

A core challenge in linear algebra is to find a subspace that captures a lot of the action of a matrix. We call this the *rangefinder* problem. As motivation for considering this problem, we will use the rangefinder to derive the randomized SVD algorithm. Then we will introduce several randomized algorithms for computing the rangefinder primitive, along with theoretical guarantees for these methods. These algorithms all make use of random embeddings, but their performance depends on more subtle features than the basic subspace embedding property.

In Section 12, we will complement the algorithmic discussion with details about error estimation and adaptivity for the rangefinder primitive. In Sections 13–16, we will see that the subspace produced by the rangefinder can be used as a primitive for other linear algebra computations.

Most of the material in this section is adapted from our papers (Halko et al. 2011*a*) and (Halko et al. 2011*b*). We have also incorporated more recent perspectives.

11.1. **The rangefinder: Problem statement.** Let $\mathbf{B} \in \mathbb{F}^{m \times n}$ be an input matrix, and let $\ell \leq \min\{m, n\}$ be the subspace dimension. The goal of the rangefinder problem is to produce an orthonormal matrix $\mathbf{Q} \in \mathbb{F}^{m \times \ell}$ whose range aligns with the dominant left singular vectors of $\mathbf{B}$.

To measure the quality of $\mathbf{Q}$, we use the spectral norm error

$$\|\mathbf{B} - \mathbf{Q}\mathbf{Q}^*\mathbf{B}\| = \|(\mathbf{I} - \mathbf{Q}\mathbf{Q}^*)\mathbf{B}\|. \tag{11.1}$$

If the error measure (11.1) is small, then the rank-ℓ matrix $\widehat{\mathbf{B}} = \mathbf{Q}\mathbf{Q}^*\mathbf{B}$ can serve as a proxy for $\mathbf{B}$. See Section 11.2 for an important application.

11.1.1. *The randomized rangefinder: A pseudoalgorithm.* Using randomized methods, it is remarkably easy to find an initial solution to the rangefinder problem. We simply multiply the target matrix by a random embedding and then orthogonalize the resulting matrix.

More rigorously: consider a target matrix $\mathbf{B} \in \mathbb{F}^{m \times n}$ and a subspace dimension ℓ. We draw a random test matrix $\mathbf{\Omega} \in \mathbb{F}^{n \times \ell}$, where $\mathbf{\Omega}^*$ is a mixing random embedding. We form the product $\mathbf{Y} = \mathbf{B\Omega} \in \mathbb{F}^{m \times \ell}$. Then we compute an orthonormal basis $\mathbf{Q} \in \mathbb{F}^{m \times \ell}$ for the range of $\mathbf{Y}$ using a QR factorization method. See Algorithm 7 for pseudocode.

In a general setting, the arithmetic cost of this procedure is dominated by $O(mn\ell)$ operations for the matrix–matrix multiplication. The QR factorization of $\mathbf{Y}$ requires $O(m\ell^2)$ arithmetic, and we also need to simulate the $n \times \ell$ random matrix $\mathbf{\Omega}$. Economies are possible when either $\mathbf{B}$ or $\mathbf{\Omega}$ admits fast multiplication.

11.1.2. *Practicalities.* To implement Algorithm 7 effectively, several computational aspects require attention.

- **How do we choose the subspace dimension?** If we have advance knowledge of the "effective rank" r of the target matrix $\mathbf{B}$, the theory (Theorem 11.5 and Corollary 11.9) indicates that we can select the subspace dimension ℓ to be just slightly larger, say, $\ell = r + p$ where $p = 5$ or $p = 10$. The value p is called the *oversampling* parameter. Alternatively, we can use an error estimator to decide when the computed subspace $\mathbf{Q}$ is sufficiently accurate; see Section 12.1.
- **What kind of random matrix?** We can use most types of mixing random embeddings to implement Algorithm 7. We highly recommend Gaussians and random partial isometries (Section 8). Sparse maps, SRTTs, and tensor random embeddings (Section 9) also work very well. In practice, all these approaches exhibit similar behavior; see Section 11.5 for more discussion. We present analysis only for Gaussian dimension reduction because it is both simple and precise.
- **Matrix multiplication.** The randomized rangefinder is powerful because most of the computation takes place in the matrix multiplication step, which is a highly optimized primitive on most computer systems. When the target matrix admits fast matrix–vector multiplications (e.g., due to sparsity), the rangefinder can exploit this property.
- **Powering.** As we will discuss in Sections 11.6 and 11.7, it is often beneficial to enhance Algorithm 7 by means of powering or Krylov subspace techniques.
- **Orthogonalization.** The columns of the matrix $\mathbf{Y}$ tend to be strongly aligned, so it is important to use a numerically stable orthogonalization procedure (Golub and Van Loan 2013, Chap. 5), such as Householder reflectors, double Gram–Schmidt, or rank-revealing QR. The rangefinder algorithm is also a natural place to invoke a TSQR algorithm (Demmel, Grigori, Hoemmen and Langou 2012).

See Halko et al. (2011*a*) for much more information.

11.2. **The randomized singular value decomposition (RSVD).** Before we continue with our discussion of the rangefinder, let us summarize one of the key applications: the randomized SVD algorithm.

Low-rank approximation problems often arise when a user seeks an incomplete matrix factorization that exposes structure, such as a truncated eigenvalue decomposition or a partial QR factorization. The randomized rangefinder, described in Section 11.1.1, can be used to perform the heavy lifting in these computations. Afterwards, we perform some light postprocessing to reach the desired factorization.

To illustrate how this works, suppose that we want to compute an approximate rank-ℓ truncated singular value decomposition of the input matrix $\mathbf{B} \in \mathbb{F}^{m \times n}$. That is,

$$\mathbf{B} \approx \mathbf{U\Sigma V}^*,$$

where $\mathbf{U} \in \mathbb{F}^{m \times \ell}$ and $\mathbf{V} \in \mathbb{F}^{n \times \ell}$ are orthonormal matrices and $\mathbf{\Sigma} \in \mathbb{F}^{\ell \times \ell}$ is a diagonal matrix whose diagonal entries approximate the largest singular values of $\mathbf{B}$.

Algorithm 8 *Randomized singular value decomposition (RSVD).*
Implements the procedure from Section 11.2.

Input: Input matrix $\mathbf{B} \in \mathbb{F}^{m \times n}$, factorization rank ℓ
Output: Orthonormal matrices $\mathbf{U} \in \mathbb{F}^{m \times \ell}$, $\mathbf{V} \in \mathbb{F}^{n \times \ell}$ and a diagonal matrix $\mathbf{\Sigma} \in \mathbb{F}^{\ell \times \ell}$ such that $\mathbf{B} \approx \mathbf{U\Sigma V}^*$.

```
function RSVD(B, ℓ)
    Q = RandomRangefinder(B, ℓ)                    ▷ Algorithm 7
    C = Q*B
    [Û, Σ, V] = svd_econ(C)
    U = QÛ
    [optional] Truncate the factorization to rank r ≤ ℓ
```

Choose a target rank ℓ, and suppose that $\mathbf{Q} \in \mathbb{F}^{m \times \ell}$ is a computed solution to the rangefinder problem. The rangefinder furnishes an approximate rank-ℓ factorization of the input matrix: $\mathbf{B} \approx \mathbf{Q}(\mathbf{Q}^*\mathbf{A})$. To convert this representation into a truncated SVD, we just compute an economy-size SVD of the matrix $\mathbf{C} := \mathbf{Q}^*\mathbf{A} \in \mathbb{F}^{\ell \times n}$ and consolidate the factors.

In symbols, once $\mathbf{Q}$ is available, the computation proceeds as follows:

$$\begin{aligned}
\mathbf{B} &\approx \mathbf{QQ}^*\mathbf{B} && \{\text{matrix–matrix multiplication: } \mathbf{C} = \mathbf{Q}^*\mathbf{B}\} \\
&= \mathbf{QC} && \{\text{economy-size SVD: } \mathbf{C} = \widehat{\mathbf{U}}\mathbf{\Sigma V}^*\} \\
&= \mathbf{Q}\widehat{\mathbf{U}}\mathbf{\Sigma V}^* && \{\text{matrix–matrix multiplication: } \mathbf{U} = \mathbf{Q}\widehat{\mathbf{U}}\} \\
&= \mathbf{U\Sigma V}^*.
\end{aligned}$$

After the rangefinder step, the remaining computations are all exact (modulo floating-point arithmetic errors). Therefore,

$$\|\mathbf{B} - \mathbf{U\Sigma V}^*\| = \|\mathbf{B} - \mathbf{QQ}^*\mathbf{B}\|.$$

In words, the accuracy of the approximate SVD is determined entirely by the error in the rangefinder computation!

Empirically, the smallest singular values and singular vectors of the approximate SVD contribute to the accuracy of the approximation, but they are not good estimates for the true singular values and vectors of the matrix. Therefore, it can be valuable to truncate the rank by zeroing out the smallest computed singular values. We omit the details. See (Halko et al. 2011*a*, Gu 2015) and (Tropp et al. 2019) for further discussion.

Algorithm 8 contains pseudocode for the randomized SVD. In a general setting, the dominant cost after the rangefinder step is the matrix–matrix multiply, which requires $O(mn\ell)$ operations. The storage requirements are $O((m+n)\ell)$ numbers.

The rangefinder primitive allows us to perform other matrix computations as well. For example, in Section 13, we explain how to use the rangefinder to construct matrix factorizations where a subset of the rows/columns are picked to form a basis for the row/column spaces. This approach gives far better results than the more obvious randomized algorithms based on coordinate sampling.

11.3. **The rangefinder and Schur complements.** Why does the randomized rangefinder work? We will demonstrate that the procedure has its most natural expression in the language of Schur complements. This point is implicit in the analysis in Halko et al. (2011*a*), and it occasionally appears more overtly in the literature (e.g., in Gittens (2013) and Tropp, Yurtsever, Udell and Cevher (2017*a*)). Nevertheless, this connection has not been explored in a systematic way.

Proposition 11.1 (Rangefinder: Schur complements)**.** *Let* $\mathbf{Y} = \mathbf{BX}$ *for an arbitrary test matrix* $\mathbf{X} \in \mathbb{F}^{n \times \ell}$*, and let* $\mathbf{P}_\mathbf{Y}$ *be the orthogonal projector onto the range of* $\mathbf{Y}$*. Define the*

approximation error as

$$\mathbf{E} := \mathbf{E}(\mathbf{B}, \mathbf{X}) := (\mathbf{I} - \mathbf{P}_\mathbf{Y})\mathbf{B}.$$

Then the squared error can be written as a Schur complement (2.4)*:*

$$|\mathbf{E}|^2 := \mathbf{E}^*\mathbf{E} = (\mathbf{B}^*\mathbf{B})/\mathbf{X}.$$

We emphasize that $|\mathbf{E}|^2$ *is a psd matrix, not a scalar.*

Proof. This result follows from a short calculation. We can write the orthogonal projector $\mathbf{P}_\mathbf{Y}$ in the form

$$\mathbf{P}_\mathbf{Y} = (\mathbf{BX})((\mathbf{BX})^*(\mathbf{BX}))^\dagger(\mathbf{BX})^*.$$

Abbreviating $\mathbf{A} = \mathbf{B}^*\mathbf{B}$, we have

$$\mathbf{E}^*\mathbf{E} = \mathbf{B}^*(\mathbf{I} - \mathbf{P}_\mathbf{Y})\mathbf{B} = \mathbf{A} - (\mathbf{AX})(\mathbf{X}^*\mathbf{AX})^\dagger(\mathbf{AX})^*.$$

This is precisely the definition (2.4) of the Schur complement $\mathbf{A}/\mathbf{X}$. □

Proposition 11.1 gives us access to the deep theory of Schur complements (Zhang 2005). In particular, we have a beautiful monotonicity property that follows instantly from Ando (2005, Thm. 5.3).

Corollary 11.2 (Monotonicity)**.** *Suppose that* $\mathbf{B}^*\mathbf{B} \preccurlyeq \mathbf{C}^*\mathbf{C}$ *with respect to the semidefinite order* $\preccurlyeq$*. For each fixed test matrix* $\mathbf{X}$*,*

$$\begin{aligned}|\mathbf{E}(\mathbf{B}, \mathbf{X})|^2 &= (\mathbf{B}^*\mathbf{B})/\mathbf{X} \\ &\preccurlyeq (\mathbf{C}^*\mathbf{C})/\mathbf{X} = |\mathbf{E}(\mathbf{C}, \mathbf{X})|^2\end{aligned}$$

In particular, the error increases if we increase any singular value of $\mathbf{B}$ while retaining the same right singular vectors; the error decreases if we decrease any singular value of $\mathbf{B}$. The left singular vectors do not play a role here. This observation allows us to identify which target matrices are hardest to approximate.

Example 11.3 (Extremals)**.** *Consider the parameterized matrix* $\mathbf{B}(\boldsymbol{\sigma}) = \mathbf{U}\,\mathrm{diag}(\boldsymbol{\sigma})\mathbf{V}^* \in \mathbb{F}^{n\times n}$*, where* $\mathbf{U}, \mathbf{V}$ *are unitary. Suppose that we fix* σ_1 *and* σ_{k+1}*. For each test matrix* $\mathbf{X}$*, the error* $|\mathbf{E}(\mathbf{B}(\boldsymbol{\sigma}), \mathbf{X})|^2$ *is maximal in the semidefinite order when*

$$\boldsymbol{\sigma} = (\underbrace{\sigma_1, \dots, \sigma_1}_{k}, \underbrace{\sigma_{k+1}, \dots, \sigma_{k+1}}_{n-k}).$$

It has long been appreciated that Example 11.3 is the hardest matrix to approximate; cf. Martinsson et al. (2006*a*, Sec. 5, Ex. 4, 5). The justification of this insight is new.

11.4. **A priori error bounds.** Proposition 11.1 shows that the error in the rangefinder procedure can be written as a Schur complement. Incredibly, the Schur complement of a psd matrix with respect to a random subspace tends to be quite small. In this section, we summarize a theoretical analysis, due to Halko et al. (2011*a*), that explains why this claim is true.

11.4.1. *Master error bound.* First, we present a deterministic upper bound on the error incurred by the rangefinder procedure. This requires some notation.

Without loss of generality, we may assume that $m = n$ by extending $\mathbf{B}$ with zeros. For any $k \le \ell$, construct a partitioned SVD of the target matrix:

$$\mathbf{B} = \mathbf{U}\begin{bmatrix}\boldsymbol{\Sigma}_1 & \\ & \boldsymbol{\Sigma}_2\end{bmatrix}\begin{bmatrix}\mathbf{V}_1 & \mathbf{V}_2\end{bmatrix}^* \quad \text{with } \boldsymbol{\Sigma}_1 \in \mathbb{R}^{k\times k} \text{ and } \mathbf{V}_1 \in \mathbb{F}^{n\times k}.$$

The factors $\mathbf{U}, \boldsymbol{\Sigma}, \mathbf{V}$ are all square matrices. As usual, the entries of $\boldsymbol{\Sigma} = \mathrm{diag}(\sigma_1, \sigma_2, \dots)$ are arranged in weakly decreasing order. So $\boldsymbol{\Sigma}_1$ lists the first k singular values, and $\boldsymbol{\Sigma}_2$ lists the remaining $n - k$ singular values. The matrix $\mathbf{V}_1$ contains the first k right singular vectors; the matrix $\mathbf{V}_2$ contains the remaining $n - k$ right singular vectors.

For any test matrix $\mathbf{X} \in \mathbb{F}^{n\times\ell}$, define

$$\mathbf{X}_1 = \mathbf{V}_1^*\mathbf{X} \quad \text{and} \quad \mathbf{X}_2 = \mathbf{V}_2^*\mathbf{X}.$$

These matrices reflect the alignment of the test matrix $\mathbf{X}$ with the matrix $\mathbf{V}_1$ of dominant right singular vectors of $\mathbf{B}$. We assume $\mathbf{X}_1$ has full row rank.

With this notation, we can present a strong deterministic bound on the error in Algorithm 7.

Theorem 11.4 (Rangefinder: Deterministic bound)**.** *Let $\mathbf{Y} = \mathbf{B}\mathbf{X}$ be the sample matrix obtained by testing $\mathbf{B}$ with $\mathbf{X}$. With the notation and assumptions above, for all $k \leq \ell$,*

$$\|(\mathbf{I} - \mathbf{P}_{\mathbf{Y}})\mathbf{B}\| \leq \sigma_{k+1} + \|\mathbf{\Sigma}_2\mathbf{X}_2\mathbf{X}_1^\dagger\|. \tag{11.2}$$

A related inequality holds for every quadratic unitarily invariant norm.

The result and its proof are drawn from Halko et al. (2011*a*, Thm. 9.1). The same bound was obtained independently in Boutsidis, Mahoney and Drineas (2009) by means of a different technique.

Theorem 11.4 leads to sharp bounds on the performance of the rangefinder in most situations of practical interest. Let us present a sketch of the argument. Our approach can be modified to obtain matching lower and upper bounds, but they do not give any additional insight into the performance.

Proof. In view of Proposition 11.1, we want to bound the spectral norm of

$$|(\mathbf{I} - \mathbf{P}_{\mathbf{Y}})\mathbf{B}|^2 = (\mathbf{B}^*\mathbf{B})/\mathbf{X}.$$

First, change coordinates so that the right singular vectors of $\mathbf{B}$ are the identity: $\mathbf{V} = \mathbf{I}$. In particular, $\mathbf{B}^*\mathbf{B} = \mathbf{\Sigma}^2$ is diagonal. By homogeneity of (11.2), we may assume that $\sigma_1 = 1$. Next, using Corollary 11.2, we may also assume that $\sigma_1 = \cdots = \sigma_k = 1$, which leads to the worst-case error. Thus, it suffices to bound the spectral norm of the psd matrix

$$\mathbf{S} := \begin{bmatrix} \mathbf{I}_k & \mathbf{0} \\ \mathbf{0} & \mathbf{\Sigma}_2^2 \end{bmatrix} / \mathbf{X}.$$

To accomplish this task, we may as well take the Schur complement of the diagonal matrix with respect to a test matrix that has a smaller range than $\mathbf{X}$; see Ando (2005, Thm. 5.9(iv)). Define $\widetilde{\mathbf{X}} := \mathbf{X}\mathbf{X}_1^\dagger = [\mathbf{I}_k; \mathbf{X}_2\mathbf{X}_1^\dagger]$. Since $\operatorname{range}(\widetilde{\mathbf{X}}) \subseteq \operatorname{range}(\mathbf{X})$,

$$\mathbf{S} \preccurlyeq \begin{bmatrix} \mathbf{I}_k & \mathbf{0} \\ \mathbf{0} & \mathbf{\Sigma}_2^2 \end{bmatrix} / \widetilde{\mathbf{X}} =: \widetilde{\mathbf{S}}.$$

Using the definition of the Schur complement, we can write out the matrix on the right-hand side in block form. With the abbreviation $\mathbf{F} := \mathbf{\Sigma}_2\mathbf{X}_2\mathbf{X}_1^\dagger$,

$$\widetilde{\mathbf{S}} = \begin{bmatrix} \mathbf{I} - (\mathbf{I} + \mathbf{F}\mathbf{F}^*)^{-1} & \star \\ \star & \mathbf{\Sigma}_2^2 - \mathbf{F}(\mathbf{I} + \mathbf{F}\mathbf{F}^*)^{-1}\mathbf{F}^* \end{bmatrix}.$$

The $\star$ symbol denotes matrices that do not play a role in the rest of the argument. We can bound the block matrix above in the psd order:

$$\widetilde{\mathbf{S}} \preccurlyeq \begin{bmatrix} \mathbf{F}\mathbf{F}^* & \star \\ \star & \mathbf{\Sigma}_2^2 \end{bmatrix}$$

The inequality for the top-left block holds because $1 - (1 + a)^{-1} \leq a$ for all numbers $a \geq 0$. Last, take the spectral norm:

$$\|\mathbf{S}\| \leq \|\widetilde{\mathbf{S}}\| \leq \left\| \begin{bmatrix} \mathbf{F}\mathbf{F}^* & \star \\ \star & \mathbf{\Sigma}_2^2 \end{bmatrix} \right\| \leq \|\mathbf{F}\mathbf{F}^*\| + \|\mathbf{\Sigma}_2^2\|.$$

This bound is stronger than the stated result. □

11.4.2. *Gaussian test matrices.* We can obtain precise results for the behavior of the randomized rangefinder when the test matrix is (real) standard normal. Let us present a variant of Halko et al. (2011*a*, Thm. 10.1).

Theorem 11.5 (Rangefinder: Gaussian analysis)**.** *Fix a matrix $\mathbf{B} \in \mathbb{R}^{m \times n}$ with singular values $\sigma_1 \geq \sigma_2 \geq \dots$. Draw a standard normal test matrix $\mathbf{\Omega} \in \mathbb{F}^{n \times \ell}$, and construct the sample matrix $\mathbf{Y} = \mathbf{B\Omega}$. Choose $k < \ell - 1$, and introduce the random variable*

$$Z = \|\mathbf{\Gamma}^\dagger\| \quad \text{where } \mathbf{\Gamma} \in \mathbb{R}^{k \times \ell} \text{ is standard normal.}$$

Then the expected error in the random rangefinder satisfies

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

In other words, the randomized rangefinder computes an ℓ-dimensional subspace that captures as much of the action of the matrix $\mathbf{B}$ as the best k-dimensional subspace. If we think about k as fixed and ℓ as the variable, we only need to choose ℓ slightly larger than k to enjoy this outcome.

The error is comparable with σ_{k+1}, the error in the best rank-k approximation, provided that the tail singular values σ_j for $j > k$ have small ℓ_2 norm. This situation occurs, for example, when $\mathbf{B}$ has a rapidly decaying spectrum.

Proof. Here is a sketch of the argument. Since the test matrix $\mathbf{\Omega}$ is standard normal, the matrices $\mathbf{\Omega}_1 := \mathbf{V}_1^*\mathbf{\Omega}$ and $\mathbf{\Omega}_2 := \mathbf{V}_2^*\mathbf{\Omega}$ are independent standard normal matrices because $\mathbf{V}_1$ and $\mathbf{V}_2$ are orthonormal and mutually orthogonal. Using Chevet's theorem (Halko et al. 2011*a*, Prop. 10.1),

$$\begin{aligned}\mathbb{E}\,\|\mathbf{\Sigma}_2\mathbf{\Omega}_2\mathbf{\Omega}_1^\dagger\| &= \mathbb{E}_{\mathbf{\Omega}_1}\,\mathbb{E}_{\mathbf{\Omega}_2}\left[\|\mathbf{\Sigma}_2\mathbf{\Omega}_2\mathbf{\Omega}_1^\dagger\|\right] \\ &\leq \mathbb{E}\left[\|\mathbf{\Sigma}_2\|\|\mathbf{\Omega}_1^\dagger\|_{\mathrm{F}} + \|\mathbf{\Sigma}_2\|_{\mathrm{F}}\|\mathbf{\Omega}_1^\dagger\|\right] \\ &\leq \sqrt{\frac{k}{\ell - k - 1}}\|\mathbf{\Sigma}_2\| + (\mathbb{E}\,Z)\|\mathbf{\Sigma}_2\|_{\mathrm{F}}.\end{aligned}$$

The last inequality involves a well-known estimate for the trace of an inverted Wishart matrix (Halko et al. 2011*a*, Prop. 10.2). □

To make use of the result, we simply insert estimates for the expectation of the random variable Z. For instance,

$$\mathbb{E}\,Z \leq \frac{\mathrm{e}\sqrt{\ell}}{\ell - k} \quad \text{when } 2 \leq k < \ell \quad \text{and} \quad \mathbb{E}\,Z \approx \frac{1}{\sqrt{\ell} - \sqrt{k}} \quad \text{for } k \ll \ell.$$

These estimates lead to very accurate performance bounds across a wide selection of matrices and parameters.

The rangefinder also operates in the regime $k \in \{\ell - 1, \ell\}$. In this case, it attains significantly larger errors. A heuristic is

$$\|(\mathbf{I} - \mathbf{P_Y})\mathbf{B}\| \lessapprox (1 + k)\sigma_{k+1} + \sqrt{k}\left(\sum\nolimits_{j>k} \sigma_j^2\right)^{1/2}.$$

This point follows because $\|\mathbf{\Omega}_1^\dagger\|_{\mathrm{F}} \approx k$ and $\|\mathbf{\Omega}_1^\dagger\| \approx \sqrt{k}$ when $k \approx \ell$.

For relevant results about Gaussian random matrices, we refer the reader to (Edelman 1989, Davidson and Szarek 2001, Chen and Dongarra 2005, Bai and Silverstein 2010) and (Halko et al. 2011*a*).

11.5. **Other test matrices.** In many cases, it is too expensive to use Gaussian test matrices to implement Algorithm 7. Instead, we may prefer to apply (the adjoint of) one of the structured random embeddings discussed in Section 9.

11.5.1. *Random embeddings for the rangefinder.* Good alternatives to Gaussian test matrices include the following.

- **Sparse maps.** Sparse dimension reduction maps work well in the rangefinder procedure, even if the input matrix is sparse. The primary shortcoming is the need to use sparse data structures and arithmetic. See Section 9.2.
- **SRTTs.** In practice, subsampled randomized trigonometric transforms perform slightly better than Gaussian maps. The main difficulty is that the implementation requires fast trigonometric transforms. See Section 9.3.
- **Tensor product maps.** Emerging evidence suggests that tensor product random projections are also effective in practice. See Section 9.4.

Some authors have proposed using random coordinate sampling to solve the rangefinder problem. We cannot recommend this approach unless it is impossible to use one of the random embeddings described above. See Section 9.6 for a discussion of random coordinate sampling and situations where it may be appropriate.

11.5.2. *Universality.* In practice, if the test matrix is a mixing random embedding, the error in the rangefinder is somewhat insensitive to the precise distribution of the test matrix. In this case, we can use the Gaussian theory to obtain good heuristics about the performance of other types of embeddings. Regardless, we always recommend using *a posteriori* error estimates to validate the performance of the rangefinder method, as well as downstream matrix approximations; see Section 12.

11.5.3. *Aside: Subspace embeddings.* If we merely assume that the test matrix is a subspace embedding, then we can still perform a theoretical analysis of the rangefinder algorithm. Here is a typical result, adapted from Halko et al. (2011*a*, Thm. 11.2).

Theorem 11.6 (Rangefinder: SRTT)**.** *Fix a matrix $\mathbf{B} \in \mathbb{F}^{m \times n}$ with singular values $\sigma_1 \geq \sigma_2 \geq \dots$. Choose a natural number k, and draw an SRTT embedding matrix $\mathbf{\Omega} \in \mathbb{F}^{n \times \ell}$ where*

$$\ell \geq 8(k + 8\log(kn))\log k.$$

Construct the sample matrix $\mathbf{Y} = \mathbf{B\Omega}$. Then

$$\|(\mathbf{I} - \mathbf{P_Y})\mathbf{B}\| \leq (1 + 3\sqrt{n/\ell}) \cdot \sigma_{k+1},$$

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

Proof. (Sketch) By Theorem 11.4,

$$\|(\mathbf{I} - \mathbf{P_Y})\mathbf{B}\| \leq \left[1 + \|\mathbf{\Omega}_2\|\|\mathbf{\Omega}_1^\dagger\|\right] \sigma_{k+1}.$$

With the specified choice of ℓ, the test matrix $\mathbf{\Omega}$ is likely to be an oblivious subspace embedding of the k-dimensional subspace $\mathbf{V}_1$ with distortion $1/3$. Thus, the matrix $\mathbf{\Omega}_1^\dagger$ has spectral norm bounded by 3. The matrix $\sqrt{\ell/n}\,\mathbf{\Omega}$ is orthonormal, so the spectral norm of $\mathbf{\Omega}_2$ is bounded by $\sqrt{n/\ell}$. □

The lower bound in the subspace embedding property (8.5) is the primary fact about the SRTT used in the proof. But this is only part of the reason that the rangefinder works. Accordingly, the outcome of this "soft" analysis is qualitatively weaker than the "hard" analysis in Theorem 11.5. The resulting bound does not explain the actual (excellent) performance of Algorithm 7 when implemented with an SRTT.

11.6. **Subspace iteration.** Theorem 11.5 shows that the basic randomized rangefinder procedure, Algorithm 7, can be effective for target matrices $\mathbf{B}$ with a rapidly decaying spectrum. Nevertheless, in many applications, we encounter matrices that do not meet this criterion. As in the case of spectral norm estimation (Section 6), we can resolve the problem by powering the matrix.

11.6.1. *Rangefinder with powering.* Let $\mathbf{B} \in \mathbb{F}^{m\times n}$ be a fixed input matrix, and let q be a natural number. Let $\mathbf{\Omega} \in \mathbb{F}^{m\times \ell}$ be a random test matrix. We form the sample matrix

$$\mathbf{Y} = (\mathbf{B}\mathbf{B}^*)^q\mathbf{\Omega}$$

by repeated multiplication. Then we compute an orthobasis $\mathbf{Q} \in \mathbb{F}^{m\times\ell}$ for the range of $\mathbf{Y}$ using a QR factorization method.

See Algorithm 9 for pseudocode. In general, the arithmetic cost is dominated by the $O(qmn\ell)$ cost of the matrix–matrix multiplications. Economies are possible when $\mathbf{B}$ admits fast multiplication. Unfortunately, when powering is used, it is not possible to fundamentally accelerate the computation by using a structured test matrix $\mathbf{\Omega}$.

Algorithm 9 coincides with the classic subspace iteration algorithm with a random start. Historically, subspace iteration was regarded as a method for spectral computations. The block size ℓ was often chosen to be quite small, say $\ell = 3$ or $\ell = 4$, because the intention was simply to resolve singular values with multiplicity greater than one.

The randomized NLA literature contains several new insights about the behavior of randomized subspace iteration. It is now recognized that *iteration is not required*. In practice, $q = 2$ or $q = 3$ is entirely adequate to solve the rangefinder problem to fairly high accuracy. The modern perspective also emphasizes the value of running subspace iteration with a very large block size ℓ to obtain matrix approximations.

Remark 11.7 (History)**.** *Rokhlin, Szlam and Tygert (2009) introduced the idea of using randomized subspace iteration to obtain matrix approximations by selecting a large block size ℓ and a small power q. Halko et al. (2011*a*) refactored and simplified the algorithm, and they presented a complete theoretical justification for the approach. Subsequent analysis appears in Gu (2015).*

11.6.2. *Analysis.* The analysis of the powered rangefinder is an easy consequence of the following lemma (Halko et al. 2011*a*, Prop. 8.6).

Lemma 11.8 (Powering)**.** *Let $\mathbf{B} \in \mathbb{F}^{m\times n}$ be a fixed matrix, and let $\mathbf{P} \in \mathbb{F}^{m\times m}$ be an orthogonal projector. For any number $q \geq 1$,*

$$\|(\mathbf{I} - \mathbf{P})\mathbf{B}\|^{2q} \leq \|(\mathbf{I} - \mathbf{P})(\mathbf{B}\mathbf{B}^*)^q\|.$$

Proof. This bound follows immediately from the Araki–Lieb–Thirring inequality Bhatia (1997, Thm. IX.2.10). □

Theorem 11.5 gives us bounds for the right-hand side of the inequality in Lemma 11.8 when $\mathbf{P}$ is the orthogonal projector onto the subspace generated by the powered rangefinder.

Corollary 11.9 (Powered rangefinder: Gaussian analysis)**.** *Under the same conditions as Theorem 11.5, let $\mathbf{Y} = (\mathbf{B}\mathbf{B}^*)^q\mathbf{\Omega}$ be the sample matrix computed by Algorithm 9. Then*

$$\begin{aligned}\mathbb{E}\,\|(\mathbf{I} - \mathbf{P}_\mathbf{Y})\mathbf{B}\| &\leq \left(\mathbb{E}\,\|(\mathbf{I} - \mathbf{P}_\mathbf{Y})\mathbf{B}\|^{2q}\right)^{1/(2q)} \\ &\leq \left[\left(1 + \sqrt{\frac{k}{\ell - k - 1}}\right)\sigma_{k+1}^{2q} + (\mathbb{E}\,Z)\left(\sum\nolimits_{j>k}\sigma_j^{4q}\right)^{1/2}\right]^{1/(2q)}.\end{aligned}$$

In other words, powering the matrix $\mathbf{B}$ drives the error in the rangefinder to σ_{k+1} exponentially fast as the parameter q increases. In cases where the target matrix has some spectral decay, it suffices to take $q = 2$ or $q = 3$ to achieve satisfactory results. For matrices with a flat spectral tail, however, we may need to set $q \approx \log\min\{m, n\}$ to make the error a constant multiple of σ_{k+1}.

11.7. **Block Krylov methods.** As in the case of spectral norm estimation (Section 6), we can achieve more accurate results with Krylov subspace methods. Nevertheless, this improvement comes at the cost of additional storage and more complicated algorithms.

Algorithm 9 *The powered randomized rangefinder.*
Implements the procedure from Section 11.6.

Input: Input matrix $\mathbf{B} \in \mathbb{F}^{m \times n}$, target rank ℓ, depth q.
Output: Orthonormal matrix $\mathbf{Q} \in \mathbb{F}^{m \times \ell}$

```
function PowerRangefinder(B, ℓ, q)
    Draw a random matrix Ω ∈ F^{m×ℓ}
    Y_0 = Ω
    for i = 1, ..., q do
        [Y_{i-1}, ∼] = qr_econ(Y_{i-1})
        Y_i = B(B^* Y_{i-1})
    [Q, ∼] = qr_econ(Y_q)
```

11.7.1. *Rangefinder with a Krylov subspace.* Let $\mathbf{B} \in \mathbb{F}^{m \times n}$ be a fixed input matrix, and let q be a natural number. Let $\mathbf{\Omega} \in \mathbb{F}^{m \times \ell}$ be a random test matrix. We can form the extended sample matrix

$$\mathbf{Y} = \begin{bmatrix} \mathbf{\Omega} & (\mathbf{B}^*\mathbf{B})\mathbf{\Omega} & \dots & (\mathbf{B}^*\mathbf{B})^q\mathbf{\Omega} \end{bmatrix}.$$

Then we compute an orthobasis $\mathbf{Q} \in \mathbb{F}^{m \times (q+1)\ell}$ using a QR factorization method.

See Algorithm 10 for pseudocode. This dominant source of arithmetic is the $O(qmn\ell)$ cost of matrix–matrix multiplication. The QR factorization now requires $O(q^2\ell^2 m)$ operations, a factor of q^2 more than the powered rangefinder. We also need to store $O(qm\ell)$ numbers, which is roughly a factor q more than the powered rangefinder. Nevertheless, there is evidence that we can balance the values of ℓ and q to make the computational cost of the Krylov method comparable with the cost of the power method—and still achieve higher accuracy.

Historically, block Lanczos methods were used for spectral computations and for SVD computations. The block size ℓ was typically chosen to be fairly small, say $\ell = 3$ or $\ell = 4$, with the goal of resolving singular values with multiplicity greater than one. The depth q of the iteration was usually chosen to be quite large. There is theoretical and empirical evidence that this parameter regime is the most efficient for resolving the largest singular values to high accuracy (Yuan, Gu and Li 2018).

The randomized NLA literature has recognized that there are still potential advantages to choosing the block size ℓ to be very large and to choose the depth q to be quite small (Halko et al. 2011*b*, Musco and Musco 2015). This parameter regime leads to algorithms that are more efficient on modern computer architectures, and it still works extremely well for matrices with a modest amount of spectral decay (Tropp 2018). The contemporary literature also places a greater emphasis on the role of block Krylov methods for computing matrix approximations.

Remark 11.10 (History)**.** *Block Lanczos methods, which are an efficient implementation of the block Krylov method for a symmetric matrix, were proposed by Cullum and Donath (1974) and by Golub and Underwood (1977). The extension to the rectangular case appears in Golub, Luk and Overton (1981). These algorithms have received renewed attention, beginning with (Halko et al. 2011*b*). The theoretical analysis of randomized block Krylov methods is much more difficult than the analysis of randomized subspace iteration. See (Musco and Musco 2015, Yuan et al. 2018) and (Tropp 2018) for some results.*

11.7.2. *Alternative bases.* Algorithm 10 computes a monomial basis for the Krylov subspace, which is a poor choice numerically. Let us mention two more practical alternatives.

(1) **Block Lanczos.** The classical approach uses the block Lanczos iteration with reorthogonalization to compute a Lanczos-type basis for the Krylov subspace. Algorithm 11 contains pseudocode for this approach adapted from Golub et al. (1981). The basis computed by this algorithm has the property that the blocks $\mathbf{Q}_i \in \mathbb{F}^{m \times \ell}$ of the sample matrix are mutually orthogonal. The cost is similar to the cost of computing the monomial basis, but there are advantages when the subspace is used for

Algorithm 10 *The Krylov randomized rangefinder.*
Implements the procedure from Section 11.7.
Use with caution! Unreliable in floating-point arithmetic.

Input: Input matrix $\mathbf{B} \in \mathbb{F}^{m \times n}$, target rank ℓ, depth q
Output: Orthonormal matrix $\mathbf{Q} \in \mathbb{F}^{m \times 2(q+1)\ell}$

```
function KrylovRangefinder(B, ℓ, q)
    Draw a random matrix Ω ∈ F^{n×ℓ}
    [Y_0, ∼] = qr_econ(Ω)
    for i = 1, ..., q do
        [Y_{i−1}, ∼] = qr_econ(Y_{i−1})
        Y_i = B(B^* Y_{i−1})
    [Q, ∼] = qr_econ([Y_0, ..., Y_q])
```

spectral computations (Section 6). Indeed, we can approximate the largest singular values of $\mathbf{A}$ by means of the largest singular values of the band matrix

$$\mathbf{R} = \begin{bmatrix} \mathbf{R}_1 & \mathbf{R}_2^* & & & & \\ & \mathbf{R}_3 & \mathbf{R}_4^* & & & \\ & & \ddots & \ddots & & \\ & & & \mathbf{R}_{2q-1} & \mathbf{R}_{2q}^* & \\ & & & & \mathbf{R}_{2q+1} & \end{bmatrix}.$$

(The notation in this paragraph corresponds to the quantities computed in Algorithm 11.)

(2) **Chebyshev.** Suppose that we have a good upper bound for the spectral norm of the target matrix. (For example, we can obtain one using techniques from Section 6.) Then we can compute a Chebyshev basis for the Krylov subspace. The advantage of this approach is that we can postpone all normalization and orthogonalization steps to the end of the computation, which is beneficial for distributed computation. This approach, which is being presented for the first time, is inspired by Joubert and Carey (1991). See Algorithm 12 for pseudocode.

11.7.3. *Analysis.* We are not aware of a direct analysis of Krylov subspace methods for solving the rangefinder problem. One may extract some bounds from analysis of randomized SVD algorithms. The following result is adapted from Tropp (2018).

Theorem 11.11 (Krylov rangefinder: Gaussian analysis)**.** *Under the conditions in Theorem 11.5, let* $\mathbf{Y}$ *be the sample matrix computed by Algorithm 10. For* $0 \le \varepsilon \le 1/2$*,*

$$\mathbb{E}\,\|(\mathbf{I} - \mathbf{P}_{\mathbf{Y}})\mathbf{B}\|^2 \le \left[1 + 2\varepsilon + \frac{9nk(\ell - k)}{\ell - k - 2} \cdot \mathrm{e}^{-4q\sqrt{\varepsilon}}\right] \sigma_{k+1}^2.$$

In other words, the Krylov method can drive the error bound for the rangefinder to $O(\varepsilon)$ by using a Krylov subspace with depth $q \approx \log(n/\varepsilon)/\sqrt{\varepsilon}$. In contrast, the power method needs about $q \approx (\log n)/\varepsilon$ iterations to reach the same target. The difference can be very substantial when ε is small.

11.7.4. *Spectral computations.* The Krylov rangefinder can also be used for highly accurate computation of the singular values of a general matrix and the eigenvalues of a self adjoint matrix. See (Musco and Musco 2015, Yuan et al. 2018) and (Tropp 2018) for discussion and analysis.

12. Error estimation and adaptivity

The theoretical analysis of the randomized rangefinder in Section 11 describes with great precision when the procedure is effective, and what errors to expect. From a practical point

Algorithm 11 *The Lanczos randomized rangefinder.*
Implements the Krylov rangefinder (Section 11.7) with block Lanczos bidiagonalization.

Input: Input matrix $\mathbf{B} \in \mathbb{F}^{m \times n}$, target rank ℓ, depth q
Output: Orthonormal matrix $\mathbf{Q} \in \mathbb{F}^{m \times (q+1)\ell}$

function LANCZOSRANGEFINDER($\mathbf{B}$, ℓ, q)
  Draw a random matrix $\mathbf{\Omega} \in \mathbb{F}^{m \times \ell}$
  $[\mathbf{Q}_0, \sim] = \texttt{qr\_econ}(\mathbf{\Omega})$
  $\mathbf{W}_0 = \mathbf{B}\mathbf{Q}_0$
  $[\mathbf{P}_0, \mathbf{R}_1] = \texttt{qr\_econ}(\mathbf{W}_0)$
  for $i = 1, \dots, q$ **do**
    $\mathbf{Z}_i = \mathbf{B}^*\mathbf{P}_{i-1} - \mathbf{Q}_{i-1}\mathbf{R}_{2i-1}^*$ ▷ Lanczos recursion, part 1
    for $j = 0, \dots, i-1$ **do** ▷ Double Gram–Schmidt
      $\mathbf{Z}_i = \mathbf{Z}_i - \mathbf{Q}_j(\mathbf{Q}_j^*\mathbf{Z}_i)$
      $\mathbf{Z}_i = \mathbf{Z}_i - \mathbf{Q}_j(\mathbf{Q}_j^*\mathbf{Z}_i)$
    $[\mathbf{Q}_i, \mathbf{R}_{2i}] = \texttt{qr\_econ}(\mathbf{Z}_i)$
    $\mathbf{W}_i = \mathbf{B}\mathbf{Q}_i - \mathbf{P}_{i-1}\mathbf{R}_{2i}^*$ ▷ Lanczos recursion, part 2
    for $j = 0, \dots, i-1$ **do** ▷ Double Gram–Schmidt
      $\mathbf{W}_i = \mathbf{W}_i - \mathbf{P}_j(\mathbf{P}_j^*\mathbf{W}_i)$
      $\mathbf{W}_i = \mathbf{W}_i - \mathbf{P}_j(\mathbf{P}_j^*\mathbf{W}_i)$
    $[\mathbf{P}_i, \mathbf{R}_{2i+1}] = \texttt{qr\_econ}(\mathbf{W}_i)$
  $\mathbf{Q} = [\mathbf{Q}_0, \dots, \mathbf{Q}_q]$

Algorithm 12 *The Chebyshev randomized rangefinder.*
Implements the Krylov rangefinder (Section 11.7) with a Chebyshev basis.

Input: Input matrix $\mathbf{B} \in \mathbb{F}^{m \times n}$, target rank ℓ, depth q, and norm bound $\|\mathbf{B}\| \le \nu$
Output: Orthonormal matrix $\mathbf{Q} \in \mathbb{F}^{m \times (q+1)\ell}$

function CHEBYSHEVRANGEFINDER($\mathbf{B}$, ℓ, q)
  Draw a random matrix $\mathbf{\Omega} \in \mathbb{F}^{m \times \ell}$
  $[\mathbf{Y}_0, \sim] = \texttt{qr\_econ}(\mathbf{\Omega})$
  $\mathbf{Y}_1 = (2/\nu)\mathbf{B}(\mathbf{B}^*\mathbf{Y}_0) - \mathbf{Y}_0$
  for $i = 2, \dots, q$ **do**
    $\mathbf{Y}_i = (4/\nu)\mathbf{B}(\mathbf{B}^*\mathbf{Y}_{i-1}) - 2\mathbf{Y}_{i-1} - \mathbf{Y}_{i-2}$ ▷ Chebyshev recursion
  $[\mathbf{Q}, \sim] = \texttt{qr\_econ}([\mathbf{Y}_0, \dots, \mathbf{Y}_q])$

of view, however, the usefulness of this analysis is limited by the fact that we rarely have advance knowledge of the singular values of the matrix to be approximated. In this section, we consider the more typical situation where we are given a matrix $\mathbf{A} \in \mathbb{F}^{m \times n}$ and a tolerance ε, and it is our job to find a low-rank factorization of $\mathbf{A}$ that is accurate to within precision ε. In other words, part of the task to be solved is to determine the ε-rank of $\mathbf{A}$.

To complete this job, we must equip the rangefinder with an *a posteriori* error estimator: given an orthonormal matrix $\mathbf{Q} \in \mathbb{F}^{m \times \ell}$ whose columns form a putative basis for the range of $\mathbf{A}$, it estimates the corresponding approximation error $\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|$.

To solve the fixed-error approximation problem, the idea is to start with a lowball "guess" at the rank, run the rangefinder, and then check to see if we are within the requested tolerance. If the answer is negative, then there are several strategies for how to proceed. Typically, we would just draw more samples to enrich the basis we already have on hand. But in some circumstances, it may be better to start over or to try an alternative approach, such as increasing the amount of powering that is done.

12.1. **A posteriori error estimation.** Let $\mathbf{A} \in \mathbb{F}^{m \times n}$ be a target matrix, and let $\mathbf{Q} \in \mathbb{F}^{m \times \ell}$ be an orthonormal matrix whose columns may or may not form a good basis for the range of $\mathbf{A}$. We typically think of $\mathbf{Q}$ as being the output of one of the rangefinder algorithms described in Section 11. Our goal is now to produce an inexpensive and reliable estimate of the error $\|(\mathbf{I} - \mathbf{Q}\mathbf{Q}^*)\mathbf{A}\|$ with respect to some norm $\|\cdot\|$.

To do so, we draw on the techniques for norm estimation by sampling (Sections 4–5). The basic idea is to collect a (small) auxiliary sample

$$\mathbf{Z} = \mathbf{A}\mathbf{\Phi}, \tag{12.1}$$

where the test matrix $\mathbf{\Phi} \in \mathbb{F}^{n \times s}$ is drawn from a Gaussian distribution. We assume that $\mathbf{\Phi}$ is statistically independent from whatever process was used to compute $\mathbf{Q}$. Then

$$(\mathbf{I} - \mathbf{Q}\mathbf{Q}^*)\mathbf{Z} = (\mathbf{I} - \mathbf{Q}\mathbf{Q}^*)\mathbf{A}\mathbf{\Phi} \in \mathbb{F}^{m \times s}. \tag{12.2}$$

is a random sample of the error in the approximation. We can now use any of the methods from Sections 4–5 to estimate the error from this sample. For example,

$$\|(\mathbf{I} - \mathbf{Q}\mathbf{Q}^*)\mathbf{A}\|_{\mathrm{F}}^2 \approx \frac{1}{s}\|(\mathbf{I} - \mathbf{Q}\mathbf{Q}^*)\mathbf{Z}\|_{\mathrm{F}}^2.$$

The theory in Section 4 gives precise tail bounds for this estimator. Similarly, we can approximate the Schatten 4-norm by computing a sample variance. Beyond that, the Valiant–Kong estimator (Section 5.4) allows us to approximate higher-order Schatten norms.

The cost of extracting the auxiliary sample (12.1) is almost always much smaller than the cost of running the rangefinder itself; it involves only s additional matrix–vector multiplications, where s can be thought of as a small fixed number, say $s = 10$.

12.2. **A certificate of accuracy for structured random matrices.** The idea of drawing an auxiliary sample (12.1) for purposes of error estimation is particularly appealing when the rangefinder implementation involves a structured random test matrix (see Section 11.5). These structured random maps can be much faster than Gaussian random matrices, while producing errors that are just as small (Halko et al. 2011*a*, Sec. 7.4). Their main weakness is that they come with far weaker *a priori* error guarantees; see Theorem 11.6.

Now, consider a situation where we use a structured random matrix to compute an approximate basis $\mathbf{Q}$ for the range of a matrix $\mathbf{A}$ (via Algorithm 7), and a small Gaussian random matrix $\mathbf{\Phi}$ to draw an auxiliary sample (12.1) from $\mathbf{A}$. The additional cost of extracting the "extra" sample is small, both in terms of practical execution time and in terms of the asymptotic cost estimates (which would generally remain unchanged). However, the computed output can now be relied on with supreme confidence, since it is backed up by the strong theoretical results that govern Gaussian matrices.

One may even push these ideas further and use the "certificate of accuracy" to gain confidence in randomized sampling methods based on heuristics or educated guesses about the matrix being estimated. For instance, one may observe that matrices that arise in some application typically have low coherence (Section 9.6), and one may implement a fast uniform sampling strategy that only works in this setting. No matter how unsafe the sampling strategy, we can trust the *a posteriori* error estimator when it promises that the computed factorization is sufficiently accurate.

The general idea of observing the action of a residual on random vectors in order to get an estimate for its magnitude can be traced back at least as far as Girard (1989). Here, we followed the discussion in Martinsson (2018, Sec. 14) and Tropp et al. (2019, Sec. 6).

Remark 12.1 (Rank doubling)**.** *Let us consider what should be done if the a posteriori error estimator tells us that a requested tolerance has not yet been met. Since many structured random matrices have the unfortunate property that it is not an easy matter to recycle the sample already computed in order to build a larger one, it often makes sense to simply start from scratch, but doubling the number of columns in the test matrix. Such a strategy of doubling the rank at each attempt typically does not change the order of the dominant term in the asymptotic cost. It may, however, be somewhat wasteful from a practical point of view.*

12.3. **Adaptive rank determination using Gaussian test matrices.** When a Gaussian test matrix is used, we can incorporate *a posteriori* error estimation into the rangefinder algorithm with negligible increase in the amount of computation.

To illustrate, let us first consider a situation where we are given a matrix $\mathbf{A}$, and we use the randomized rangefinder (Algorithm 7) with a Gaussian random matrix to find an orthonormal basis for its approximate range. We do not know the numerical rank of $\mathbf{A}$ in advance, so we use a number ℓ of samples that we believe is likely to be more than enough.

In order to include *a posteriori* error estimation, we *conceptually* split the test matrix so that

$$\mathbf{\Omega} = \begin{bmatrix} \mathbf{\Omega}_1 & \mathbf{\Omega}_2 \end{bmatrix}, \tag{12.3}$$

where $\mathbf{\Omega}_1$ holds the first $\ell - s$ columns of $\mathbf{\Omega}$. The thin sliver $\mathbf{\Omega}_2$ that holds the last s columns will temporarily play the part of the independent test matrix $\mathbf{\Phi}$. (Note that the matrices $\mathbf{\Omega}_i$ defined here are different from those in the proof of Theorem 11.5.)

The sample matrix inherits a corresponding split

$$\mathbf{Y} = \mathbf{A\Omega} = \begin{bmatrix} \mathbf{A\Omega}_1 & \mathbf{A\Omega}_2 \end{bmatrix} =: \begin{bmatrix} \mathbf{Y}_1 & \mathbf{Y}_2 \end{bmatrix}. \tag{12.4}$$

In order to orthonormalize the columns of $\mathbf{Y}$ to form an approximate basis for the column space, we perform an unpivoted QR factorization:

$$\mathbf{Y} = \begin{bmatrix} \mathbf{Q}_1 & \mathbf{Q}_2 \end{bmatrix} \begin{bmatrix} \mathbf{R}_{11} & \mathbf{R}_{12} \\ \mathbf{0} & \mathbf{R}_{22} \end{bmatrix},$$

where the partitioning conforms with (12.4). Thus, $\mathbf{Y}_1 = \mathbf{Q}_1\mathbf{R}_{11}$ and $\mathbf{Y}_2 = \mathbf{Q}_1\mathbf{R}_{12} + \mathbf{Q}_2\mathbf{R}_{22}$. Now, observe that

$$\mathbf{Q}_2\mathbf{R}_{22} = \mathbf{Y}_2 - \mathbf{Q}_1\mathbf{R}_{12} = \mathbf{Y}_2 - \mathbf{Q}_1\mathbf{Q}_1^*\mathbf{Y}_2 = \big(\mathbf{I} - \mathbf{Q}_1\mathbf{Q}_1^*\big)\mathbf{A\Omega}_2.$$

In other words, the matrix $\mathbf{Q}_2\mathbf{R}_{22}$ is a sample of the residual error resulting from using $\mathbf{\Omega}_1$ as the test matrix. We can analyze the sample $\mathbf{Q}_2\mathbf{R}_{22}$ using the techniques described in Section 12.1 to derive an estimate on the norm of the residual error. If the resulting estimate is small enough, we can confidently trust the computed factorization. (When the rangefinder is used as a preliminary step towards computing a partial SVD of the matrix, we may as well use the full orthonormal basis in $\mathbf{Q}$ in any steps that follow.)

If the error estimate computed is larger than what is acceptable, then additional work must be done, typically by drawing additional samples to enrich the basis already computed, as described in the next section.

12.4. **An incremental algorithm based on Gaussian test matrices.** The basic error estimation procedure outlined in Section 12.3 is appropriate when it is not onerous to draw a large number ℓ of samples and when the *a posteriori* error merely serves as an insurance policy against a rare situation where the singular values decay more slowly than expected. In this section, we describe a technique that is designed for situations where we have no notion what the rank may be in advance. The idea is to build the approximate basis incrementally by drawing and processing one batch of samples at a time, while monitoring the errors as we go. The upshot is that this computation can be organized in such a way that the total cost is essentially the same as it would have been had we known the numerical rank in advance.

We frame the rangefinder problem as usual: for a given matrix $\mathbf{A}$ and a given tolerance τ, we seek to build an orthonormal matrix $\mathbf{Q}$ such that

$$\|\mathbf{A} - \mathbf{QQ}^*\mathbf{A}\| \leq \tau.$$

The procedure that we describe will be controlled by a tuning parameter b that specifies how many columns we process at a time. If we choose b too large, we may overshoot the numerical rank of $\mathbf{A}$ and perform more work than necessary. If b is too small, computational efficiency may suffer; see Section 16.2. In many environments, picking b between 10 and 100 would be about right.

While the *a posteriori* error estimator signals that the error tolerance has not been met, the incremental rangefinder successively draws blocks of b Gaussian random vectors, computes

Algorithm 13 *Incremental rangefinder.*
Implements the first procedure from Section 12.4.
This technique builds an orthonormal basis for a given matrix by processing blocks of vectors at a time. The method stops when an *a posteriori* error estimator indicates that a specified tolerance has been met.

Input: Target matrix $\mathbf{A} \in \mathbb{F}^{m \times n}$, tolerance $\tau \in \mathbb{R}_+$, block size b
Output: Orthonormal matrix $\mathbf{Q}$ such that $\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\| \leq \tau$ with high probability

1 **function** INCREMENTALRANGEFINDER($\mathbf{A}$, τ, b)
2 $\quad \mathbf{Y} = \mathbf{A}\boldsymbol{\Omega}$ ▷ Draw $\boldsymbol{\Omega} \in \mathbb{F}^{n \times b}$ from a Gaussian distribution
3 $\quad [\mathbf{Q}_1, \sim] = \texttt{qr\_econ}(\mathbf{Y})$
4 $\quad i = 1$
5 $\quad$ **while** $\texttt{norm\_est}(\mathbf{Y}) > \tau$ **do** ▷ Norm estimator in Section 12.1
6 $\quad\quad i = i + 1$
7 $\quad\quad \mathbf{Y} = \mathbf{A}\boldsymbol{\Omega}$ ▷ Draw $\boldsymbol{\Omega} \in \mathbb{F}^{n \times b}$ from a Gaussian distribution
8 $\quad\quad \mathbf{Y} = \mathbf{Y} - \sum_{j=1}^{i-1} \mathbf{Q}_j (\mathbf{Q}_j^* \mathbf{Y})$
9 $\quad\quad [\mathbf{Q}_i, \sim] = \texttt{qr\_econ}(\mathbf{Y})$
10 $\quad \mathbf{Q} = \begin{bmatrix} \mathbf{Q}_1 & \mathbf{Q}_2 & \cdots & \mathbf{Q}_{i-1} \end{bmatrix}$

the corresponding samples, and adds them to the basis. See Algorithm 13. To understand how the method works, observe that, after line 8 has been executed, the matrix $\mathbf{Y}$ holds the sample

$$\mathbf{Y} = \big(\mathbf{I} - \mathbf{Q}\mathbf{Q}^*\big)\mathbf{A}\boldsymbol{\Omega} \tag{12.5}$$

from the residual $\big(\mathbf{I} - \mathbf{Q}\mathbf{Q}^*\big)\mathbf{A}$, where $\mathbf{Q}$ is the cumulative basis that has been built at that point and where $\boldsymbol{\Omega}$ is an $n \times b$ matrix drawn from a Gaussian distribution. Since (12.5) holds, we can estimate $\|\big(\mathbf{I} - \mathbf{Q}\mathbf{Q}^*\big)\mathbf{A}\|$ using the techniques described in Section 12.1.

In situations where the matrix $\mathbf{A}$ is small enough to fit in RAM, it often makes sense to explicitly update it after every step. The benefit to doing so is that one can then determine the norm of the remainder matrix explicitly. Algorithm 14 summarizes the resulting procedure. After line 9 of the algorithm has been executed, the formula (12.5) holds because, at this point in the computation, $\mathbf{A}$ has been overwritten by $\big(\mathbf{I} - \mathbf{Q}\mathbf{Q}^*\big)\mathbf{A}$. Algorithm 14 relates to Algorithm 13 in the same way that modified Gram–Schmidt relates to classical Gram–Schmidt.

For matrices whose singular values decay slowly, incorporating a few steps of power iteration (as described in Section 11.6) is very beneficial. In this environment, it is often necessary to incorporate additional reorthonormalizations to combat loss of orthogonality due to round-off errors. Full descriptions of the resulting techniques can be found in Martinsson and Voronin (2016).

A version of Algorithm 14 suitable for sparse matrices or matrices stored out-of-core is described in Yu, Gu, Li, Liu and Li (2017*b*). This variant reorganizes the computation to avoid the explicit updating step and to reduce the communication requirements overall. Observe that it is still possible to evaluate the Frobenius norm of the residual exactly (without randomized estimation) by using the identity

$$\begin{aligned} \|\mathbf{A}\|_{\mathrm{F}}^2 &= \|\big(\mathbf{I} - \mathbf{Q}\mathbf{Q}^*\big)\mathbf{A}\|_{\mathrm{F}}^2 + \|\mathbf{Q}\mathbf{Q}^*\mathbf{A}\|_{\mathrm{F}}^2 \\ &= \|\big(\mathbf{I} - \mathbf{Q}\mathbf{Q}^*\big)\mathbf{A}\|_{\mathrm{F}}^2 + \|\mathbf{Q}\mathbf{B}\|_{\mathrm{F}}^2 = \|\big(\mathbf{I} - \mathbf{Q}\mathbf{Q}^*\big)\mathbf{A}\|_{\mathrm{F}}^2 + \|\mathbf{B}\|_{\mathrm{F}}^2, \end{aligned}$$

where the first equality holds since the column spaces of $\big(\mathbf{I} - \mathbf{Q}\mathbf{Q}^*\big)\mathbf{A}$ and $\mathbf{Q}\mathbf{Q}^*\mathbf{A}$ are orthogonal and where the third equality holds since $\mathbf{Q}$ is orthonormal. For an error measure, we can use the resulting relationship:

$$\|\big(\mathbf{I} - \mathbf{Q}\mathbf{Q}^*\big)\mathbf{A}\|_{\mathrm{F}} = \sqrt{\|\mathbf{A}\|_{\mathrm{F}}^2 - \|\mathbf{B}\|_{\mathrm{F}}^2},$$

observing that both $\|\mathbf{A}\|_{\mathrm{F}}$ and $\|\mathbf{B}\|_{\mathrm{F}}$ can be computed explicitly.

Algorithm 14 *Incremental rangefinder with updating.*
Implements the second procedure from Section 12.4.
This variation of Algorithm 13 is suitable when the given matrix $\mathbf{A}$ is small enough that it can be updated explicitly. The method builds an approximate factorization $\mathbf{A} \approx \mathbf{QB}$ that is *guaranteed* to satisfy $\|\mathbf{A} - \mathbf{QB}\| \le \tau$ for a requested tolerance τ.

Input: Target matrix $\mathbf{A} \in \mathbb{F}^{m \times n}$, tolerance $\tau \in \mathbb{R}_+$, block size b.
Output: Orthonormal matrix $\mathbf{Q}$ and a matrix $\mathbf{B}$ such that $\|\mathbf{A} - \mathbf{QB}\| \le \tau$.

function IncrementalRangefinderWithUpdating($\mathbf{A}$, τ, b)
  $\mathbf{Y} = \mathbf{A\Omega}$ ▷ Draw $\mathbf{\Omega} \in \mathbb{F}^{n \times b}$ from a Gaussian distribution.
  $[\mathbf{Q}_1, \sim] = \texttt{qr\_econ}(\mathbf{Y})$
  $\mathbf{B}_1 = \mathbf{Q}_1^* \mathbf{A}$
  $\mathbf{A} = \mathbf{A} - \mathbf{Q}_1 \mathbf{B}_1$
  $i = 1$
  while $\|\mathbf{A}\| > \tau$ **do** ▷ Use an inexpensive norm such as Frobenius
    $i = i + 1$
    $\mathbf{Y} = \mathbf{A\Omega}$ ▷ Draw $\mathbf{\Omega} \in \mathbb{F}^{n \times b}$ from a Gaussian distribution.
    $[\mathbf{Q}_i, \sim] = \texttt{qr\_econ}(\mathbf{Y})$
    $\mathbf{B}_i = \mathbf{Q}_i^* \mathbf{A}$
    $\mathbf{A} = \mathbf{A} - \mathbf{Q}_i \mathbf{B}_i$
  $\mathbf{Q} = \begin{bmatrix} \mathbf{Q}_1 & \mathbf{Q}_2 & \cdots & \mathbf{Q}_i \end{bmatrix}$
  $\mathbf{B} = \begin{bmatrix} \mathbf{B}_1^* & \mathbf{B}_2^* & \cdots & \mathbf{B}_i^* \end{bmatrix}^*$

13. Finding natural bases: QR, ID, and CUR

In Section 11, we explored a number of efficient techniques for building a tall thin matrix $\mathbf{Q}$ whose columns form an approximate basis for the range of an input matrix $\mathbf{A}$ that is numerically rank-deficient. The columns of $\mathbf{Q}$ are orthonormal, and they are formed as linear combinations of many columns from the matrix $\mathbf{A}$.

It is sometimes desirable to work with a basis for the range that consists of a subset of the columns of $\mathbf{A}$ itself. In this case, one typically has to give up on the requirement that the basis vectors be orthogonal. We gain the advantage of a basis that shares properties with the original matrix, such as sparsity or nonnegativity. Moreover, for purposes of data interpretation and analysis, it can be very useful to identify a subset of the columns that distills the information in the matrix.

In this section, we start by describing some popular matrix decompositions that use "natural" basis vectors for the column space, for the row space, or for both. We show how these matrices can be computed somewhat efficiently by means of slight modifications to classical deterministic techniques. Then we describe how to combine deterministic and randomized methods to obtain algorithms with superior performance.

13.1. **The CUR decomposition, and three flavors of interpolative decompositions.** To introduce the low-rank factorizations that we investigate in this section, we describe how they can be used to represent an $m \times n$ matrix $\mathbf{A}$ of *exact* rank k, where $k < \min(m, n)$. This is an artificial setting, but it allows us to convey the key ideas using a minimum of notational overhead.

A basic interpolative decomposition (ID) of a matrix $\mathbf{A}$ with exact rank k takes the form

$$\underset{m \times n}{\mathbf{A}} = \underset{m \times k}{\mathbf{C}} \; \underset{k \times n}{\mathbf{Z}}, \tag{13.1}$$

where the matrix $\mathbf{C}$ is given by a subset of the columns of $\mathbf{A}$ and where $\mathbf{Z}$ is a matrix that contains the $k \times k$ identity matrix as a submatrix. The fact that the decomposition (13.1) exists is an immediate consequence of the definition of rank. A more significant observation is that there exists a factorization of the form (13.1) that is *well-conditioned*, in the sense

that no entry of $\mathbf{Z}$ is larger than one in modulus. This claim can be established through an application of Cramer's rule. See (Goreinov, Zamarashkin and Tyrtyshnikov 1997) and (Martinsson, Rokhlin and Tygert 2006*b*).

The factorization (13.1) uses a subset of the columns of $\mathbf{A}$ to span its column space. Of course, there is an analog factorization that uses a subset of the rows of $\mathbf{A}$ to span the row space. We write this as

$$\underset{m\times n}{\mathbf{A}} = \underset{m\times k}{\mathbf{X}} \; \underset{k\times n}{\mathbf{R}}, \tag{13.2}$$

where $\mathbf{R}$ is a matrix consisting of k rows of $\mathbf{A}$, and where $\mathbf{X}$ is a matrix that contains the $k \times k$ identity matrix.

For bookkeeping purposes, we introduce index vectors J_{s} and I_{s} that identify the columns and rows chosen in the factorizations (13.1) and (13.2). To be precise, let $J_{\mathrm{s}} \subset \{1, 2, \ldots, n\}$ denote the index vector of length k such that

$$\mathbf{C} = \mathbf{A}(:, J_{\mathrm{s}}).$$

Analogously, we let $I_{\mathrm{s}} \subset \{1, 2, \ldots, m\}$ denote the index vector for which

$$\mathbf{R} = \mathbf{A}(I_{\mathrm{s}}, :).$$

The index vectors I_{s} and J_{s} are often referred to as *skeleton* index vectors, whence the subscript "s". This terminology arises from the original literature about these factorizations (Goreinov et al. 1997).

A related two-sided factorization is based on extracting a row/column submatrix. In this case, the basis vectors for the row and column space are less interpretable. More precisely,

$$\underset{m\times n}{\mathbf{A}} = \underset{m\times k}{\mathbf{X}} \; \underset{k\times k}{\mathbf{A}_{\mathrm{s}}} \; \underset{k\times n}{\mathbf{Z}}, \tag{13.3}$$

where $\mathbf{X}$ and $\mathbf{Z}$ are the same matrices as those that appear in (13.1) and (13.2), and where $\mathbf{A}_{\mathrm{s}}$ is the $k \times k$ submatrix of $\mathbf{A}$ given by

$$\mathbf{A}_{\mathrm{s}} = \mathbf{A}(I_{\mathrm{s}}, J_{\mathrm{s}}).$$

To distinguish among these variants, we refer to (13.1) as a *column ID*, to (13.2) as a *row ID*, and to (13.3) as a *double-sided ID*.

We introduce a fourth factorization, often called the *CUR decomposition*. For a matrix of exact rank k, it takes the form

$$\underset{m\times n}{\mathbf{A}} = \underset{m\times k}{\mathbf{C}} \; \underset{k\times k}{\mathbf{U}} \; \underset{k\times n}{\mathbf{R}}, \tag{13.4}$$

where $\mathbf{C}$ and $\mathbf{R}$ are the matrices that appeared in (13.1) and (13.2), which consist of k columns and k rows of $\mathbf{A}$, and where $\mathbf{U}$ is a small matrix that links them together. In the present case, where $\mathbf{A}$ has exact rank k, the matrix $\mathbf{U}$ must take the form

$$\mathbf{U} = \big(\mathbf{A}(I_{\mathrm{s}}, J_{\mathrm{s}})\big)^{-1}. \tag{13.5}$$

The factorizations (13.3) and (13.4) are related through the formula

$$\mathbf{A} = \mathbf{X}\mathbf{A}_{\mathrm{s}}\mathbf{Z} = \underbrace{\big(\mathbf{X}\mathbf{A}_{\mathrm{s}}\big)}_{=\mathbf{C}} \underbrace{\mathbf{A}_{\mathrm{s}}^{-1}}_{=\mathbf{U}} \underbrace{\big(\mathbf{A}_{\mathrm{s}}\mathbf{Z}\big)}_{=\mathbf{R}}.$$

Comparing the two formats, we see that the CUR (13.4) has an advantage in that it requires very little storage. As long as $\mathbf{A}$ is stored explicitly (or is easy to retrieve), the CUR factorization (13.4) is determined by the index vectors I_{s} and J_{s} and the linking matrix $\mathbf{U}$. If $\mathbf{A}$ is not readily available, then, in order to use the CUR, we need to evaluate and store the matrices $\mathbf{C}$ and $\mathbf{R}$. When $\mathbf{A}$ is sparse, the latter approach can still be more efficient than storing the matrices $\mathbf{X}$ and $\mathbf{Z}$.

A disadvantage of the CUR factorization (13.4) is that, when the singular values of $\mathbf{A}$ decay rapidly, the factorization (13.4) is typically numerically ill-conditioned. The reason is that, whenever the factorization is a good representation of $\mathbf{A}$, the singular values of $\mathbf{A}_{\mathrm{s}}$ should approximate the k dominant singular values of $\mathbf{A}$, so the singular values of $\mathbf{U}$ end up

approximating the *inverses* of these singular values. This means that $\mathbf{U}$ will have elements of magnitude $1/\sigma_k$, which is clearly undesirable when σ_k is small. In contrast, the ID (13.3) is numerically benign.

In the numerical analysis literature, what we refer to as an interpolative decomposition is often called a *skeleton factorization* of $\mathbf{A}$. This term dates back at least as far as Goreinov et al. (1997), where the term *pseudo-skeleton* was used for the CUR decomposition (13.4).

Remark 13.1 (Storage-efficient ID)**.** *We mentioned that the matrices $\mathbf{X}$ and $\mathbf{Z}$ that appear in the ID are almost invariably dense, which appears to necessitate the storage of $(m + n)k$ floating-point numbers for the double-sided ID. Observe, however, that these matrices satisfy the relations*

$$\mathbf{X} = \mathbf{C}\mathbf{A}_{\mathrm{s}}^{-1}, \qquad \text{and} \qquad \mathbf{Z} = \mathbf{A}_{\mathrm{s}}^{-1}\mathbf{R}.$$

This means that as long as we store the index vectors I_{s} and J_{s}, the matrices $\mathbf{X}$ and $\mathbf{Z}$ can be applied on the fly whenever needed, and do not need to be explicitly formed.

13.2. **Approximate rank.** In practical applications, the situation we considered in Section 13.1 where a matrix has exact rank k is rare. Instead, we typically work with a matrix whose singular values decay fast enough that it is advantageous to form a low-rank approximation. Both the ID and the CUR can be used in this environment, but now the discussion becomes slightly more involved.

To illustrate, let us consider a situation where we are given a tolerance ε, and we seek to compute an approximation $\mathbf{A}_k$ of rank k, with k as small as possible, such that $\|\mathbf{A} - \mathbf{A}_k\| \leq \varepsilon$. If $\mathbf{A}_k$ is a truncated singular value decomposition, then the Eckart–Young theorem implies that the rank k of the approximation $\mathbf{A}_k$ will be minimal. When we use an approximate algorithm, such as the RSVD (Section 11.2), we may not find the exact optimum, but we typically get very close.

What happens if we seek an ID $\mathbf{A}_k$ that approximates $\mathbf{A}$ to a fixed tolerance? There is no guarantee that the rank k (that is, the number of rows or columns involved) will be close to the rank of the truncated SVD. How close can we get in practice?

When the singular values of $\mathbf{A}$ decay rapidly, then the minimal rank attainable by an approximate ID is close to what is attainable with an SVD. Moreover, the algorithms we will describe for computing an ID produce an answer that is close to the optimal one.

When the singular values decay slowly, however, the difference in rank between the optimal ID and the optimal SVD can be quite substantial (Gu and Eisenstat 1996). On top of that, the algorithms used to compute the ID can result in answers that are still further away from the optimal value (Cheng, Gimbutas, Martinsson and Rokhlin 2005).

When the CUR decomposition is used in an environment of approximate rank, standard algorithms start by determining index sets I_{s} and J_{s} that identify the spanning rows and columns, and then proceed to the problem of finding a "good" linking matrix $\mathbf{U}_{\mathrm{s}}$. One could still use the formula (13.5), but this is rarely a good idea. The most obvious reason is that the matrix $\mathbf{A}(I_{\mathrm{s}}, J_{\mathrm{s}})$ need not be invertible in this situation. Indeed, when randomized sampling is used to find the index sets, it is common practice to compute index vectors that hold substantially more elements than is theoretically necessary, which can easily make $\mathbf{A}(I_{\mathrm{s}}, J_{\mathrm{s}})$ singular or, at the very least, highly ill-conditioned. In this case, a better approximation is given by

$$\mathbf{U} = \mathbf{C}^{\dagger}\mathbf{A}\mathbf{R}^{\dagger}, \tag{13.6}$$

with $\mathbf{C}^{\dagger}$ and $\mathbf{R}^{\dagger}$ the pseudoinverses of $\mathbf{C}$ and $\mathbf{R}$. (As always, pseudoinverses should be applied numerically by computing a QR or SVD factorization.)

13.3. **Deterministic methods, and the connection to column-pivoted QR.** A substantial amount of research effort has been dedicated to the question of how to find a set of good spanning columns and/or rows of a given matrix. It is known that the task of finding the absolutely optimal one is combinatorially hard, but efficient algorithms exist that are guaranteed to produce a close-to-optimal answer (Gu and Eisenstat 1996). In this subsection, we briefly discuss some deterministic methods that work well for dense matrices of modest size. In Section 13.4, we will show how these methods can be combined with randomized techniques to arrive

at algorithms that work well for general matrices, whether they are small or huge, sparse or dense, available explicitly or not, etc.

Perhaps the most obvious deterministic method for computing an ID is the classical Gram–Schmidt process, which selects the columns or rows in a greedy fashion. Say we are interested in the column ID (13.1) of a given matrix $\mathbf{A}$. The Gram–Schmidt procedure first grabs the largest column and places it in the first column of $\mathbf{C}$. Then it projects the remaining columns onto the orthogonal complement of the one that was picked. It places the largest of the resulting columns in the second column of $\mathbf{C}$, and so on.

In the traditional numerical linear algebra literature, it is customary to formulate the Gram–Schmidt process as a column-pivoted QR (CPQR) decomposition. After k steps, this factorization results in a partial decomposition of $\mathbf{A}$ such that

$$\underset{m\times n}{\mathbf{A}} \quad \underset{n\times n}{\mathbf{\Pi}} \quad = \quad \underset{m\times k}{\mathbf{Q}} \quad \underset{k\times n}{\mathbf{S}} \quad + \quad \underset{m\times n}{\mathbf{E}}, \tag{13.7}$$

where the columns of $\mathbf{Q}$ form an orthonormal basis for the space spanned by the k selected columns of $\mathbf{A}$, where $\mathbf{S}$ is upper-triangular, where $\mathbf{E}$ is a "remainder matrix" holding what remains of the $n-k$ columns of $\mathbf{A}$ that have not yet been picked, and where $\mathbf{\Pi}$ is a permutation matrix that reorders the columns of $\mathbf{A}$ in such a way that the k columns picked are the first k columns of $\mathbf{A\Pi}$. (We use the letter $\mathbf{S}$ for the upper-triangular factor in lieu of the more traditional $\mathbf{R}$ to avoid confusion with the matrix $\mathbf{R}$ holding spanning rows in (13.2) and (13.4).)

In order to convert (13.7) into the ID (13.1), we split off the first k columns of $\mathbf{S}$ into a $k \times k$ upper-triangular matrix $\mathbf{S}_{11}$, so that

$$\mathbf{S} = \begin{array}{c} \\ k \end{array} \begin{array}{c} \begin{array}{cc} k & n-k \end{array} \\ \begin{bmatrix} \mathbf{S}_{11} & \mathbf{S}_{12} \end{bmatrix} \end{array}.$$

Upon multiplying (13.7) by $\mathbf{\Pi}^*$ from the right, we obtain

$$\mathbf{A} = \underbrace{\mathbf{Q}\mathbf{S}_{11}}_{=:\mathbf{C}} \underbrace{\begin{bmatrix} \mathbf{I}_k & \mathbf{S}_{11}^{-1}\mathbf{S}_{12} \end{bmatrix} \mathbf{\Pi}^*}_{=:\mathbf{Z}} + \mathbf{E}\mathbf{\Pi}^*. \tag{13.8}$$

We recognize equation (13.8) as the ID (13.1), with the only difference that there is now a remainder term that results from the fact that $\mathbf{A}$ is only approximately rank-deficient. (Observe that the remainder terms in (13.7) and in (13.8) are identical, up to a permutation of the columns.)

A row ID can obviously be computed by applying Gram–Schmidt to the rows of $\mathbf{A}$ instead of the columns. Alternatively, one may express this as a column-pivoted QR factorization of $\mathbf{A}^*$ instead of $\mathbf{A}$.

In order to build a double-sided ID, one starts by computing a single-sided ID. If $m \geq n$, it is best to start with a column ID of $\mathbf{A}$ to determine J_{s} and $\mathbf{Z}$. Then we perform a row ID *on the rows of* $\mathbf{A}(:, J_{\mathrm{s}})$ to determine I_{s} and $\mathbf{X}$.

Finally, in order to build a CUR factorization of $\mathbf{A}$, we can easily convert the double-sided ID to a CUR factorization using (13.5) or (13.6).

Detailed descriptions of all algorithms can be found in Sections 10 & 11 of Martinsson (2018), while analysis and numerical results are given in Voronin and Martinsson (2017). A related set of deterministic techniques that are efficient and often result in slightly higher quality spanning sets than column pivoting are described in Sorensen and Embree (2016). Techniques based on optimized spanning volumes of submatrices are described in (Goreinov, Oseledets, Savostyanov, Tyrtyshnikov and Zamarashkin 2010) and (Thurau, Kersting and Bauckhage 2012).

Remark 13.2 (Quality of ID)**.** *In this section, we have described simple methods based on the column-pivoted QR factorization for computing a CUR decomposition, as well as all three flavors of interpolatory decompositions. In discussing the quality of the resulting factorizations, we will address two questions: (1) How close to minimal is the resulting approximation error? (2) How well-conditioned are the basis matrices?*

The NLA literature contains a detailed study of both questions. This inquiry was instigated by Kahan's construction of matrices for which CPQR performs very poorly (Kahan 1966, Sec. 5). Gu and Eisenstat (1996) provided a comprehensive analysis of the situation and presented an algorithm whose asymptotic complexity in typical environments is only slightly worse than that of CPQR and that is guaranteed to produce near-optimal results.

*In practice, CPQR works well. In almost all cases, it yields factorizations that are close to optimal. Moreover, it gives well-conditioned factorizations as long as orthonormality of the basis is scrupulously maintained (Martinsson et al. 2006*b*, Cheng et al. 2005).*

A more serious problem with the ID and the CUR is that these decompositions can exhibit much larger approximation errors than the SVD when the input matrix has slowly decaying singular values. This issue persists even when the optimal index sets are used.

13.4. **Randomized methods for finding natural bases.** The deterministic techniques for computing an ID or a CUR decomposition in Section 13.3 work very well for small, dense matrices. In this section, we describe randomized methods that work much better for matrices that are sparse or are just very large.

To be concrete, we consider the problem of finding a vector I_{s} that identifies a set of rows that form a good basis for the row space of a given matrix $\mathbf{A}$. To do so, we use the randomized rangefinder to build a matrix $\mathbf{Y}$ whose columns accurately span the column space of $\mathbf{A}$ as in Section 11. Since $\mathbf{Y}$ is far smaller than $\mathbf{A}$, we can use the deterministic methods in Section 13.3 to find a set I_{s} of rows of $\mathbf{Y}$ that form a basis for the row space of $\mathbf{Y}$. Next, we establish a simple but perhaps non-obvious fact: the set I_{s} also identifies a set of rows of $\mathbf{A}$ that form a good basis for the row space of $\mathbf{A}$.

To simplify the argument, let us first suppose that we are given an $m \times n$ matrix $\mathbf{A}$ of *exact* rank k, and that we have determined by some means (say, the randomized rangefinder) an $m \times k$ matrix $\mathbf{Y}$ whose columns span the column space of $\mathbf{A}$. Then $\mathbf{A}$ admits by definition a factorization

$$\underset{m \times n}{\mathbf{A}} \quad = \quad \underset{m \times k}{\mathbf{Y}} \quad \underset{k \times n}{\mathbf{F}}, \tag{13.9}$$

for some matrix $\mathbf{F}$. Now compute a row ID of $\mathbf{Y}$, by performing Gram–Schmidt on its rows, as described in Section 13.3. The result is a matrix $\mathbf{X}$ and an index vector I_{s} such that

$$\underset{m \times k}{\mathbf{Y}} \quad = \quad \underset{m \times k}{\mathbf{X}} \quad \underset{k \times k}{\mathbf{Y}(I_{\mathrm{s}}, :)}. \tag{13.10}$$

The claim is now that $\{I_{\mathrm{s}}, \mathbf{X}\}$ is automatically a row ID of $\mathbf{A}$ as well. To prove this, observe that

$$\begin{aligned} \mathbf{X}\mathbf{A}(I_{\mathrm{s}}, :) &= \mathbf{X}\mathbf{Y}(I_{\mathrm{s}}, :)\mathbf{F} && \{\text{use (13.9) restricted to the rows in } I_{\mathrm{s}}\} \\ &= \mathbf{Y}\mathbf{F} && \{\text{use (13.10)}\} \\ &= \mathbf{A} && \{\text{use (13.9)}\} \end{aligned}$$

The key insight here is simple and powerful: *In order to compute a row ID of a matrix* $\mathbf{A}$*, the only information needed is a matrix* $\mathbf{Y}$ *whose columns span the column space of* $\mathbf{A}$.

The task of finding a matrix $\mathbf{Y}$ such that (13.9) holds to high accuracy is particularly well suited for the randomized rangefinder described in Section 11. Putting everything together, we obtain Algorithm 15. When a Gaussian random matrix is used, the method has complexity $O(mnk)$.

Remark 13.3 ($O(mn \log k)$ complexity methods)**.** *An interesting thing happens if we replace the Gaussian random matrix* $\mathbf{\Omega}$ *in Algorithm 15 with a structured random matrix, as described in Section 9: Then* $\mathbf{Y}$ *is computed at cost* $O(mn \log k)$*, and every step after that has cost* $O((m+n)k^2)$ *or less.*

13.5. **Techniques based on coordinate sampling.** To find natural bases for a matrix, it is tempting just to sample coordinates from some probability distribution on the full index vector. Some advantages and disadvantages of this approach were discussed in Section 9.6.

Algorithm 15 *Randomized ID.*
Implements the procedure from Section 13.4.
The function `ID_row` refers to any algorithm for computing a row-ID, so that given a matrix $\mathbf{B}$ and a rank k, calling $[I_{\mathrm{s}}, \mathbf{X}] = \texttt{row\_ID}(\mathbf{B}, k)$ results in an approximate factorization $\mathbf{B} \approx \mathbf{X}\mathbf{B}(I_{\mathrm{s}}, :)$. The techniques based on column-pivoted QR described in Section 13.3 work well.

Input: Matrix $\mathbf{A} \in \mathbb{F}^{m \times n}$, target rank k, oversampling parameter p.
Output: An $m \times k$ interpolation matrix $\mathbf{X}$ and an index vector I_{s} such that $\mathbf{A} \approx \mathbf{X}\mathbf{A}(I_{\mathrm{s}}, :)$.

function RANDOMIZEDID($\mathbf{A}$, k, p)
  Draw an $n \times (k + p)$ test matrix $\mathbf{\Omega}$, e.g., from a Gaussian distribution
  Form the sample matrix $\mathbf{Y} = \mathbf{A}\mathbf{\Omega}$ ▷ Powering may be used
  Form an ID of the $n \times (k + p)$ sample matrix: $[I_{\mathrm{s}}, \mathbf{X}] = \texttt{ID\_row}(\mathbf{Y}, k)$

In the current context, the main appeal of coordinate sampling is that the cost is potentially lower than the techniques described in this section—provided that we do not need to expend much effort to compute the sampling probabilities. This advantage can be decisive in applications where mixing random embeddings are too expensive.

Coordinate sampling has several disadvantages in comparison to using mixing random embeddings. Coordinate sampling typically results in worse approximations for a given budget of rows or columns. Moreover, the quality of the approximation obtained from coordinate sampling tends to be highly variable. These vulnerabilities are less pronounced when the matrix has very low coherence, so that uniform sampling works well. There are also a few specialized situations where we can compute subspace leverage scores efficiently. (Section 9.6.1 defines coherence and leverage scores.)

In certain applications, a hybrid approach can work well. First, form an initial approximation by drawing a very large subset of columns using a cheap coordinate sampling method. Then slim it down using the techniques described here, based on mixing random embeddings. An example of this methodology appears in Li, Bi, Kwok and Lu (2015*a*).

There is a distinct class of methods, based on coresets, that explicitly takes advantage of coordinate structure for computing matrix approximations. For example, see Feldman, Volkov and Rus (2016). These techniques can be useful for processing enormous matrices that are very sparse. On the other hand, they may require larger sets of basis vectors to achieve the same quality of approximation.

14. NYSTRÖM APPROXIMATION

We continue our discussion of matrix approximation with the problem of finding a low-rank approximation of a positive semidefinite (PSD) matrix. There is an elegant randomized method for accomplishing this goal that is related to our solution to the rangefinder problem.

14.1. **Low-rank PSD approximation.** Let $\mathbf{A} \in \mathbb{H}_n$ be a PSD matrix. For a rank parameter k, the goal is to produce a rank-k PSD matrix $\widehat{\mathbf{A}}_k \in \mathbb{H}_n$ that approximates $\mathbf{A}$ nearly as well as the best rank-k matrix:

$$\|\mathbf{A} - \widehat{\mathbf{A}}_k\| \lesssim \sigma_{k+1}.$$

To obtain the approximation, we will adapt the randomized rangefinder method (Algorithm 7).

14.2. **The Nyström approximation.** The most natural way to construct a low-rank approximation of a PSD matrix is via the Nyström method. Let $\mathbf{X} \in \mathbb{F}^{n \times \ell}$ be an arbitrary test matrix. The *Nyström approximation* of $\mathbf{A}$ with respect to $\mathbf{X}$ is the PSD matrix

$$\mathbf{A}\langle\mathbf{X}\rangle := (\mathbf{A}\mathbf{X})(\mathbf{X}^*\mathbf{A}\mathbf{X})^{\dagger}(\mathbf{A}\mathbf{X})^*. \tag{14.1}$$

An alternative presentation of this formula is

$$\mathbf{A}\langle\mathbf{X}\rangle = \mathbf{A}^{1/2}\mathbf{P}_{\mathbf{A}^{1/2}\mathbf{X}}\mathbf{A}^{1/2},$$

Algorithm 16 *Randomized Nyström approximation.*
Implements the procedure from Section 14.3.

Input: Psd target matrix $\mathbf{A} \in \mathbb{H}_n(\mathbb{F})$, rank k for approximation, number ℓ of samples
Output: Rank-k PSD approximation $\widehat{\mathbf{A}}_k \in \mathbb{H}_n(\mathbb{F})$ expressed in factored form $\widehat{\mathbf{A}}_k = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^*$ where $\mathbf{U} \in \mathbb{F}^{n\times k}$ is orthonormal and $\mathbf{\Lambda} \in \mathbb{H}_k(\mathbb{F})$ is nonnegative and diagonal.

function RANDOMNYSTRÖM($\mathbf{A}$, k, ℓ)
  Draw random matrix $\mathbf{\Omega} \in \mathbb{F}^{n\times\ell}$
  Form $\mathbf{Y} = \mathbf{A}\mathbf{\Omega}$
  $\nu = \sqrt{n}\,\mathtt{eps}(\mathtt{norm}(\mathbf{Y}))$ ▷ Compute shift
  $\mathbf{Y}_\nu = \mathbf{Y} + \nu\mathbf{\Omega}$ ▷ Samples of shifted matrix
  $\mathbf{C} = \mathtt{chol}(\mathbf{\Omega}^*\mathbf{Y}_\nu)$
  $\mathbf{B} = \mathbf{Y}_\nu\mathbf{C}^{-1}$ ▷ Triangular solve!
  $[\mathbf{U}, \mathbf{\Sigma}, \sim] = \mathtt{svd}(\mathbf{B})$ ▷ Dense SVD
  $\mathbf{\Lambda} = \max\{0, \mathbf{\Sigma}^2 - \nu\mathbf{I}\}$ ▷ Remove shift
  $\mathbf{U} = \mathbf{U}(:, 1:k)$ and $\mathbf{\Lambda} = \mathbf{\Lambda}(1:k, 1:k)$ ▷ Truncate rank to k

where $\mathbf{P}_\mathbf{Y}$ is the orthogonal projector onto the range of $\mathbf{Y}$. In particular, the Nyström approximation only depends on the range of the matrix $\mathbf{X}$.

The Nyström approximation (14.1) is closely related to the Schur complement (2.4) of $\mathbf{A}$ with respect to $\mathbf{X}$. Indeed,

$$\mathbf{A}/\mathbf{X} = \mathbf{A} - \mathbf{A}\langle\mathbf{X}\rangle = \mathbf{A}^{1/2}(\mathbf{I} - \mathbf{P}_{\mathbf{A}^{1/2}\mathbf{X}})\mathbf{A}^{1/2}.$$

That is, the Schur complement of $\mathbf{A}$ with respect to $\mathbf{X}$ is precisely the error in the Nyström approximation.

Proposition 11.1 indicates that the Nyström decomposition is also connected with our approach to solving the rangefinder problem. We immediately perceive the opportunity to use a random test matrix $\mathbf{X}$ to form the Nyström approximation. Let us describe how this choice leads to algorithms for computing a near-optimal low-rank approximation of the matrix $\mathbf{A}$.

14.3. **Randomized Nyström approximation algorithms.** Here is a simple and effective procedure for computing a rank-k PSD approximation of the PSD matrix $\mathbf{A} \in \mathbb{H}_n$.

First, draw a random test matrix $\mathbf{\Omega} \in \mathbb{F}^{n\times\ell}$, where $\ell \geq k$. Form the sample matrix $\mathbf{Y} = \mathbf{A}\mathbf{\Omega} \in \mathbb{F}^{n\times\ell}$. Then compute the Nyström approximation

$$\widehat{\mathbf{A}} = \mathbf{A}\langle\mathbf{\Omega}\rangle = \mathbf{Y}(\mathbf{\Omega}^*\mathbf{Y})^\dagger\mathbf{Y}^*. \tag{14.2}$$

The initial approximation $\widehat{\mathbf{A}}$ has rank ℓ. To truncate the rank to k, we just report a best rank-k approximation $\widehat{\mathbf{A}}_k$ of the initial approximation $\widehat{\mathbf{A}}$ with respect to the Frobenius norm; see (Tropp et al. 2017*a*, Pourkamali-Anaraki and Becker 2019) and (Wang, Gittens and Mahoney 2019).

Let us warn the reader that the formula (14.2) is not suitable for numerical computation. See Algorithm 16 for a numerically stable implementation adapted from (Li et al. 2017) and (Tropp et al. 2017*a*). In general, the matrix–matrix multiply with $\mathbf{A}$ dominates the cost, with $O(n^2\ell)$ arithmetic operations; this expense can be reduced if either $\mathbf{A}$ or $\mathbf{\Omega}$ admits fast multiplication. The approximation steps involve $O(n\ell^2)$ arithmetic. Meanwhile, storage costs are $O(n\ell)$.

Another interesting aspect of Algorithm 16 is that it only uses linear information about the matrix $\mathbf{A}$. Therefore, it can be implemented in the one-pass or the streaming data model. Remark 15.1 gives more details about matrix approximation in the streaming model. See Section 19.3.5 for an application to kernel principal component analysis.

14.4. **Analysis.** The randomized Nyström method enjoys the same kind of guarantees as the randomized rangefinder. The following result (Tropp et al. 2017*a*, Thm. 4.1) extends earlier contributions from (Halko et al. 2011*a*) and (Gittens 2013).

Theorem 14.1 (Nyström: Gaussian analysis)**.** *Fix a PSD matrix $\mathbf{A} \in \mathbb{H}_n(\mathbb{F})$ with eigenvalues $\lambda_1 \geq \lambda_2 \geq \dots$. Let $1 \leq k < \ell \leq n$. Draw a random test matrix $\boldsymbol{\Omega} \in \mathbb{F}^{n \times \ell}$ that is standard normal. Then the rank-k PSD approximation $\widehat{\mathbf{A}}_k$ computed by Algorithm 16 satisfies*

$$\mathbb{E}\,\|\mathbf{A} - \widehat{\mathbf{A}}_k\| \leq \lambda_{k+1} + \frac{k}{\ell - k - 1}\left(\sum\nolimits_{j>k} \lambda_j\right).$$

In other words, the computed rank-k approximation $\widehat{\mathbf{A}}_k$ achieves almost the error λ_{k+1} in the optimal rank-k approximation of $\mathbf{A}$. The error declines as the number ℓ of samples increases and as the ℓ_1 norm of the tail eigenvalues decreases.

When the target matrix $\mathbf{A}$ has a sharply decaying spectrum, Theorem 14.1 can be pessimistic. See Tropp et al. (2017*a*, Sec. 4) for additional theoretical results.

14.5. **Powering.** We can reduce the error in the randomized Nyström approximation by powering the input matrix, much as subspace iteration improves the performance in the randomized rangefinder (Section 11.6).

Let $\mathbf{A} \in \mathbb{H}_n$ be a PSD matrix. Draw a random test matrix $\boldsymbol{\Omega} \in \mathbb{F}^{n \times \ell}$. For a natural number q, we compute $\mathbf{Y} = \mathbf{A}^q\boldsymbol{\Omega}$ by repeated multiplication. Then the Nyström approximation of the input matrix takes the form

$$\widehat{\mathbf{A}} = \left[(\mathbf{A}^q)\langle\boldsymbol{\Omega}\rangle\right]^{1/q} = \left[\mathbf{Y}(\boldsymbol{\Omega}^*\mathbf{Y})^\dagger\mathbf{Y}^*\right]^{1/q}.$$

This approach requires very careful numerical implementation. As in the case of subspace iteration, it drives down the error exponentially fast as q increases. We omit the details.

14.6. **History.** The Nyström approximation was developed in the context of integral equations (Nyström 1930). It has had a substantial impact in machine learning, beginning with the work of Williams and Seeger (2001) on randomized low-rank approximation of kernel matrices. Section 19 contains a discussion of this literature. Note that the Nyström approximation of a kernel matrix is almost always computed with respect to a random coordinate subspace, in contrast to the uniformly random subspace induced by a Gaussian test matrix.

Algorithmic and theoretical results on Nyström approximation with respect to general test matrices have appeared in a number of papers, including (Halko et al. 2011*a*, Gittens 2013, Li et al. 2017) and (Tropp, Yurtsever, Udell and Cevher 2017*b*).

15. Single-view algorithms

In this section, we will describe a remarkable class of algorithms that are capable of computing a low-rank approximation of a matrix that is so large that it cannot be stored at all.

We will consider the specific problem of computing an approximate singular value decomposition of a matrix $\mathbf{A} \in \mathbb{F}^{m \times n}$ under the assumption that we are allowed to view each entry of $\mathbf{A}$ only once and that we cannot specify the order in which they are viewed. To the best of our knowledge, no deterministic techniques can carry off such a computation without *a priori* information about the singular vectors of the matrix.

For the case where $\mathbf{A}$ is psd, we have already seen a single-view algorithm: the Nyström technique of Algorithm 16. Here, we concentrate on the more difficult case of general matrices. This presentation is adapted from Halko et al. (2011*a*, Sec. 5.5) and the papers (Tropp, Yurtsever, Udell and Cevher 2017*c*) and (Tropp et al. 2019).

15.1. **Algorithms.** In the basic RSVD algorithm (Section 11.2), we view each element of the given matrix $\mathbf{A}$ at least twice. In the first view, we form a sample matrix $\mathbf{Y} = \mathbf{A}\boldsymbol{\Omega}$ for a given test matrix $\boldsymbol{\Omega}$. We orthonormalize the columns of $\mathbf{Y}$ to form the matrix $\mathbf{Q}$ and then visit $\mathbf{A}$ again to form a second sample $\mathbf{C} = \mathbf{Q}^*\mathbf{A}$. The columns of $\mathbf{Y}$ form an approximate basis for the column space of $\mathbf{A}$, and the columns of $\mathbf{C}$ form an approximate basis for the row space.

In the single-view framework, we can only visit $\mathbf{A}$ once, which means that we must sample both the row and the column space simultaneously. To this end, let us draw tall thin random matrices

$$\boldsymbol{\Upsilon} \in \mathbb{F}^{m \times \ell} \qquad \text{and} \qquad \boldsymbol{\Omega} \in \mathbb{F}^{n \times \ell} \tag{15.1}$$

and then form the two corresponding sample matrices

$$\mathbf{X} = \mathbf{A}^*\boldsymbol{\Upsilon} \in \mathbb{F}^{n\times\ell} \qquad \text{and} \qquad \mathbf{Y} = \mathbf{A}\boldsymbol{\Omega} \in \mathbb{F}^{m\times\ell}. \tag{15.2}$$

In (15.1), we draw a number ℓ of samples that is slightly larger than the rank k of the low-rank approximation that we seek. (Section 15.2 gives details about how to choose ℓ.) Observe that both $\mathbf{X}$ and $\mathbf{Y}$ can be formed in a single pass over the matrix $\mathbf{A}$.

Once we have seen the entire matrix, the next step is to orthonormalize the columns of $\mathbf{X}$ and $\mathbf{Y}$ to obtain orthonormal matrices

$$[\mathbf{P}, \sim] = \texttt{qr\_econ}(\mathbf{X}), \qquad \text{and} \qquad [\mathbf{Q}, \sim] = \texttt{qr\_econ}(\mathbf{Y}). \tag{15.3}$$

At this point, $\mathbf{P}$ and $\mathbf{Q}$ hold approximate bases for the row and column spaces of $\mathbf{A}$, so we anticipate that

$$\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^*\mathbf{A}\mathbf{P}\mathbf{P}^* = \mathbf{Q}\mathbf{C}\mathbf{P}^*, \tag{15.4}$$

where we defined the "core" matrix

$$\mathbf{C} := \mathbf{Q}^*\mathbf{A}\mathbf{P} \in \mathbb{F}^{\ell\times\ell}. \tag{15.5}$$

Unfortunately, since we cannot revisit $\mathbf{A}$, we are not allowed to form $\mathbf{C}$ directly by applying formula (15.5).

Instead, we develop a relation that $\mathbf{C}$ must satisfy approximately, which allows us to estimate $\mathbf{C}$ from the quantities we have on hand. To do so, right-multiply the definition (15.5) by $\mathbf{P}^*\boldsymbol{\Omega}$ to obtain

$$\mathbf{C}\big(\mathbf{P}^*\boldsymbol{\Omega}\big) = \mathbf{Q}^*\mathbf{A}\mathbf{P}\big(\mathbf{P}^*\boldsymbol{\Omega}\big). \tag{15.6}$$

Inserting the approximation $\mathbf{A}\mathbf{P}\mathbf{P}^* \approx \mathbf{A}$ into (15.6), we find that

$$\mathbf{C}\big(\mathbf{P}^*\boldsymbol{\Omega}\big) \approx \mathbf{Q}^*\mathbf{A}\boldsymbol{\Omega} = \mathbf{Q}^*\mathbf{Y}. \tag{15.7}$$

In (15.7), all quantities except $\mathbf{C}$ are known explicitly, which means that we can solve it, in the least-squares sense, to arrive at an estimate

$$\mathbf{C}_{\text{approx}} = \big(\mathbf{Q}^*\mathbf{Y}\big)\big(\mathbf{P}^*\boldsymbol{\Omega}\big)^\dagger, \tag{15.8}$$

where $\dagger$ denotes the Moore-Penrose pseudoinverse. As always, the pseudoinverse is applied by means of an orthogonal factorization. Once $\mathbf{C}_{\text{approx}}$ has been computed via (15.8), we obtain the rank-ℓ approximation

$$\mathbf{A} \approx \mathbf{Q}\mathbf{C}_{\text{approx}}\mathbf{P}^*, \tag{15.9}$$

which we can convert into an approximate SVD using the standard postprocessing steps. For additional implementation details, see Halko et al. (2011*a*, Sec. 5.5) and Martinsson (2018, Sec. 5). Extensions of this approach, with theoretical analysis, appear in Tropp et al. (2017*b*).

Recently, Tropp et al. (2019) have demonstrated that the numerical performance of the single-view algorithm can be improved by extracting a third sketch of $\mathbf{A}$ that is independent from $\mathbf{X}$ and $\mathbf{Y}$. The idea is to draw tall thin random matrices

$$\boldsymbol{\Phi} \in \mathbb{F}^{m\times s}, \qquad \text{and} \qquad \boldsymbol{\Psi} \in \mathbb{F}^{n\times s},$$

where s is another oversampling parameter. Then we form a "core sketch"

$$\mathbf{Z} = \boldsymbol{\Phi}^*\mathbf{A}\boldsymbol{\Psi} \in \mathbb{F}^{s\times s}. \tag{15.10}$$

This extra data allows us to derive an alternative equation for the core matrix $\mathbf{C}$. We left- and right-multiply the definition (15.5) by $\boldsymbol{\Phi}^*\mathbf{Q}$ and $\mathbf{P}^*\boldsymbol{\Psi}$ to obtain the relation

$$\big(\boldsymbol{\Phi}^*\mathbf{Q}\big)\mathbf{C}\big(\mathbf{P}^*\boldsymbol{\Psi}\big) = \boldsymbol{\Phi}^*\mathbf{Q}\mathbf{Q}^*\mathbf{A}\mathbf{P}\mathbf{P}^*\boldsymbol{\Psi}. \tag{15.11}$$

Inserting the approximation $\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^*\mathbf{A}\mathbf{P}\mathbf{P}^*$ into (15.11), we find that

$$\big(\boldsymbol{\Phi}^*\mathbf{Q}\big)\mathbf{C}\big(\mathbf{P}^*\boldsymbol{\Psi}\big) \approx \boldsymbol{\Phi}^*\mathbf{A}\boldsymbol{\Psi} = \mathbf{Z}. \tag{15.12}$$

An improved approximation to the core matrix $\mathbf{C}$ results by solving (15.12) in a least-squares sense; to wit, $\mathbf{C} = (\boldsymbol{\Phi}^*\mathbf{Q})^\dagger\mathbf{Z}(\mathbf{P}^*\boldsymbol{\Psi})^\dagger$.

Algorithm 17 *Single-view SVD.*
Implements the algorithm from Section 15.1.
This algorithm computes an approximate partial singular value decomposition of a given matrix $\mathbf{A}$, under the constraint that each entry of $\mathbf{A}$ may be viewed only once.

Input: Target matrix $\mathbf{A} \in \mathbb{F}^{m\times n}$, rank k, sampling sizes ℓ and s.
Output: Orthonormal matrices $\mathbf{U} \in \mathbb{F}^{m\times k}$ and $\mathbf{V} = \mathbb{F}^{n\times k}$, and a diagonal matrix $\mathbf{\Sigma} \in \mathbb{F}^{k\times k}$ such that $\mathbf{A} \approx \mathbf{U\Sigma V}^*$.

function SINGLEVIEWSVD($\mathbf{A}$, k, ℓ, s)
  Draw test matrices $\mathbf{\Upsilon} \in \mathbb{F}^{m\times \ell}$, $\mathbf{\Omega} \in \mathbb{F}^{n\times \ell}$, $\mathbf{\Phi} \in \mathbb{F}^{m\times s}$, $\mathbf{\Psi} \in \mathbb{F}^{n\times s}$
  Form $\mathbf{X} = \mathbf{A}^*\mathbf{\Upsilon}$, $\mathbf{Y} = \mathbf{A\Omega}$, $\mathbf{Z} = \mathbf{\Phi}^*\mathbf{A\Psi}$ ▷ Viewing $\mathbf{A}$ only once!
  $[\mathbf{P}, \sim] = \texttt{qr\_econ}(\mathbf{X})$, $[\mathbf{Q}, \sim] = \texttt{qr\_econ}(\mathbf{Y})$
  $\mathbf{C} = \left(\mathbf{\Phi}^*\mathbf{Q}\right)^\dagger \mathbf{Z} \left(\mathbf{P}^*\mathbf{\Psi}\right)^\dagger$ ▷ Execute using a least-squares solver
  $[\widehat{\mathbf{U}}, \widehat{\mathbf{\Sigma}}, \widehat{\mathbf{V}}] = \texttt{svd}(\mathbf{C})$ ▷ A full SVD
  $\mathbf{U} = \mathbf{Q}\widehat{\mathbf{U}}(:, 1:k)$, $\mathbf{V} = \mathbf{P}\widehat{\mathbf{V}}(:, 1:k)$, $\mathbf{\Sigma} = \widehat{\mathbf{\Sigma}}(1:k, 1:k)$

Remark 15.1 (Streaming algorithms)**.** *Single-view algorithms are related to the streaming model of computation (Muthukrishnan 2005). Clarkson and Woodruff (2009) were the first to explicitly study matrix computations in streaming data models.*

One important streaming model poses the assumption that the input matrix $\mathbf{A}$ is presented as a sequence of innovations:

$$\mathbf{A} = \mathbf{H}_1 + \mathbf{H}_2 + \mathbf{H}_3 + \cdots$$

Typically, each update $\mathbf{H}_i$ is simple; for instance, it may be sparse or low-rank. The challenge is that the full matrix $\mathbf{A}$ is too large to be stored. Once an innovation $\mathbf{H}_i$ has been processed, it cannot be retained. This is called the "turnstile" model in the theoretical computer science literature.

The algorithms described in this section handle this difficulty by creating a random linear transform $\mathcal{S}$ that maps $\mathbf{A}$ down to a low-dimensional sketch that is small enough to store. What we actually retain in memory is the evolving sketch of the input:

$$\mathcal{S}(\mathbf{A}) = \mathcal{S}(\mathbf{H}_1) + \mathcal{S}(\mathbf{H}_2) + \mathcal{S}(\mathbf{H}_3) + \cdots.$$

In Algorithm 17, we instantiate $\mathcal{S}$ by drawing the random matrices $\mathbf{\Upsilon}$, $\mathbf{\Omega}$, $\mathbf{\Phi}$, and $\mathbf{\Psi}$, and then work with the sketch

$$\mathcal{S}(\mathbf{H}) = \left(\mathbf{\Upsilon}^*\mathbf{H},\ \mathbf{H\Omega},\ \mathbf{\Phi}^*\mathbf{H\Psi}\right).$$

The fact that the sketch is a linear map is essential here. Li et al. (2014b) prove that randomized linear embeddings are essentially the only kind of algorithm for handling the turnstile model. In contrast, the sketch implicit in the RSVD algorithm from Section 11.2 is a quadratic or higher-order polynomial in the input matrix.

Remark 15.2 (Single-view versus out-of-core algorithms)**.** *In principle, the methods discussed in this section can also be used in situations where a matrix is stored in slow memory, such as a spinning disk hard drive, or on a distributed memory system. However, one has to carefully weigh whether the decrease in accuracy and increase in uncertainty that is inherent to single-view algorithms is worth the cost savings. As a general matter, revisiting the matrix at least once is advisable whenever it is possible.*

15.2. **Error estimation, parameter choices and truncation.** In the single-view computing environment, one must choose sampling parameters before the computation starts, and there is no way to revisit these choices after data has been gathered. This constraint makes *a priori* error analysis particularly important, because we need guidance on how large to make the sketches given some prior knowledge about the spectral decay of the input matrix. To illustrate how this may work, let us cite Tropp et al. (2019, Thm. 5.1):

Theorem 15.3 (Single-view SVD: Gaussian analysis)**.** *Suppose that Algorithm 17 is executed for an input matrix $\mathbf{A} \in \mathbb{C}^{m \times n}$ and for sampling parameters s and ℓ that satisfy $s \geq 2\ell$. When the test matrices are drawn from a standard normal distribution, the computed matrices $\mathbf{P}$, $\mathbf{C}$, and $\mathbf{Q}$ satisfy*

$$\mathbb{E}\,\|\mathbf{A} - \mathbf{QCP}^*\|_{\mathrm{F}}^2 \leq \frac{s}{s-\ell} \min_{k<\ell} \left(\frac{\ell+k}{\ell-k} \sum_{j=k+1}^{\min(m,n)} \sigma_j^2 \right).$$

As usual, σ_j is the jth largest singular value of $\mathbf{A}$. A very similar bound holds for the real field.

This result suggests that more aggressive oversampling is called for in the single-view setting, as compared to the basic rangefinder problem. For instance, if we aim for an approximation error that is comparable to the best possible approximation with rank k, then we might choose $\ell = 4k$ and $s = 8k$ to obtain

$$\mathbb{E}\|\mathbf{A} - \mathbf{QCP}^*\|_{\mathrm{F}}^2 \leq \frac{10}{3} \sum_{j=k+1}^{\min(m,n)} \sigma_j^2 = \frac{10}{3}\|\mathbf{A} - \mathbf{A}_k\|_{\mathrm{F}}^2,$$

where $\mathbf{A}_k$ is the best possible rank-k approximation of $\mathbf{A}$. As usual, the likelihood of large deviations from the expectation is negligible. (For contrast, recall that the basic rangefinder algorithm often works well when we select $\ell = k + 5$ or $\ell = k + 10$.)

Besides computing $\mathbf{P}$, $\mathbf{C}$, and $\mathbf{Q}$, Algorithm 17 also prunes the approximation $\mathbf{A} \approx \mathbf{QCP}^*$ by computing an SVD of $\mathbf{C}$ (line 6) and then throwing out the trailing $\ell - k$ modes (line 7). The motivation for this truncation is that the approximation $\mathbf{A} \approx \mathbf{QCP}^*$, tends to capture the dominant singular modes of $\mathbf{A}$ well, but the trailing ones have very low accuracy. The same thing happens with the basic RSVD (Section 11.2), but the phenomenon is more pronounced in the single-view environment, in part because ℓ is substantially larger than k. Theorem 15.3 can be applied to prove that the truncated factorization is as accurate as one can reasonably hope for; see Tropp et al. (2019, Cor. 5.5) for details.

Remark 15.4 (Spectral norm bounds?)**.** *Theorem 15.3 provides a Frobenius norm error bound for a matrix approximation algorithm. For our survey, this is a* rara avis in terra*. Unfortunately, relative error spectral norm error bounds are not generally possible in the streaming setting (Woodruff 2014, Chap. 6).*

15.3. **Structured test matrices.** Algorithm 17 can – and should – be implemented with structured test matrices, rather than Gaussian test matrices. This modification is especially appealing in the single-view environment, where storage is often the main bottleneck.

For instance, consider the parameter selections $\ell = 4k$ and $s = 8k$ that we referenced above. Then the four test matrices consist of $12k(m + n)$ floats that must be stored, and the sketches add another $4k(m + n) + 64k^2$ floats. Since m and n can be huge, these numbers could severely limit the rank k of the final matrix approximation.

If we swap out the Gaussian matrices for structured random matrices, we can almost remove the cost associated with storing the test matrices. In particular, the addition of the core sketch (15.10) has a very light memory footprint because the sketch itself only uses $O(k^2)$ floats. Empirically, when we use a structured random matrix, such as a sparse sign matrix (Section 9.2) or an SRTT (Section 9.3), the observed errors are more or less indistinguishable from the errors attained with Gaussian test matrices. See Tropp et al. (2019, Supplement, Figs. SM2–7).

15.4. **A posteriori error estimation.** In order to reduce the uncertainty associated with the single-view algorithms described in this section, the "certificate of accuracy" technique described in Section 12.2 is very useful.

Recall that the idea is to draw a separate test matrix whose only purpose is to provide an independent estimate of the error in the computed solution. This additional test matrix can be *very* thin (say 5 or 10 columns wide), and it still provides a dependable bound on the

computed error. These techniques can be incorporated without any difficulty in the single-view environment, as outlined in Tropp et al. (2019, Sec. 6).

Let us mention one caveat. In the single-view environment, we have no recourse when the *a posteriori* error estimator signals that the approximation error is unacceptable. On the other hand, it is reassuring that the algorithm can sound a warning that it has not met the desired accuracy.

15.5. **History.** To the best of our knowledge, Woolfe et al. (2008, Sec. 5.2) described the first algorithm that can compute a low-rank matrix approximation in the single-view computational model. Their paper introduced the idea of independently sampling the row- and the column-space of a matrix, as summarized in formulas (15.1)–(15.9). This approach inspired the single-view algorithms presented in Halko et al. (2011*a*, Sec. 5.5). It is interesting that the primary objective of Woolfe et al. (2008) was to reduce the asymptotic flop count of the computation through the use of structured random test matrices.

Clarkson and Woodruff (2009) gave an explicit discussion of randomized NLA in a streaming computational model. They independently proposed a variant of the algorithm from (Halko et al. 2011*a*, Sec. 5.5). Later contributions to the field appeared in (Li et al. 2014*b*, Boutsidis, Woodruff and Zhong 2016, Feldman et al. 2016, Ghashami, Liberty, Phillips and Woodruff 2016*a*) and (Tropp et al. 2017*b*). The idea of introducing an additional sketch such as (15.10) to capture the "core" matrix was proposed by Upadhyay (2016). Tropp et al. (2019) have provided improvements of his approach, further analysis, and computational considerations.

16. Factoring matrices of full or nearly full rank

So far, we have focused on techniques for computing low-rank approximations of an input matrix. We will now upgrade to techniques for computing *full* rank-revealing factorizations such as the column-pivoted QR (CPQR) decomposition.

Classical deterministic techniques for computing these factorizations proceed through a sequence of rank-one updates to the matrix, making them communication-intensive and slow when executed on modern hardware. Randomization allows the algorithms to be reorganized so that the vast majority of the arithmetic takes place inside matrix–matrix multiplications, which greatly accelerates the execution speed.

When applied to an $n \times n$ matrix, most of the algorithms described in this section have the same $O(n^3)$ asymptotic complexity as traditional methods; the objective is to improve the practical speed by reducing communication costs. However, randomization also allows us to incorporate Strassen-type techniques to accelerate the matrix multiplications in a numerically stable manner, attaining an overall cost of $O(n^\omega)$ where ω is the exponent of square matrix–matrix multiplication.

As well as the CPQR decomposition, we will consider algorithms for computing factorizations of the form $\mathbf{A} = \mathbf{U}\mathbf{R}\mathbf{V}^*$ where $\mathbf{U}$ and $\mathbf{V}$ are unitary matrices and $\mathbf{R}$ is upper-triangular. Factorizations of this form can be used for almost any task where either the CPQR or the SVD is currently used. The additional flexibility allows us to improve on both the computational speed and on the rank-revealing qualities of the factorization.

Sections 16.1–16.4 introduce the key concepts by describing a simple algorithm for computing a rank-revealing factorization of a matrix. This method is both faster than traditional column-pivoted QR and better at revealing the spectral properties of the matrix. Sections 16.5–16.7 are more technical; they describe how randomization can be used to resolve a long-standing challenge of how to *block* a classical algorithm for computing a column-pivoted QR decomposition by applying groups of Householder reflectors simultaneously. They also describe how these ideas can be extended to the task of computing a URV factorization.

16.1. **Rank-revealing factorizations.** Before we discuss algorithms, let us first define what we mean when we say that a factorization is *rank-revealing*. Given an $m \times n$ matrix $\mathbf{A}$, we will consider factorizations of the form

$$\underset{m\times n}{\mathbf{A}} = \underset{m\times c}{\mathbf{U}}\ \underset{c\times n}{\mathbf{R}}\ \underset{n\times n}{\mathbf{V}^*}, \tag{16.1}$$

where $c = \min(m, n)$, where $\mathbf{U}$ and $\mathbf{V}$ are orthonormal, and where $\mathbf{R}$ is upper-triangular (or banded upper-triangular). We want the factorization to reveal the numerical rank of $\mathbf{A}$ in the sense that we obtain a near-optimal approximation of $\mathbf{A}$ when we truncate (16.1) to any level k. That is,

$$\|\mathbf{A} - \mathbf{U}(:, 1:k)\mathbf{R}(1:k, :)\mathbf{V}^*\| \approx \inf\{\|\mathbf{A} - \mathbf{B}\| : \mathbf{B} \text{ has rank } k\} \tag{16.2}$$

for $k \in \{1, 2, \ldots, c\}$. The factorization (16.1) can be viewed either as a generalization of the SVD (for which $\mathbf{R}$ is diagonal) or as a generalization of the column-pivoted QR factorization (for which $\mathbf{V}$ is a permutation matrix).

A factorization such as (16.1) that satisfies (16.2) is very handy. It can be used to solve ill-conditioned linear systems or least-squares problems; it can be used for estimating the singular spectrum of $\mathbf{A}$ (and all Schatten p-norms); and it provides orthonormal bases for approximations to the four fundamental subspaces of the matrix. Finally, it can be used to compute approximate low-rank factorizations efficiently in situations where the numerical rank of the matrix is not that much smaller than m or n. In contrast, all techniques described up to now are efficient only when the numerical rank k satisfies $k \ll \min(m, n)$.

16.2. **Blocking of matrix computations.** A well-known feature of modern computing is that we can execute increasingly many floating-point operations because CPUs are gaining more cores while GPUs and other accelerators are becoming more affordable and more energy-efficient. In contrast, the cost of communication (data transfer up and down levels of a memory hierarchy, among servers, and across networks, etc.) is declining very slowly. As a result, reducing communication is often the key to accelerating numerical algorithms in the real world.

In the context of matrix computations, the main reaction to this development has been to cast linear-algebraic operations as operating on blocks of the matrix, rather than on individual entries or individual columns and rows (Whaley and Dongarra 1998, Mathias and Stewart 1993, Martinsson, Quintana-Ortí, Heavner and van de Geijn 2017). The objective is to reorganize an algorithm so that the majority of flops can be executed using highly efficient algorithms for matrix–matrix multiplication (BLAS3), rather than the slower methods for matrix–vector multiplications (BLAS2).

Unfortunately, it turns out that classical algorithms for computing rank-revealing factorizations of matrices are very challenging to block. Column-pivoted QR proceeds through a sequence of rank-1 updates to the matrix. The next pivot cannot be found until the previous update has been applied. Techniques for computing an SVD of a matrix start by reducing the matrix to bidiagonal form. Then they iterate on the bidiagonal matrix to drive it towards diagonal form. Both steps are challenging to block.

To emphasize just how much of a difference blocking makes, let us peek ahead at the plot in Figure 1. This graph shows computational times for computing certain matrix factorizations versus matrix size on a standard desktop PC. In particular, look at the times for column-pivoted QR (red solid line) and for unpivoted QR (red dashed line). The asymptotic flop counts of these two algorithms are identical in the dominant term. Yet the unpivoted factorization can easily be blocked, which means that it executes one order of magnitude faster than the pivoted one.

16.3. **The powerURV algorithm.** There is a simple randomized algorithm for computing a rank-revealing factorization of a matrix that perfectly illustrates the power of randomization to reduce communication. Our starting point is an algorithm proposed by Demmel, Dumitriu and Holtz (2007). Given an $m \times n$ matrix $\mathbf{A}$, typically with $m \geq n$, it proceeds as follows.

(1) Draw an $n \times n$ matrix $\mathbf{\Omega}$ from a standard normal distribution.

(2) Perform an unpivoted QR factorization of $\mathbf{\Omega}$ so that $[\mathbf{V}, \sim] = \mathtt{qr}(\mathbf{\Omega})$.

(3) Perform an unpivoted QR factorization of $\mathbf{AV}$ so that $[\mathbf{U}, \mathbf{R}] = \mathtt{qr}(\mathbf{AV})$.

Observe that the purpose of steps (1) and (2) is simply to generate a matrix $\mathbf{V}$ whose columns serve as a "random" orthonormal basis. It is easily verified that the matrices $\mathbf{U}$, $\mathbf{R}$, and $\mathbf{V}$

Algorithm 18 *powerURV.*
This algorithm computes a rank-revealing URV factorization of a given matrix $\mathbf{A}$; see Section 16.3. Reorthonormalization may be required between applications of $\mathbf{A}$ and $\mathbf{A}^*$ on line 3 to combat round-off errors.

Input: Target matrix $\mathbf{A} \in \mathbb{F}^{m\times n}$ for $m \geq n$, power parameter q
Output: Orthonormal matrices $\mathbf{U} \in \mathbb{F}^{m\times n}$ and $\mathbf{V} = \mathbb{F}^{n\times n}$, and upper triangular $\mathbf{R} \in \mathbb{F}^{n\times n}$ such that $\mathbf{A} = \mathbf{U}\mathbf{R}\mathbf{V}^*$

1 **function** POWERURV($\mathbf{A}$, q)
2 Draw a test matrix $\mathbf{\Omega} \in \mathbb{F}^{n\times n}$ from a standard normal distribution
3 $[\mathbf{V}, \sim] = \texttt{qr\_econ}((\mathbf{A}^*\mathbf{A})^q\mathbf{\Omega})$ ▷ Unpivoted QR
4 $[\mathbf{U}, \mathbf{R}] = \texttt{qr\_econ}(\mathbf{A}\mathbf{V})$ ▷ Unpivoted QR

satisfy

$$\mathbf{A} = \mathbf{U}\mathbf{R}\mathbf{V}^*. \tag{16.3}$$

The factorization (16.3) is rank-revealing in theory (see Demmel et al. (2007, Theorem 5.2) and Ballard, Demmel, Dumitriu and Rusciano (2019)), but it does not reveal the rank particularly well in practice.

The cost to compute (16.3) is dominated by the cost to perform two unpivoted QR factorizations, and one matrix–matrix multiplication. (Simulating the random matrix $\mathbf{\Omega}$ requires only $O(n^2)$ flops.)

To improve the rank-revealing ability of the factorization, one can incorporate a small number of power iteration steps (Gopal and Martinsson 2018), so that step (2) in the recipe gets modified to

$$[\mathbf{V}, \sim] = \texttt{qr}((\mathbf{A}^*\mathbf{A})^q\mathbf{\Omega}), \tag{16.4}$$

where q is a small integer. In practice, $q = 1$ or $q = 2$ is often enough to dramatically improve the accuracy of the computation. Of course, incorporating power iteration increases the cost of the procedure by adding $2q$ additional matrix–matrix multiplications. When the singular values of $\mathbf{A}$ decay rapidly, reorthonormalization in between each application of $\mathbf{A}$ is sometimes required to avoid loss of accuracy due to floating-point arithmetic.

Algorithm 18 summarizes the techniques introduced in this section. The method is simple, and easy to code. It requires far more flops than traditional methods for computing rank-revealing factorizations, yet it is faster in practice. For instance, if $m = n$ and $q = 2$, then powerURV requires $\approx 5n^3$ flops versus $0.5\,n^3$ flops for CPQR, but Figure 1 shows that powerURV is still faster. This is noteworthy, since powerURV with $q = 2$ does a *far* better job at revealing the numerical rank of $\mathbf{A}$ than CPQR, as shown in Figure 2. See Gopal and Martinsson (2018) for details.

16.4. **Computing a rank-revealing factorization of an $n \times n$ matrix in less than $O(n^3)$ operations.** The basic version of the randomized URV factorization algorithm described in Section 16.3 was originally proposed by Demmel et al. (2007) for purposes loftier than practical acceleration. Indeed, randomization allows us to exploit fast matrix–matrix multiplication primitives to design accelerated algorithms for other NLA problems, such as constructing rank-revealing factorizations. The main point of this research is that, whenever the fast matrix–matrix multiplication is stable, the computation of a rank-revealing factorization is stable too.

Let us be more precise. Demmel et al. (2007) embark from the observation that there exist algorithms[1] for multiplying two $n \times n$ matrices using $O(n^\omega)$ flops, where $\omega < 3$. Once such an algorithm is available, one can stably perform a whole range of other standard matrix

[1]The celebrated method of Strassen (1969) has exponent $\omega = \log_2(7) = 2.807\cdots$. It is a compelling algorithm in terms of both its numerical stability and its practical speed, even for modest matrix sizes. More exotic algorithms, such as the Coppersmith–Winograd method and variants, attain complexity of about $\omega \approx 2.37$, but they are not considered to be practically useful.

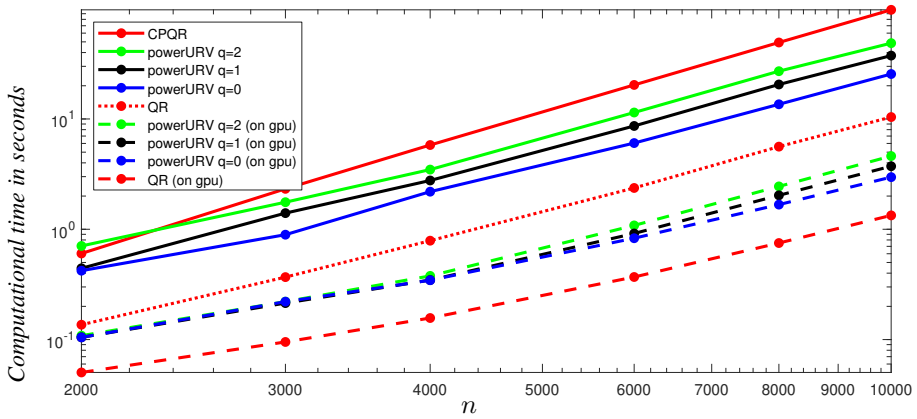


FIGURE 1. Computational times required for column-pivoted QR (CPQR) and unpivoted QR (QR) of an $n \times n$ real matrix using MATLAB on an Intel i7-8700k CPU. We see that the unpivoted factorization is an order of magnitude faster, despite having the identical asymptotic flop count. The graph also shows the times required for the randomized rank-revealing factorization described in Section 16.4, executed both on a CPU (solid lines) and an Nvidia Titan V GPU (dashed lines).

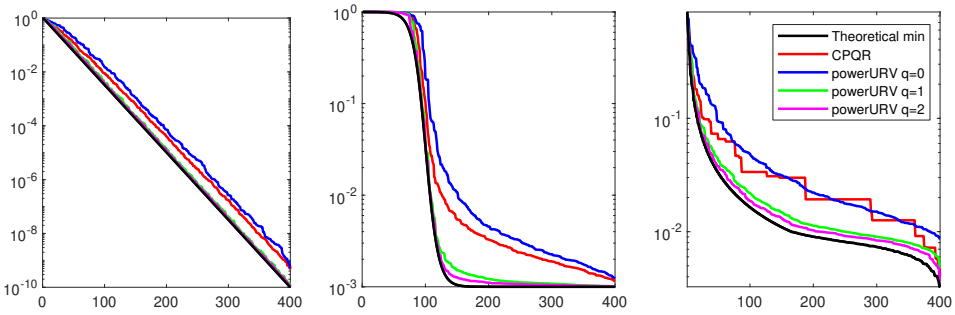


FIGURE 2. The rank-revealing ability of CPQR and powerURV for different values of the power parameter q, as discussed in Section 16.3. The error $e_k = \|\mathbf{A} - \mathbf{U}(:, 1:k)\mathbf{R}(1:k, :)\mathbf{V}^*\|$ (see (16.2)), is plotted versus k for three different matrices $\mathbf{A}$ of size 400×400. The black lines plot the theoretical minimal values σ_{k+1}. (a) A matrix whose singular values decay rapidly; we see that all methods perform well. (b) A matrix whose singular values plateau; we see that CPQR performs poorly, and so does the randomized method unless powering is used. (c) A discretized boundary integral operator whose singular values decay slowly; we again see the high precision of powerURV for $q = 1$ and $q = 2$.

operations at the same asymptotic complexity. The idea is to apply a divide-and-conquer approach that moves essentially all flops into the matrix–matrix multiplication. This approach turns out to be relatively straightforward for decompositions that do not reveal the numerical rank such as the unpivoted QR factorization. It is harder to implement, however, for (pivoted) rank-revealing factorizations.

16.5. **Classical column-pivoted QR.** The powerURV algorithm described in Section 16.3 can be very effective, but it operates on the whole matrix at once, and it cannot be used to compute a partial factorization. In the remainder of this section, we describe algorithms that

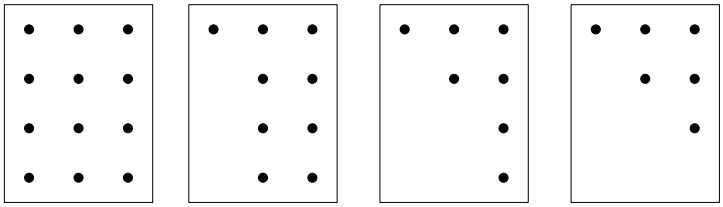


FIGURE 3. The figure shows the sparsity pattern of a 4×3 matrix $\mathbf{A}$ as it is driven to upper-triangular form in the column-pivoted QR factorization algorithm described in Section 16.5. The process takes three steps in this case, and step j involves the application of a permutation matrix $\mathbf{\Pi}_j$ from the right, and by a Householder reflector $\mathbf{Q}_j$ from the left.

build a rank-revealing factorization incrementally. These methods enjoy the property that the factorization can be halted once a specified tolerance has been met.

We start off this discussion by reviewing a classical (deterministic) method for computing a column-pivoted QR factorization. This material is elementary, but the discussion serves to set up a notational framework that lets us describe the randomized version succinctly in Section 16.6. Suppose that we are given an $m \times n$ matrix $\mathbf{A}$ with $m \geq n$. We seek a factorization of the form

$$\underset{m \times n}{\mathbf{A}} = \underset{m \times n}{\mathbf{Q}} \quad \underset{n \times n}{\mathbf{R}} \quad \underset{n \times n}{\mathbf{\Pi}^*}, \tag{16.5}$$

where $\mathbf{Q}$ has orthonormal columns, where $\mathbf{\Pi}$ is a permutation matrix, and where $\mathbf{R}$ is upper-triangular with diagonal elements that decay in magnitude so that $|\mathbf{R}(1,1)| \geq |\mathbf{R}(2,2)| \geq |\mathbf{R}(3,3)| \geq \cdots$. The factors are typically built through a sequence of steps, where $\mathbf{A}$ is driven to upper-triangular form one column at a time.

To be precise, we start by forming the matrix $\mathbf{A}_0 = \mathbf{A}$. Then we proceed using the iteration formula

$$\mathbf{A}_j = \mathbf{Q}_j^*\mathbf{A}_{j-1}\mathbf{\Pi}_j,$$

where $\mathbf{\Pi}_j$ is a permutation matrix that swaps the jth column of $\mathbf{A}_{j-1}$ with the column in $\mathbf{A}_{j-1}(:, j : n)$ that has the largest magnitude, and where $\mathbf{Q}_j$ is a Householder reflector that zeros out all elements below the diagonal in the jth column of $\mathbf{A}_{j-1}\mathbf{\Pi}_j$; see Figure 3. Once the process concludes, the relation (16.5) holds for

$$\mathbf{Q} = \mathbf{Q}_n\mathbf{Q}_{n-1}\mathbf{Q}_{n-2}\cdots\mathbf{Q}_1, \quad \mathbf{R} = \mathbf{A}_n, \quad \mathbf{\Pi} = \mathbf{\Pi}_n\mathbf{\Pi}_{n-1}\mathbf{\Pi}_{n-2}\cdots\mathbf{\Pi}_1.$$

This algorithm is well understood, and it is ubiquitous in numerical computations. For exotic matrices, it can produce factorizations that are quite far from optimal (Kahan 1966, Sec. 5), but it typically works very well for many tasks. For instance, it serves for revealing the numerical rank of a matrix or for solving an ill-conditioned linear system. However, a serious drawback to this algorithm is that it fundamentally consists of a sequence of $n - 1$ steps (or n steps if $m > n$), where a large part of the matrix is updated in each step.

16.6. **A randomized algorithm for computing a CPQR decomposition.** Our next objective is to recast the algorithm for computing a CPQR decomposition that was introduced in Section 16.5 so that it works with "panels" of b contiguous columns, as shown in Figure 4. The difficulty is to find a set of b pivot vectors without updating the matrix between each selection. Fortuitously, the randomized algorithm for interpolatory decomposition (Section 13.4) is well adapted for this task. Indeed, a set of b columns that forms a good basis for the column space also forms a good set of pivot vectors.

To be specific, let us describe how to pick the first group of b pivot columns for an $m \times n$ matrix $\mathbf{A}$. Adapting the ideas in Section 13.4, we draw a Gaussian random matrix $\mathbf{\Omega}$ of size

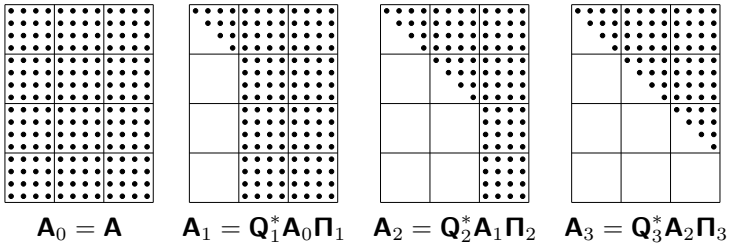


FIGURE 4. The sparsity pattern of a matrix $\mathbf{A}$ consisting of 4×3 blocks, each of size 3×3, as it undergoes the blocked version of the Householder QR algorithm described in Section 16.6. Each matrix $\mathbf{Q}_j$ is a product of three Householder reflectors. The difficulty in building an algorithm of this type is to find groups of pivot vectors *before* applying the corresponding Householder reflectors.

$(b + p) \times m$, where p is a small oversampling parameter. We form a sample matrix

$$\underset{(b+p) \times n}{\mathbf{Y}} = \underset{(b+p) \times m}{\mathbf{\Omega}} \quad \underset{m \times n}{\mathbf{A}},$$

and then we execute b steps of column-pivoted QR on the matrix $\mathbf{Y}$ (either Householder or Gram–Schmidt is fine for this step). The resulting b pivot columns turn out to be good pivot columns for $\mathbf{A}$ as well. Once these b columns have been moved to the front of $\mathbf{A}$, we perform a local CPQR factorization of this panel. We update the remaining $n - b$ columns using the computed Householder reflectors.

We could then proceed using exactly the same method a second time: draw a $(b+p) \times (m-b)$ Gaussian random matrix $\mathbf{\Omega}$, form a $(b + p) \times (n - b)$ sample matrix $\mathbf{Y}$, perform classical CPQR on $\mathbf{Y}$, and so on. However, there is a shortcut. We can update the sample matrix that was used in the first step, which renders the overhead cost induced by randomization almost negligible (Duersch and Gu 2017, Sec. 4).

Extensive numerical work has demonstrated dramatic acceleration over deterministic algorithms. Figure 5 draws on data from Martinsson et al. (2017) that illustrates the acceleration over a state-of-the-art software implementation of the classical CPQR method. Computer experiments also show that the randomized scheme chooses pivot columns whose quality is

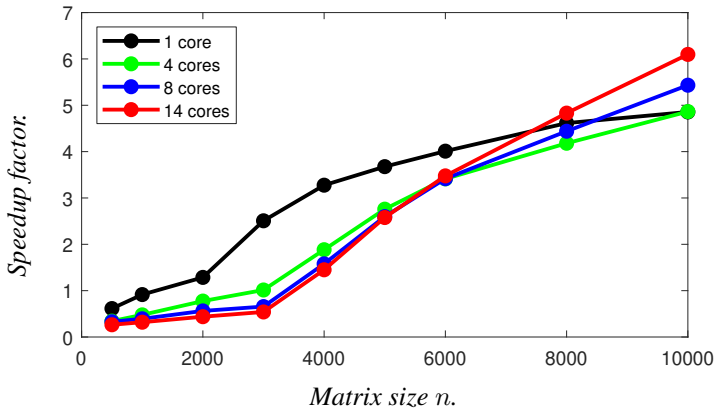


FIGURE 5. Speedup of the randomized algorithm for computing a column-pivoted QR decomposition described in Section 16.6, relative to LAPACK's faster routine (dgeqp3) as implemented in the Intel MKL library (version 11.2.3), running on an Intel Xeon E5-2695 v3 processor.

almost indistinguishable from those chosen by traditional pivoting, in the sense that the relation (16.2) holds to about the same accuracy. (However, the diagonal entries of $\mathbf{R}$ do not strictly decay in magnitude across the block boundaries.)

To understand the behavior of the algorithm, it is helpful to think about two extreme cases. In the first, suppose that the singular values of $\mathbf{A}$ decay very rapidly. Here, the analysis in Section 11 can be modified to show that, for any j, the first j pivot columns chosen by the randomized algorithm is likely to span the column space nearly as well as the optimal set of j columns. Therefore, they are excellent pivot vectors. At the other extreme, suppose that the singular values of $\mathbf{A}$ hardly decay at all. In this case, the randomized method may pick a completely different set of pivot vectors than the deterministic method, but this outcome is unproblematic because we can take any group of columns as pivot vectors. Of course, the interesting cases are intermediate between these two extremes. It turns out that the randomized methods work well regardless of how rapidly the singular values decay. For a detailed analysis, see (Duersch and Gu 2017, Xiao et al. 2017, Melgaard and Gu 2015, Feng, Xiao and Gu 2018).

Remark 16.1 (History)**.** *Finding a blocked version of the CPQR algorithm described in Section 16.5 has remained an open challenge in NLA for some time (Bischof and Quintana-Ortí 1998, Demmel, Grigori, Gu and Xiang 2015). The randomized technique described in this section was introduced in Martinsson (2015), while the updating technique was described in Duersch and Gu (2015). For full details, see (Martinsson et al. 2017) and (Duersch and Gu 2017).*

16.7. **A randomized algorithm for computing a URV decomposition.** In this section, we describe an incremental randomized algorithm for computing the URV factorization (16.1). Let us recall that for $\mathbf{A} \in \mathbb{F}^{m\times n}$, with $m \geq n$, this factorization takes the form

$$\underset{m\times n}{\mathbf{A}} = \underset{m\times n}{\mathbf{U}} \quad \underset{n\times n}{\mathbf{R}} \quad \underset{n\times n}{\mathbf{V}^*}, \tag{16.6}$$

where $\mathbf{U}$ and $\mathbf{V}$ have orthonormal columns and where $\mathbf{R}$ is upper-triangular.

The algorithm we describe is blocked, and executes efficiently on modern computing platforms. It is similar in speed to the randomized CPQR described in Section 16.6, and it shares the advantage that the decomposition is built incrementally so that the process can be stopped once a requested accuracy has been met. However, the URV factorization offers compelling advantages: (1) It is almost as good at revealing the numerical rank as the SVD (unlike the CPQR). (2) The URV factorization provides us with orthonormal basis vectors for both the column and the row spaces. (3) The off-diagonal entries of $\mathbf{R}$ are very small in magnitude. (4) The diagonal entries of $\mathbf{R}$ form excellent approximations to the singular values of $\mathbf{A}$.

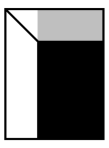
The randomized algorithm for computing a URV factorization we present follows the same algorithmic template as the randomized CPQR described in Section 16.6. It drives $\mathbf{A}$ to upper-triangular form one block at a time, but it replaces the permutation matrices $\mathbf{\Pi}_j$ in the CPQR with general unitary matrices $\mathbf{V}_j$. Using this increased freedom, we can obtain a factorization where all diagonal blocks are themselves diagonal matrices and all off-diagonal elements have small magnitude. Figure 6 summarizes the process.

To provide details on how the algorithm works, suppose that we are given an $m \times n$ matrix $\mathbf{A}$ and a block size b. In the first step of the process, our objective is then to build unitary matrices $\mathbf{U}_1$ and $\mathbf{V}_1$ such that

$$\mathbf{A} = \mathbf{U}_1 \mathbf{A}_1 \mathbf{V}_1^*,$$

where $\mathbf{A}_1$ has the block structure

$$\mathbf{A}_1 = \begin{bmatrix} \mathbf{A}_{1,11} & \mathbf{A}_{1,12} \\ \mathbf{0} & \mathbf{A}_{1,22} \end{bmatrix} = \quad ,$$



so that the $b \times b$ matrix $\mathbf{A}_{1,11}$ is diagonal and the entries of $\mathbf{A}_{1,12}$ are small in magnitude. To build $\mathbf{V}_1$, we use the randomized power iteration described in Section 11.6 to find a basis that

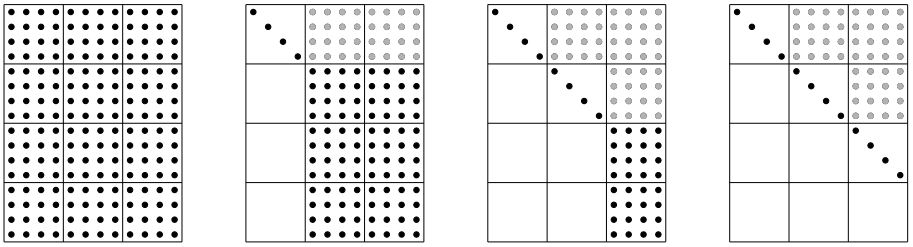


FIGURE 6. Sparsity pattern of a matrix being driven to upper-triangular form in the randomized URV factorization algorithm described in Section 16.7; see Figure 4. The matrices $\mathbf{U}_i$ and $\mathbf{V}_i$ are now more general unitary transforms (consisting in the bulk of Householder reflectors). The entries shown as gray are nonzero, but are typically very small in magnitude.

approximately spans the same space as the top b right singular vectors of $\mathbf{A}$. To be precise, we form the sample matrix

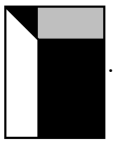
$$\mathbf{Y} = \mathbf{\Omega}\mathbf{A}\big(\mathbf{A}^*\mathbf{A}\big)^q,$$

where $\mathbf{\Omega}$ is a Gaussian random matrix of size $b \times m$ and where q is a parameter indicating the number of steps of power iteration taken. We then perform an unpivoted QR factorization of the *rows* of $\mathbf{Y}$ to form a matrix $\widetilde{\mathbf{V}}$ whose first b columns form an orthonormal basis for the column space of $\mathbf{Y}$. We then apply $\widetilde{\mathbf{V}}$ from the right to form the matrix $\mathbf{A}\widetilde{\mathbf{V}}$, and we perform an unpivoted QR factorization of the first b columns of $\mathbf{A}\widetilde{\mathbf{V}}$. This results in a new matrix

$$\widetilde{\mathbf{A}} = \big(\widetilde{\mathbf{U}}\big)^* \, \mathbf{A} \, \widetilde{\mathbf{V}}$$

that has the block structure

$$\widetilde{\mathbf{A}} = \begin{bmatrix} \widetilde{\mathbf{A}}_{1,11} & \widetilde{\mathbf{A}}_{1,12} \\ \mathbf{0} & \widetilde{\mathbf{A}}_{1,22} \end{bmatrix} = \quad .$$



The top left $b \times b$ block $\widetilde{\mathbf{A}}_{1,11}$ is upper-triangular, and the bottom left block is zero. The entries of $\widetilde{\mathbf{A}}_{1,12}$ are typically small in magnitude. Next, we compute a full SVD of the block $\widetilde{\mathbf{A}}_{1,11}$:

$$\widetilde{\mathbf{A}}_{1,11} = \widehat{\mathbf{U}}\mathbf{D}_{11}\widehat{\mathbf{V}}^*.$$

This step is inexpensive because $\widetilde{\mathbf{A}}_{1,11}$ has size $b \times b$, where b is small. As a final step, we form the transformation matrices

$$\mathbf{U}_1 = \widetilde{\mathbf{U}} \begin{bmatrix} \widehat{\mathbf{U}} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{m-b} \end{bmatrix}, \qquad \text{and} \qquad \mathbf{V}_1 = \widetilde{\mathbf{V}} \begin{bmatrix} \widehat{\mathbf{V}} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{n-b} \end{bmatrix}$$

and set

$$\mathbf{A}_1 = \mathbf{U}_1^* \mathbf{A} \mathbf{V}_1.$$

The result of this process is that the diagonal entries of $\mathbf{D}_{11}$ typically give accurate approximations to the first b singular values of $\mathbf{A}$, and the "remainder" matrix $\mathbf{A}_{1,22}$ has spectral norm that is similar to σ_{b+1}. Thus,

$$\|\mathbf{A}_{1,22}\| \approx \inf\{\|\mathbf{A} - \mathbf{B}\| : \ \mathbf{B} \text{ has rank } b\}.$$

This process corresponds to the first step in Figure 6. The succeeding iterations execute the same process, at each step working on the remaining lower-right part of the matrix that has not yet been driven to upper-triangular form. We refer to Martinsson, Quintana Orti and Heavner (2019) and Martinsson (2018) for details.

From a theoretical point of view, the first step of the URV factorization described is well understood since it is mathematically equivalent to the randomized power iteration described

in Section 11.6. We observe that the first b columns of $\mathbf{U}_1$ do a better job of spanning the column space of $\mathbf{A}$ than the first b columns of $\mathbf{V}_1$ do for spanning the row space; the reason for this asymmetry is that by forming the product $\mathbf{AV}_1$, we in effect perform an additional step of the power iteration (Gopal and Martinsson 2018, Sec. 6).

An important feature of the method described in this section is that it is incremental, and it can be halted once a given computational tolerance has been met. This feature has been a key competitive advantage of the column-pivoted QR decomposition, and it is often cited as the motivation for using CPQR. The method described in this section has almost all the advantages of the randomized Householder CPQR factorization (it is blocked, it is incremental, and it executes very fast in practice), while resulting in a factorization that is far closer to optimal in revealing the rank.

Remark 16.2 (Related work)**.** *The idea of loosening the requirements on the factors in a rank-revealing factorization and searching for a decomposition such as (16.6) is well explored in the literature (Fierro, Hansen and Hansen 1999, Stewart 1994, Park and Eldén 1995) and (Stewart 1998). Deterministic techniques for computing the URV decomposition are described in (Fierro et al. 1999, Stewart 1999); these algorithms combine some of the appealing qualities of the SVD (high accuracy in revealing the rank) with some of the appealing qualities of CPQR (the possibility of halting the execution once a requested tolerance has been met). However, they were not blocked, and therefore they were subject to the same liabilities as deterministic algorithms for computing the SVD and the CPQR. A more recent use of randomization in this context is described in (Feng et al. 2018).*

17. General linear solvers

Researchers are currently exploring randomized algorithms for solving linear systems, such as

$$\mathbf{Ax} = \mathbf{b}, \tag{17.1}$$

where $\mathbf{A}$ is a given coefficient matrix and $\mathbf{b}$ is a given vector. This section describes a few probabilistic approaches for solving (17.1). For the most part, we restrict our attention to the case where $\mathbf{A}$ is square and the system is consistent, but we will also touch on linear regression problems. Research on randomized linear solvers has not progressed as rapidly as some other areas of randomized NLA, so the discussion here is more preliminary than other parts of this survey.

17.1. **Background: Iterative solvers.** It is important to keep in mind that existing iterative solvers often work exceptionally well. Whenever $\mathbf{A}$ is well-conditioned or, more generally, whenever its spectrum is "clustered," Krylov solvers such as the conjugate gradient (CG) algorithm or GMRES tend to converge very rapidly. For practical purposes, the cost of solving (17.1) is no larger than the cost of a handful of matrix–vector multiplications with $\mathbf{A}$. In terms of speed, it is very difficult to beat these techniques. Consequently, we focus on the cases where known iterative methods converge slowly and where we cannot deploy standard preconditioners to resolve the problem.

Having limited ourselves to this situation, the choice of solver for (17.1) will depend on properties of the coefficient matrix: Is it dense or sparse? Does it fit in RAM? Do we have access to individual matrix entries? Can we apply $\mathbf{A}$ to a vector? We will consider several of these environments.

17.2. **Accelerating solvers based on dense matrix factorizations.** As it happens, one of the early examples of randomization in NLA was a method for accelerating the solution of a dense linear system (17.1). Parker (1995) observed that we can precondition a linear system by left and right multiplying the coefficient matrix by random unitary matrices $\mathbf{U}$ and $\mathbf{V}$. With probability 1, we can solve the resulting system

$$\big(\mathbf{UAV}^*\big)\,\big(\mathbf{Vx}\big) = \mathbf{Ub} \tag{17.2}$$

by Gaussian elimination *without pivoting*. More precisely, Parker proved that, almost surely, blocked Gaussian elimination will not encounter a degenerate diagonal block.

Blocked Gaussian elimination without pivoting is substantially faster than ordinary Gaussian elimination for two reasons: matrix operations are more efficient than vector operations on modern computers, and we avoid the substantial communication costs that arise when we search for pivots. (Section 16.2 contains more discussion about blocking.) Parker also observed that structured random matrices (such as the randomized trigonometric transforms from Section 9.3) allow us to perform the preconditioning step at lower cost than the subsequent Gaussian elimination procedure.

Parker (1995) inspired many subsequent papers, including (Baboulin, Li and Rouet 2014, Trogdon 2017, Demmel et al. 2012, Baboulin, Dongarra, Rémy, Tomov and Yamazaki 2017) and (Pan and Zhao 2017). Another related direction concerns the smoothed analysis of Gaussian elimination undertaken in (Sankar, Spielman and Teng 2006).

As we saw in Section 16, randomization can be used to accelerate the computation of rank-revealing factorizations of the matrix $\mathbf{A}$. In this context, randomness allows us to block the factorization method, which increases its practical speed, even though the overall arithmetic cost remains at $O(n^3)$. Randomized rank-revealing factorizations are ideal for solving ill-conditioned linear systems because they allow the user to stabilize the computation by avoiding subspaces associated with small singular values.

For instance, suppose that we have computed a singular value decomposition (SVD):

$$\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^* = \sum_{j=1}^{n} \sigma_j \, \mathbf{u}_j \mathbf{v}_j^*.$$

Let us introduce a truncation parameter ε and ignore all singular modes where $\sigma_j \leq \varepsilon$. Then the stabilized solution to (17.1) is

$$\mathbf{x}_\varepsilon = \sum_{j \,:\, \sigma_j > \varepsilon} \frac{1}{\sigma_j} \, \mathbf{v}_j \mathbf{u}_j^* \mathbf{b}.$$

By allowing the residual to take a nonzero value, we can ensure that $\mathbf{x}_\varepsilon$ does not include large components that contribute little toward satisfying the original equation. The randomized URV decomposition, described in Section 16.7, can also be used for stabilization, and we can compute it much faster than an SVD.

Remark 17.1 (Are rank-revealing factorizations needed?)**.** *In some applications, computing a rank-revealing factorization is overkill for purposes of solving the linear system (17.1). In particular, if we compute an unpivoted QR decomposition of $\mathbf{A}$, then it is easy to block both the factorization and the solve stages so that very high speed is attained. This process is provably backwards stable, which is sometimes all that is needed. (In practice, partially pivoted LU can often be used in an analogous manner, despite being theoretically unstable.)*

In contrast, when the actual entries of the computed solution $\mathbf{x}_{\mathrm{approx}}$ matter (as opposed to the value of $\mathbf{A}\mathbf{x}_{\mathrm{approx}}$), a stabilized solver is generally preferred. As a consequence, column-pivoted QR is often cited as a method of choice for ill-conditioned problems in situations where an SVD is not affordable.

Remark 17.2 (Strassen accelerated solvers)**.** *We saw in Section 16.4 that randomization has enabled us to compute a rank-revealing factorization of an $n \times n$ matrix in less than $\mathcal{O}(n^3)$ operations. The idea was to use randomized preconditioning as in (17.2), and then accelerate an unpivoted factorization of the resulting coefficient matrix using fast algorithms for the matrix-matrix multiplication such as Strassen (Demmel et al. 2007). This methodology can of course be immediately applied to the task of solving ill-conditioned linear systems. For improved numerical stability, a few steps of power iteration can be incorporated to this approach; see* (16.4).

17.3. **Sketch and precondition.** Another approach to preconditioning is to look for a random transformation of the linear system that makes an iterative linear solver converge more quickly. Typically, these preconditioning transforms need to cluster the eigenvalues of the matrix.

The most successful example of this type of randomized preconditioning does not concern square systems, but rather highly overdetermined least-squares problems. See Section 10.5

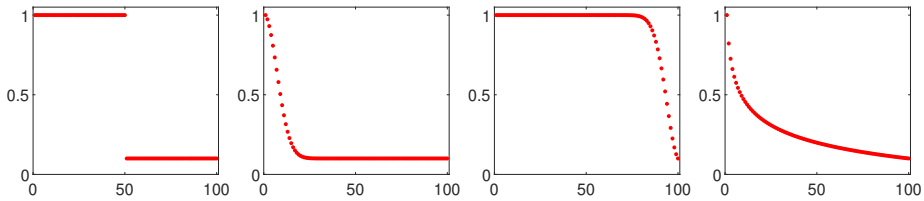


FIGURE 7. The eigenvalues of four different PD matrices that all have condition number 10 (since $\lambda_{\max} = 1$ and $\lambda_{\min} = 0.1$). As discussed in Section 17.3, the difficulty of solving the corresponding linear systems using conjugate gradients differ significantly between these cases. (a) For this matrix, CG converges in two iterations, without the need for preconditioners. (b) When the spectrum has some large outliers, the randomized preconditioner outlined in Section 17.3 works well. (c,d) Finding randomized preconditioners for matrices with spectra like these remains an open research problem.

et seq. for a discussion of this idea. This type of randomized preconditioning can greatly enhance the robustness and power of "asynchronous" solvers for communication-constrained environments (Avron, Druinsky and Gupta 2015). Related techniques for kernel ridge regression are described in Avron, Clarkson and Woodruff (2017). For linear systems involving high-dimensional tensors, see Kressner, Steinlechner and Vandereycken (2016).

For square linear systems, the search for randomized preconditioners has been less fruitful. Section 18 outlines the main success story. Nevertheless, techniques already at hand can be very helpful for solving linear systems in special situations, which we illustrate with a small example.

Consider the task of solving (17.1) for a positive-definite (PD) coefficient matrix $\mathbf{A}$. In this environment, the iterative method of choice is the conjugate gradient (CG) algorithm (Hestenes and Stiefel 1952). A detailed convergence analysis for CG is available; for example, see (Trefethen and Bau III 1997, Sec. 38). In a nutshell, CG converges rapidly when the eigenvalues of $\mathbf{A}$ are clustered, as in Figure 7(a). Therefore, our task is to find a matrix $\mathbf{M}$ for which $\mathbf{M}^{-1}$ can be applied rapidly to vectors and for which $\mathbf{M}^{-1/2}\mathbf{A}\mathbf{M}^{-1/2}$ has a tightly clustered spectrum.

In a situation where $\mathbf{A}$ has a few eigenvalues that are larger than the others (Figure 7(b)), randomized algorithms for low-rank approximation provide excellent preconditioners. For instance, we can use the randomized Nyström method (Section 14) to compute an approximation

$$\mathbf{A} \approx \mathbf{U}\mathbf{D}\mathbf{U}^* \tag{17.3}$$

where $\mathbf{D} \in \mathbb{R}_+^{k\times k}$ is a diagonal matrix whose entries hold approximations to the largest k eigenvalues of $\mathbf{A}$, and where $\mathbf{U} \in \mathbb{F}^{m\times k}$ is an orthonormal matrix holding the corresponding approximate eigenvectors. We then form a preconditioner for $\mathbf{A}$ by setting

$$\mathbf{M} = (1/\alpha)\,\mathbf{U}\mathbf{D}\mathbf{U}^* + \big(\mathbf{I} - \mathbf{U}\mathbf{U}^*\big).$$

It is trivial to invert $\mathbf{M}$ because $\mathbf{M}^{-1} = \alpha\mathbf{U}\mathbf{D}^{-1}\mathbf{U}^* + \big(\mathbf{I} - \mathbf{U}\mathbf{U}^*\big)$. Now, if (17.3) captured the top k eigenmodes of $\mathbf{A}$ exactly, then the preconditioned coefficient matrix $\mathbf{M}^{-1/2}\mathbf{A}\mathbf{M}^{-1/2}$ would have the same eigenvectors as $\mathbf{A}$, but with the top k eigenvalues replaced by α and the remaining eigenvalues unchanged. By setting $\alpha = \lambda_k$, say, the spectrum of $\mathbf{M}^{-1/2}\mathbf{A}\mathbf{M}^{-1/2}$ would become far more tightly clustered. In reality, the columns of $\mathbf{U}$ do not exactly align with the eigenvectors of $\mathbf{A}$. Even so, the accuracy will be good for the eigenvectors associated with the top eigenvalues, which is what matters.

17.4. **The randomized Kaczmarz method and its relatives.** The Kaczmarz method is an iterative algorithm for solving linear systems that is typically used for large, overdetermined problems with inconsistent equations. Randomized variants of the Kaczmarz method have

received a lot of attention in recent years, in part because of close connections to stochastic gradient descent (SGD) algorithms for solving least-squares problems.

To explain the idea, consider a (possibly inconsistent) linear system

$$\mathbf{A}^*\mathbf{x} \approx \mathbf{b} \quad \text{where} \quad \mathbf{A}^* \in \mathbb{F}^{m\times n}. \tag{17.4}$$

The basic Kaczmarz algorithm starts with an initial guess $\mathbf{x}_0 \in \mathbb{F}^n$ for the solution. At each iteration t, we select a new index $j = j(t) \in \{1, \dots, m\}$, and we make the update

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \frac{\mathbf{b}(j) - \langle \mathbf{A}(:,j), \mathbf{x}_t\rangle}{\|\mathbf{A}(:,j)\|^2}\mathbf{A}(:,j). \tag{17.5}$$

The rule (17.5) has a simple interpretation: it ensures that $\mathbf{x}_{t+1}$ is the closest point to $\mathbf{x}_t$ in the hyperplane containing solutions to the linear equation determined by the jth equation in the system.

In implementing this method, we must choose a control mechanism that determines the next index. A simple and robust approach is to cycle through the rows consecutively; that is, $j(t) = t \bmod m$. Another effective, but expensive, option is to select the equation with the largest violation.

The randomized Kaczmarz (RK) algorithm uses a probabilistic control mechanism instead. This kind of approach also has a long history, and it is useful in cases where cyclic control is ineffective. The RK method has received renewed attention owing to work of Strohmer and Vershynin (2009). They proposed sampling each $j(t)$ independently at random, with the probability of choosing the ith equation proportional to the squared ℓ_2 norm of the ith column of $\mathbf{A}$. They proved that this version of RK converges linearly with a rate determined by the (Demmel) condition number of the matrix $\mathbf{A}$. Later, it was recognized that this approach is just a particular instantiation of SGD for the least-squares problem (17.4). See Needell, Ward and Srebro (2014), which draws on results from Moulines and Bach (2011).

There are many subsequent papers that have built on the RK approach for solving inconsistent linear systems. Leventhal and Lewis (2010) observed that related ideas can be used to design randomized Gauss–Seidel and randomized Jacobi iterations. Needell and Tropp (2014) studied a blocked version of the RK algorithm, which is practically more efficient for many of the same reasons that other blocked algorithms work well (Section 16.2).

Gower and Richtarik (2015*a*) observed that the RK algorithm is a particular type of iterative sketching method. Based on this connection, they proposed a generalization. At each iteration, we draw an independent random embedding $\mathbf{S}_t \in \mathbb{F}^{\ell\times m}$. The next iterate is chosen by solving the least-squares problem

$$\mathbf{x}_t = \arg\min_{\mathbf{y}} \|\mathbf{x}_{t-1} - \mathbf{y}\|^2 \quad \text{subject to} \quad \mathbf{S}_t\mathbf{A}^*\mathbf{y} = \mathbf{S}_t\mathbf{b}. \tag{17.6}$$

The idea is to choose the dimension ℓ sufficiently small that the sketched least-squares problem can be solved explicitly using a direct method (e.g., QR factorization). This flexibility leads to algorithms that converge more rapidly in practice because the sketch $\mathbf{S}_t$ can mix equations instead of just sampling. Later, Richtárik and Takáč (2017) showed that this procedure can be accelerated to achieve rates that depend on the *square root* of an appropriate condition number; see also Gower, Hanzely, Richtarik and Stich (2018).

18. Linear solvers for graph Laplacians

In this section we describe the randomized algorithm, SparseCholesky, which can efficiently solve a linear system whose coefficient matrix is a graph Laplacian matrix. Up to a small logarithmic factor, this method achieves the minimum possible runtime and storage costs. This algorithm has the potential to accelerate many types of computations involving graph Laplacian matrices.

The SparseCholesky algorithm was developed by Kyng and Sachdeva (2016) and further refined by Kyng (2017). These approaches are based on earlier work from Dan Spielman's group, notably the paper of Kyng, Lee, Peng, Sachdeva and Spielman (2016). The presentation here is adapted from Tropp (2019).

18.1. **Overview.** We begin with a high-level approach for solving Laplacian linear systems. The basic idea is to construct a preconditioner using a randomized variant of the incomplete Cholesky method. Then we can solve the original linear system by means of the preconditioned conjugate gradient (PCG) algorithm.

18.1.1. *Approximate solutions to the Poisson problem.* Let $\mathbf{L} \in \mathbb{H}_n(\mathbb{R})$ be the Laplacian matrix of a weighted, loop-free, undirected graph on the vertex set $V = \{1, \dots, n\}$. We write m for the number of edges in the graph, i.e., the sparsity of the graph. For simplicity, we will also assume that the graph is *connected*; equivalently, $\ker(\mathbf{L}) = \operatorname{span}\{\mathbf{1}\}$ where $\mathbf{1} \in \mathbb{R}^n$ is the vector of ones. See Section 7.4 for definitions. See Figure 8 for an illustration of an unweighted, undirected graph.

The basic goal is to find the unique solution $\mathbf{x}_\star$ to the Poisson problem

$$\mathbf{L}\mathbf{x} = \mathbf{f} \quad \text{where} \quad \mathbf{1}^*\mathbf{f} = 0 \quad \text{and} \quad \mathbf{1}^*\mathbf{x} = 0.$$

For a parameter $\varepsilon > 0$, we can relax this requirement by asking instead for an approximate solution $\mathbf{x}_\varepsilon$ that satisfies

$$\|\mathbf{x}_\varepsilon - \mathbf{x}_\star\|_{\mathbf{L}} \le \varepsilon \, \|\mathbf{x}_\star\|_{\mathbf{L}}.$$

We have written $\|\mathbf{x}\|_{\mathbf{L}} := (\mathbf{x}^*\mathbf{L}\mathbf{x})^{1/2}$ for the energy seminorm induced by the Laplacian matrix.

18.1.2. *Approximate Cholesky decomposition.* Imagine that we can efficiently construct a sparse, approximate Cholesky decomposition of the Laplacian matrix $\mathbf{L}$. More precisely, we seek a morally lower-triangular matrix $\mathbf{C}$ that satisfies

$$0.5\,\mathbf{L} \preccurlyeq \mathbf{C}\mathbf{C}^* \preccurlyeq 1.5\,\mathbf{L} \quad \text{where} \quad \mathtt{nnz}(\mathbf{C}) = O(m \log n). \tag{18.1}$$

In other words, there is a known permutation of the rows that brings the matrix $\mathbf{C}$ into lower-triangular form. As usual, $\preccurlyeq$ is the semidefinite order, and $\mathtt{nnz}$ returns the number of nonzero entries in a matrix.

This section describes an algorithm, called SPARSECHOLESKY, that can complete the task outlined in the previous paragraph. This algorithm is motivated by the insight that we can produce a sparse approximation of a Laplacian matrix by random sampling (Section 7.4). The main challenge is to obtain sampling probabilities without extra computation. The resulting method can be viewed as a randomized variant of the incomplete Cholesky factorization (Golub and Van Loan 2013, Sec. 11.5.8).

18.1.3. *Preconditioning.* Given the sparse, approximate Cholesky factor $\mathbf{C}$, we can precondition the Poisson problem:

$$(\mathbf{C}^\dagger \mathbf{L} \mathbf{C}^{*\dagger})(\mathbf{C}^*\mathbf{x}) = (\mathbf{C}^\dagger \mathbf{f}). \tag{18.2}$$

When (18.1) holds, the matrix $\mathbf{C}^\dagger \mathbf{L} \mathbf{C}^{*\dagger}$ has condition number $\kappa \le 3$.

Therefore, we can solve the preconditioned system (18.2) quickly using the PCG algorithm (Golub and Van Loan 2013, Sec. 11.5). If the initial iterate $\mathbf{x}_0 = \mathbf{0}$, then j steps of PCG produce an iterate $\mathbf{x}_j$ that satisfies

$$\|\mathbf{x}_j - \mathbf{x}_\star\|_{\mathbf{L}} \le 2 \left[\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right]^j \|\mathbf{x}_\star\|_{\mathbf{L}} < 3^{1-j} \|\mathbf{x}_\star\|_{\mathbf{L}}.$$

As a consequence, we can achieve relative error ε after $1 + \log_3(1/\varepsilon)$ iterations. Each iteration requires a matrix–vector product with $\mathbf{L}$ and the solution of a (consistent) linear system $\mathbf{C}\mathbf{C}^*\mathbf{u} = \mathbf{y}$. We can perform these steps in $O(m \log n)$ operations per iteration. Indeed, $\mathbf{L}$ has only $O(m)$ nonzero entries. The matrix $\mathbf{C}$ is morally triangular with $O(m \log n)$ nonzero entries, so we can apply $(\mathbf{C}\mathbf{C}^*)^\dagger$ using two triangular solves.

Remark 18.1 (Triangular solve)**.** *To solve a consistent linear system $(\mathbf{C}\mathbf{C}^*)\mathbf{u} = \mathbf{y}$, we can apply $\mathbf{C}^\dagger$ by triangular elimination and then apply $\mathbf{C}^{*\dagger}$ by triangular elimination. The four subspace theorem ensures that solution to the first problem renders the second problem consistent too. Since $\ker(\mathbf{C}\mathbf{C}^*) = \operatorname{span}\{\mathbf{1}\}$, we can enforce consistency numerically by removing the constant component of the input $\mathbf{y}$. Similarly, we can remove the constant component of the output $\mathbf{u}$ to ensure it belongs to the correct space.*

18.1.4. *Main results.* The following theorem describes the performance of the SPARSECHOLESKY procedure. This is the main result from Kyng and Sachdeva (2016).

Theorem 18.2 (SparseCholesky)**.** *Let* L *be the Laplacian of a connected graph on* n *vertices, with* m *weighted edges. With high probability, the* SPARSECHOLESKY *algorithm produces a morally lower-triangular matrix* C *that satisfies*

$$0.5\,\mathsf{L} \preccurlyeq \mathsf{C}\mathsf{C}^* \preccurlyeq 1.5\,\mathsf{L}.$$

The matrix C *has* $O(m \log n)$ *nonzero entries. The expected running time is* $O(m \log^2 n)$ *operations.*

In view of our discussion about PCG, we arrive at the following statement about solving the Poisson problem.

Corollary 18.3 (Poisson problem)**.** *Suppose the* SPARSECHOLESKY *algorithm delivers an approximation* $\mathsf{L} \approx \mathsf{C}\mathsf{C}^*$ *that satisfies* (18.1)*. Then we can solve each consistent linear system* $\mathsf{L}\mathsf{x} = \mathsf{f}$ *to relative error* ε *in the seminorm* $\|\cdot\|_{\mathsf{L}}$ *using at most* $1 + \log_3(1/\varepsilon)$ *iterations of PCG, each with a cost of* $O(m \log n)$ *arithmetic operations.*

18.1.5. *Discussion.* As we have mentioned, the Poisson problem serves as a primitive for undertaking many computations on undirected graphs (Teng 2010). Potential applications include clustering, analysis of random walks, and finite-element discretizations of elliptic PDEs.

The SPARSECHOLESKY algorithm achieves a near-optimal runtime and storage guarantee for the Poisson problem on a graph. Indeed, for a general graph with m edges, any algorithm must use $O(m)$ storage and arithmetic. There is a proof that the cost $O(m \log^{1/2}(n))$ is achievable in theory (Cohen, Kyng, Miller, Pachocki, Peng, Rao and Xu 2014), but the resulting methods are currently impractical. Meanwhile, the simplicity of the SPARSECHOLESKY method makes it a candidate for real-world computation.

For particular classes of Laplacian matrices, existing solvers can be very efficient. Optimized sparse direct solvers (Davis, Rajamanickam and Sid-Lakhdar 2016) work very well for small- and medium-size problems, but they typically have superlinear scaling, which renders them unsuitable for truly large-scale problems. Iterative methods such as multigrid or preconditioned Krylov solvers can attain linear complexity for important classes of problems, in particular for sparse systems arising from the discretization of elliptic PDEs. However, we are not aware of competing methods that provably enjoy near-optimal complexity for all problems.

We regard the SPARSECHOLESKY algorithm as one of the most dramatic examples of how randomization has the potential to accelerate basic linear algebra computations, both in theory and in practice.

18.2. **Cholesky decomposition of a graph Laplacian.** To begin our explanation of the SPARSECHOLESKY algorithm, let us summarize what happens when we apply the standard Cholesky decomposition method to a graph Laplacian.

18.2.1. *The Laplacian of a multigraph.* For technical reasons, related to the design and analysis of the algorithm, we need to work with multigraphs instead of ordinary graphs. In the discussion, we will point out specific places where this generality is important.

Consider a weighted, undirected *multigraph* G, defined on the vertex set $V = \{1, \dots, n\}$. Each edge $e = \{u, v\}$ is an unordered pair of vertices; we typically use the abbreviated notation $e = uv = vu$. We introduce a weight function w_G that assigns a positive weight to each edge e in the multigraph G. Since G is a multigraph, there may be many multiedges, with distinct weights, connecting the same two vertices.

Taking some notational liberties, we will identify the multigraph G with its Laplacian matrix L, which we express in the form

$$\mathsf{L} = \sum\nolimits_{e \in \mathsf{L}} w_{\mathsf{L}}(e)\, \mathbf{\Delta}_e. \tag{18.3}$$

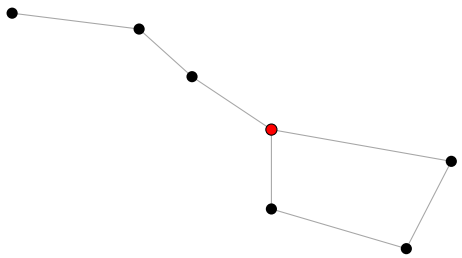


FIGURE 8. *A combinatorial graph.* The Ursa Major graph with a distinguished vertex (the star Megrez) highlighted in red.

The matrix $\mathbf{\Delta}_e$ is the elementary Laplacian on the vertex pair (u, v) that composes the edge $e = uv$. That is,

$$\mathbf{\Delta}_e := \mathbf{\Delta}_{uv} := (\boldsymbol{\delta}_u - \boldsymbol{\delta}_v)(\boldsymbol{\delta}_u - \boldsymbol{\delta}_v)^* \quad \text{where } e = uv.$$

The sum in (18.3) takes place over all the multiedges e in the multigraph $\mathbf{L}$, so the same elementary Laplacian may appear multiple times with different weights.

18.2.2. *Stars and cliques.* To describe the Cholesky algorithm on a graph, we need to introduce a few more concepts from graph theory. Define the *degree* and the *total weight* of a vertex u in the multigraph $\mathbf{L}$ to be

$$\deg_{\mathbf{L}}(u) := \sum\nolimits_{e=uv\in\mathbf{L}} 1 \quad \text{and} \quad w_{\mathbf{L}}(u) := \sum\nolimits_{e=uv\in\mathbf{L}} w_{\mathbf{L}}(e).$$

In other words, the degree of u is the total number of multiedges e that contain u. The total weight of u is the sum of the weights of the multiedges e that contain u.

Let u be a fixed vertex. The *star* induced by u is the Laplacian

$$\textsc{star}(u, \mathbf{L}) := \sum\nolimits_{e=uv\in\mathbf{L}} w_{\mathbf{L}}(e)\, \mathbf{\Delta}_e.$$

In words, the star includes precisely those multiedges e in the multigraph $\mathbf{L}$ that contain the vertex u.

The *clique* induced by u is defined implicitly as the correction that occurs when we take the Schur complement (2.4) of the Laplacian with respect to the coordinate u:

$$\mathbf{L}/\boldsymbol{\delta}_u =: (\mathbf{L} - \textsc{star}(u, \mathbf{L})) + \textsc{clique}(u, \mathbf{L}).$$

Recall that $\boldsymbol{\delta}_u$ is the standard basis vector in coordinate u. By direct calculation, one may verify that

$$\textsc{clique}(u, \mathbf{L}) = \frac{1}{2w_{\mathbf{L}}(u)} \sum_{e_1=uv_1\in\mathbf{L}} \sum_{e_2=uv_2\in\mathbf{L}} w_{\mathbf{L}}(e_1) w_{\mathbf{L}}(e_2) \mathbf{\Delta}_{v_1 v_2}.$$

Each sum takes place over all multiedges e in $\mathbf{L}$ that contain the vertex u. It can be verified that the clique is also the Laplacian of a weighted multigraph.

Figures 8 and 9 contain an illustration of a (simple) graph, along with the star and clique induced by eliminating a vertex. In our more general setting, the edges in the star and clique would have associated weights. These diagrams are courtesy of Richard Kueng.

18.2.3. *Graphs and Cholesky.* With the notation introduced in Section 18.2.2, we can present the graph-theoretic interpretation of the Cholesky algorithm as it applies to the Laplacian $\mathbf{L}$ of a weighted multigraph.

Define the initial Laplacian $\mathbf{S}_0 := \mathbf{L}$. In each step $i = 1, 2, \ldots, n$, we select a new vertex u_i. We extract the associated column of the current Laplacian:

$$\mathbf{c}_i := \frac{1}{\sqrt{(\mathbf{S}_{i-1})_{u_i u_i}}} \mathbf{S}_{i-1} \boldsymbol{\delta}_{u_i}.$$

We compute the Schur complement with respect to the vertex u_i:

$$\mathbf{S}_i := \mathbf{S}_{i-1}/\boldsymbol{\delta}_{u_i} = (\mathbf{S}_{i-1} - \textsc{star}(u_i, \mathbf{S}_{i-1})) + \textsc{clique}(u_i, \mathbf{S}_{i-1}).$$

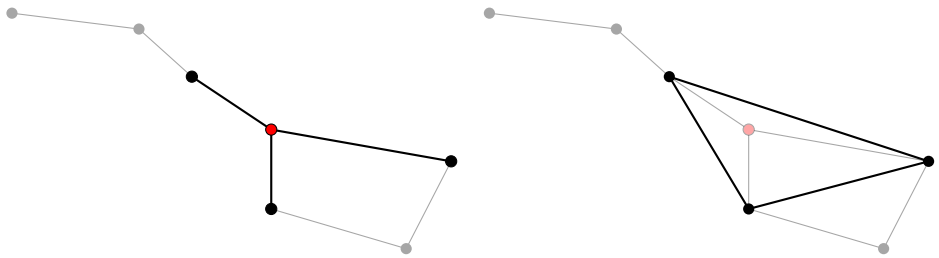


FIGURE 9. *Illustration of a star and clique in the Ursa Major graph.* (a) The star induced by the red vertex consists of the three solid black edges. (b) The clique induced by the red vertex consists of the three solid black edges. These edges are added when the red vertex is eliminated.

In other words, we remove the star induced by u_i and replace it with the clique induced by u_i. Each $\mathbf{S}_i$ is the Laplacian of a multigraph; it has no multiedge that contains any one of the vertices $u_1, \dots, u_i$. Therefore, we have reduced the size of the problem.

After n steps, the Cholesky factorization is determined by a vector $\boldsymbol{\pi} = (u_1, u_2, \dots, u_n)$ that holds the chosen indices and a matrix $\mathbf{C} = \begin{bmatrix} \mathbf{c}_1 & \dots & \mathbf{c}_n \end{bmatrix}$ such that $\mathbf{C}(\boldsymbol{\pi}, :)$ is lower-triangular. The algorithm ensures that we have the exact decomposition

$$\mathbf{L} = \mathbf{C}\mathbf{C}^*.$$

This is the (pivoted) Cholesky factorization of the Laplacian matrix.

To choose a vertex to eliminate, the classical approach is to find a vertex with minimum degree or with minimum total weight. Or one may simply select one of the remaining vertices at random.

18.2.4. *Computational costs.* The cost to compute a Cholesky factorization $\mathbf{L} = \mathbf{C}\mathbf{C}^*$ of a Laplacian matrix $\mathbf{L}$ is typically superlinear in n, with a worst-case cost of $O(n^3)$ arithmetic and $O(n^2)$ storage. This is the reason that $\mathbf{C}$ is less sparse than $\mathbf{L}$: the clique that is introduced at an elimination step has more edges than the star that it replaces, a phenomenon referred to as *fill-in*. The exact growth in the number of nonzero entries depends on the sparsity pattern of $\mathbf{L}$ and on the chosen elimination order. For special cases, using a nested dissection ordering can provably improve on the worst-case estimates (Davis et al. 2016). For instance, if $\mathbf{L}$ results from the finite-difference or finite-element discretization of an elliptic PDE, then $\mathtt{nnz}(\mathbf{C}) = O(n \log(n))$ in two dimensions and $\mathtt{nnz}(\mathbf{C}) = O(n^{4/3})$ in three.

For general graphs, one path towards improving the efficiency of the Cholesky factorization procedure is to randomly approximate the clique by sampling, in order to curb the fill-in. Section 7.4 already indicates that this innovation may be possible, provided that we can find a way to obtain sampling probabilities.

18.3. **The SPARSECHOLESKY algorithm.** We are now prepared to present the SPARSECHOLESKY procedure, which uses randomized sampling to compute a sparse, approximate Cholesky factorization.

18.3.1. *Procedure.* Let $\mathbf{L}$ be the Laplacian of a weighted multigraph on $V = \{1, \dots, n\}$. We perform the following steps.

(1) **Preprocessing.** Split each multiedge $e = uv$ in $\mathbf{L}$ into $R = \lceil 8 \log(en) \rceil$ multiedges, each connecting $\{u, v\}$, and each with weight $w_{\mathbf{L}}(e)/R$. The purpose of this step is to control the effective resistance (7.5) of each multiedge at the outset of the algorithm. Note that this splitting results in a weighted multigraph, even if we begin with a simple graph.

(2) **Initialization.** Form the initial Laplacian $\mathbf{S}_0 = \mathbf{L}$ and the list of remaining vertices $F_0 = V$.

(3) **Iteration.** For each $i = 1, 2, \dots, n$:

(a) **Select a vertex.** Choose a vertex u_i uniformly at random from F_{i-1}. Remove this vertex from the list: $F_i = F_{i-1} \setminus \{u_i\}$.

(b) **Extract the column.** Copy the normalized u_ith column from the current Laplacian:

$$\mathbf{c}_i = \frac{1}{\sqrt{(\mathbf{S}_{i-1})_{u_i u_i}}} \mathbf{S}_{i-1} \boldsymbol{\delta}_{u_i}.$$

Set $\mathbf{c}_i = \mathbf{0}$ if the denominator equals zero.

(c) **Sampling the clique.** Construct the Laplacian $\mathbf{K}_i$ of a random sparse approximation of $\textsc{clique}(u_i, \mathbf{S}_{i-1})$. We will detail this procedure in Section 18.3.2.

(d) **Approximate Schur complement.** Form

$$\mathbf{S}_i = (\mathbf{S}_{i-1} - \textsc{star}(u_i, \mathbf{S}_{i-1})) + \mathbf{K}_i.$$

(4) **Decomposition.** Collate the columns $\mathbf{c}_i$ into the Cholesky factor

$$\mathbf{C} = \begin{bmatrix} \mathbf{c}_1 & \dots & \mathbf{c}_n \end{bmatrix}.$$

Define the row permutation $\pi(i) = u_i$ for each i.

Once these operations are complete, $\mathbf{C}$ is a sparse, morally lower-triangular matrix. It is also very likely that $\mathbf{L} \approx \mathbf{C}\mathbf{C}^*$. Theorem 18.2 makes a rigorous accounting of these claims.

18.3.2. *Clique sampling.* The remaining question is how to construct a random approximation $\mathbf{K}$ of a clique $\textsc{clique}(u, \mathbf{S})$. Here is the procedure:

(1) **Probabilities.** Construct a probability mass $\mathbf{p}$ such that

$$p(e) = \frac{w_{\mathbf{S}}(e)}{w_{\mathbf{S}}(u)} \quad \text{for each } e \in \textsc{star}(u, \mathbf{S}).$$

(2) **Sampling.** For each $i = 1, \dots, d = \deg_{\mathbf{S}}(u)$,

(a) Draw a random multiedge $e_1 = uv_1$ from the multiedges in $\textsc{star}(u, \mathbf{S})$ according to the probability mass $\mathbf{p}$.

(b) Draw a second random multiedge $e_2 = uv_2$ from the multiedges in $\textsc{star}(u, \mathbf{S})$ according to the uniform distribution.

(c) Form the random Laplacian matrix of a new multiedge:

$$\mathbf{X}_i = \frac{w_{\mathbf{S}}(e_1)\, w_{\mathbf{S}}(e_2)}{w_{\mathbf{S}}(e_1) + w_{\mathbf{S}}(e_2)} \boldsymbol{\Delta}_{v_1 v_2}.$$

(3) **Approximation.** Return $\mathbf{K} = \sum_{i=1}^{d} \mathbf{X}_i$.

The key fact about this construction is that it produces an unbiased estimator $\mathbf{K}$ of the clique:

$$\mathbb{E}\, \mathbf{K} = \textsc{clique}(u, \mathbf{S}).$$

Furthermore, each summand $\mathbf{X}_i$ creates a multiedge with uniformly bounded effective resistance (7.5). This property persists as the SPARSECHOLESKY algorithm executes, and it ensures that the random matrix $\mathbf{K}$ has controlled variance. Note that the sampling procedure can result in several edges between the same pair of vertices, which is another reason we need the multigraph formalism.

Next, observe that the number d of multiedges in $\mathbf{K}$ is no greater than the number d of multiedges in the star that we are removing from $\mathbf{S}$. (For comparison, note that the full clique has d^2 multiedges.) As a consequence, the clique approximation is inexpensive to construct. Moreover, the total number of multiedges in the Laplacian can only decrease as the SPARSECHOLESKY algorithm proceeds.

18.3.3. *Analysis.* The analysis of SPARSECHOLESKY is well beyond the scope of this paper. The key technical tool is a concentration inequality for matrix-valued martingales that was derived in (Oliveira 2009*a*) and (Tropp 2011*a*). To activate this result, (Kyng and Sachdeva 2016) use the fact that the random clique approximation is unbiased and low-variance, conditional on previous choices made by the algorithm. The proof also relies heavily on the fact that we eliminate a random vertex at each step of the iteration. For the technical details, see (Kyng and Sachdeva 2016, Kyng 2017) and (Tropp 2019).

18.3.4. *Implementation.* The SPARSECHOLESKY algorithm is fairly simple to describe, but it demands some care to develop an implementation that achieves the runtime guarantees stated in Theorem 18.2.

The most important point is that we need to use data structures for weighted multigraphs. One method is to maintain the vertex–multiedge adjacency matrix, along with a list of weights. This approach requires sparse matrix libraries, including efficient iterators over the rows and columns.

A secondary point is that we need fast methods for constructing finite probability distributions and sampling from them repeatedly. See Bringmann and Panagiotou (2017).

It is unlikely that the SPARSECHOLESKY procedure will fail to produce a factor $\mathbf{C}$ that satisfies (18.1). Even so, it is reassuring to know that we can detect failures. Indeed, we can estimate the extreme singular values of the preconditioned linear system (18.2) using the methods from Section 6.

The failure probability can also be reduced by modifying the random vertex selection rule. Instead, we can draw a random vertex whose total weight is at most twice the average total weight of the remaining vertices (Kyng 2017). In practice, it may suffice to use the classical elimination rules based on minimum degree or minimum total weight.

The main shortcoming of the SPARSECHOLESKY procedure arises from the initialization step, where we split each multiedge into $O(\log n)$ pieces. This step increases the storage and computation costs of the algorithm enough to make it uncompetitive (e.g. with fast direct Poisson solvers) for some problem instances. At present, it is unclear whether the initialization step can be omitted or relaxed, while maintaining the reliability and correctness of the algorithm.

19. KERNEL MATRICES IN MACHINE LEARNING

Randomized NLA algorithms have played a major role in developing scalable kernel methods for large-scale machine learning. This section contains a brief introduction to kernels. Then it treats two probabilistic techniques that have had an impact on kernel matrix computations: Nyström approximation by random coordinate sampling (Williams and Seeger 2001) and empirical approximation by random features (Rahimi and Recht 2008).

The literature on kernel methods is truly vast, so we cannot hope to achieve comprehensive coverage within our survey. There are also many computational considerations and learning-theoretic aspects that fall outside the realm of NLA. Our goal is simply to give a taste of the ideas, along with a small selection of key references.

19.1. **Kernels in machine learning.** We commence with a crash course on kernels and their applications in machine learning. The reader may refer to Schölkopf and Smola (2001) for a more complete treatment.

19.1.1. *Kernel functions and kernel matrices.* Let $\mathcal{X}$ be a set, called the *input space* or *data space*. Suppose that we acquire a finite set of observations $\{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset \mathcal{X}$. We would like to use the observed data to perform learning tasks.

One approach is to introduce a *kernel function*:

$$k : \mathcal{X} \times \mathcal{X} \to \mathbb{F}.$$

The value $k(\mathbf{x}, \mathbf{y})$ of the kernel function is interpreted as a measure of similarity between two data points $\mathbf{x}$ and $\mathbf{y}$. We can tabulate the pairwise similarities of the observed data points in a

kernel matrix:

$$(\mathbf{K})_{ij} := k(\mathbf{x}_i, \mathbf{x}_j) \quad \text{for } i, j = 1, \ldots, n.$$

The kernel matrix is an analog of the Gram matrix of a set of vectors in a Euclidean space. In Sections 19.1.5 and 19.1.6, we will explain how to use the matrix $\mathbf{K}$ to solve some core problems in data analysis.

The kernel function is required to be *positive definite*. That is, for each natural number n and each set $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subset \mathcal{X}$ of observations, the associated kernel matrix

$$\mathbf{K} = \big[k(\mathbf{x}_i, \mathbf{x}_j)\big]_{i,j=1,\ldots,n} \in \mathbb{H}_n \quad \text{is psd.}$$

In particular, $k(\mathbf{x}, \mathbf{x}) \geq 0$ for all $\mathbf{x} \in \mathcal{X}$. The kernel must also be (conjugate) symmetric in its arguments: $k(\mathbf{x}, \mathbf{y}) = k(\mathbf{y}, \mathbf{x})^*$ for all $\mathbf{x}, \mathbf{y} \in \mathcal{X}$. These properties mirror the properties of a Gram matrix. In Section 19.1.3, we give some examples of positive definite kernel functions.

19.1.2. *The feature space.* It is common to present kernel functions using the theory of *reproducing kernel Hilbert spaces*. This approach gives an alternative interpretation of the kernel function as the inner product defined on a feature space. We give a very brief treatment, omitting all technical details.

Let $\mathcal{F}$ be a Hilbert space, called the *feature space*. We introduce a *feature map* $\Phi : \mathcal{X} \to \mathcal{F}$, which maps a point in the input space to a point in the feature space. Heuristically, the feature map extracts information from a data point that is relevant for learning applications.

Under mild conditions, we can construct a positive definite kernel function k from the feature map:

$$k(\mathbf{x}, \mathbf{y}) = \langle \Phi(\mathbf{x}),\, \Phi(\mathbf{y}) \rangle \quad \text{for all } \mathbf{x}, \mathbf{y} \in \mathcal{X}.$$

In other words, the kernel function reports the inner product between the features associated with the data points $\mathbf{x}$ and $\mathbf{y}$. Conversely, a positive definite kernel always induces a feature map into an appropriate feature space.

19.1.3. *Examples of kernels.* Kernel methods are powerful because we can select or design a kernel that automatically extracts relevant feature information from our data. This approach applies in all sorts of domains, including images and text and DNA sequences. Let us present a few kernels that commonly arise in applications. See Schölkopf and Smola (2001) for many additional examples and references.

Example 19.1 (Inner product kernel)**.** *The simplest example of a kernel is the ordinary inner product. Let $\mathcal{X} = \mathbb{F}^d$. Evidently,*

$$k(\mathbf{x}, \mathbf{y}) = \langle \mathbf{x},\, \mathbf{y} \rangle \quad \textit{for } \mathbf{x}, \mathbf{y} \in \mathbb{F}^d$$

is a positive definite kernel.

Example 19.2 (Angular similarity)**.** *Another simple example is the angular similarity map. Let $\mathcal{X} = \mathbb{S}^{d-1}(\mathbb{R}) \subset \mathbb{R}^d$. This kernel is given by the formula*

$$k(\mathbf{x}, \mathbf{y}) = \frac{2}{\pi} \arcsin \langle \mathbf{x},\, \mathbf{y} \rangle \quad \textit{for } \mathbf{x}, \mathbf{y} \in \mathcal{X}.$$

This kernel is positive definite because of Schoenberg's theorem (Schoenberg 1942). We will give a short direct proof in Example 19.8.

Example 19.3 (Polynomial kernels)**.** *Let $\mathcal{X}$ be a subset of $\mathbb{F}^d$. For a natural number p, the inhomogeneous polynomial kernel is*

$$k(\mathbf{x}, \mathbf{y}) = (1 + \langle \mathbf{x},\, \mathbf{y} \rangle)^p.$$

This kernel is also positive definite because of Schoenberg's theorem; see Kar and Karnick (2012). There is a short direct proof using the Schur product theorem.

Example 19.4 (Gaussian kernel)**.** *An important example is the Gaussian kernel. Let $\mathcal{X} = \mathbb{F}^d$. For a bandwidth parameter $\sigma > 0$, define*

$$k(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{y}\|^2}{2\sigma^2}\right) \quad \textit{for } \mathbf{x}, \mathbf{y} \in \mathbb{F}^d.$$

This kernel is positive definite because of Bochner's theorem (Bôchner 1933). We will give a short direct proof in Example 19.9.

19.1.4. *The kernel trick.* As we have mentioned, kernels can be used for a wide range of tasks in machine learning. Schölkopf and Smola (2001, Rem. 2.8) state the key idea succinctly:

> Given an algorithm which is formulated in terms of a positive definite kernel k, one can construct an alternative algorithm by replacing k with another positive definite kernel $\tilde{k}$.

In particular, any algorithm that can be formulated in terms of the inner product kernel applies to every other kernel. That is to say, an algorithm for Euclidean data that depends only on the Gram matrix can be implemented with a kernel matrix instead. The next two subsections give two specific examples of this methodology; there are many other applications.

19.1.5. *Kernel PCA.* Given a set of observations in a Euclidean space, principal component analysis (PCA) searches for orthogonal directions in which the data has the maximum variability. The nonlinear extension, kernel PCA (KPCA), was proposed in Schölkopf, Smola and Müller (1996); see also Schölkopf and Smola (2001, Chap. 14).

Let $\{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset \mathcal{X}$ be a set of observations. For a kernel k associated with a feature map Φ, construct the kernel matrix $\mathbf{K} \in \mathbb{H}_n$ associated with the observations. For a natural number ℓ, we compute a truncated eigenvalue decomposition of the kernel matrix:

$$\mathbf{K} = \sum_{i=1}^{\ell} \lambda_i \, \mathbf{u}_i \mathbf{u}_i^*.$$

Each unit-norm eigenvector $\mathbf{u}_i$ determines a direction $(n\lambda_i)^{-1/2} \sum_{j=1}^n \mathbf{u}_i(j)\, \Phi(\mathbf{x}_j)$ of high variability in the feature space, called the ith kernel principal component.

To find the projection of a new point $\mathbf{x} \in \mathcal{X}$ onto the ith kernel principal component, we embed it into the feature space via $\Phi(\mathbf{x})$ and compute the inner product with the ith kernel principal component. In terms of the kernel function,

$$\mathrm{PC}_i(\mathbf{x}) := \frac{1}{\sqrt{n\lambda_i}} \sum_{j=1}^{n} \mathbf{u}_i(j)\, k(\mathbf{x}, \mathbf{x}_j).$$

We can summarize the observation $\mathbf{x}$ with the vector

$$(\mathrm{PC}_1(\mathbf{x}), \dots, \mathrm{PC}_\ell(\mathbf{x})) \in \mathbb{F}^\ell.$$

This representation provides a data-driven feature that can be used for downstream learning tasks.

In practice, it is valuable to center the feature space representation of the data, which requires a simple modification of the kernel matrix. We also need to center each observation before computing its projection onto the kernel principal components. See Schölkopf et al. (1996, App. 1) for details.

19.1.6. *Kernel ridge regression.* Given a set of labeled observations in a Euclidean space, ridge regression uses regularized least-squares to model the labels as a linear functional of the observations. The nonlinear extension of this approach is called *kernel ridge regression* (KRR). We refer to Schölkopf and Smola (2001, Chap. 4) for a more detailed treatment, including an interpretation in terms of a nonlinear feature map.

Let $\{(\mathbf{x}_i, y_i) : i = 1, \dots, n\} \subset \mathcal{X} \times \mathbb{F}$ be a set of paired observations. For a kernel k, construct the kernel matrix $\mathbf{K} \in \mathbb{H}_n$ associated with the observations $\mathbf{x}_i$ (but not the numerical values y_i). For a regularization parameter $\tau > 0$, the kernel ridge regression problem takes the form

$$\operatorname*{minimize}_{\boldsymbol{\alpha} \in \mathbb{F}^n} \quad \frac{1}{n} \sum_{i=1}^{n} [y_i - (\mathbf{K}\boldsymbol{\alpha})_i]^2 + \frac{\tau}{2} \boldsymbol{\alpha}^* \mathbf{K} \boldsymbol{\alpha}.$$

The solution to this optimization problem is obtained by solving an ordinary linear system:

$$(\mathbf{K} + \tau n \, \mathbf{I})\boldsymbol{\alpha} = \mathbf{y} \quad \text{where} \quad \mathbf{y} = (y_1, \dots, y_n).$$

Let $\widehat{\boldsymbol{\alpha}}$ be the solution to this system.

Given a new observation $\mathbf{x} \in \mathcal{X}$, we can make a prediction $\widehat{y} \in \mathbb{F}$ for its label via the formula

$$\widehat{y}(\mathbf{x}) := \sum_{j=1}^{n} \widehat{\alpha}_j \, k(\mathbf{x}, \mathbf{x}_j).$$

In practice, the regularization parameter τ is chosen by cross-validation with a holdout set of the paired observations.

19.1.7. *The issue.* Kernel methods are powerful tools for data analysis. Nevertheless, in their native form, they suffer from two weaknesses.

First, it is very expensive to compute the kernel matrix explicitly. For example, if points in the data space $\mathcal{X}$ have a d-dimensional parameterization, we may expect that it will cost $O(d)$ arithmetic operations to evaluate the kernel a single time. Therefore, the cost of forming the kernel matrix $\mathbf{K}$ for n observations is $O(n^2 d)$.

Second, after computing the kernel matrix $\mathbf{K}$, it remains expensive to perform the linear algebra required by kernel methods. Both KPCA and KRR require $O(n^3)$ operations if we use direct methods.

The poor computational profile of kernel methods limits our ability to use them directly for large-scale data applications.

19.1.8. *The solution.* Fortunately, there is a path forward. To implement kernel methods, we simply need to *approximate* the kernel matrix (Schölkopf and Smola 2001, Sec. 10.2). Surprisingly, using the approximation often results in *better* learning outcomes than using the exact kernel matrix. Even a poor approximation of the kernel can suffice to achieve near-optimal performance, both in theory and in practice (Bach 2013, Rudi, Camoriano and Rosasco 2015, Rudi et al. 2017). Last, working with a structured approximation of the kernel can accelerate the linear algebra computations dramatically.

Randomized algorithms provide several effective tools for approximating kernel matrices. Since we pay a steep price for each kernel evaluation, we need to develop algorithms that explicitly control this cost. The rest of this section describes two independent approaches. In Section 19.2, we present coordinate sampling algorithms for Nyström approximations, while Section 19.3 develops the method of random features.

Remark 19.5 (Function approximation)**.** *Let us remark that approximation of the kernel matrix is usually incidental to the goals of learning theory. In many applications, such as KRR, we actually need to approximate a function on the input space. The sampling complexity of the latter task may be strictly lower than the complexity of approximating the full kernel matrix. We cannot discuss this issue in detail because it falls outside the scope of NLA.*

19.2. **Coordinate Nyström approximation of kernel matrices.** One way to approximate a kernel matrix is to form a Nyström decomposition with respect to a judiciously chosen coordinate subspace. A natural idea is to draw these coordinates at random. This basic technique was proposed by Williams and Seeger (2001).

Coordinates play a key role here because we only have access to individual entries of the kernel matrix. There is no direct way to compute a matrix–vector product with the kernel matrix, so we cannot easily apply the more effective constructions of random embeddings (e.g., Gaussians or sparse maps or SRTTs). Indeed, kernel computation is the primary setting where coordinate sampling is a practical idea.

19.2.1. *Coordinate Nyström approximation.* Suppose that $\{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset \mathcal{X}$ is a collection of observations. Let $\mathbf{K}$ be the psd kernel matrix associated with some kernel function k.

Given a set $I \subseteq \{1, \dots, n\}$ consisting of r indices, we can form a psd Nyström approximation of the kernel matrix:

$$\mathbf{K}\langle I \rangle := \mathbf{K}(:, I) \, \mathbf{K}(I, I)^{\dagger} \, \mathbf{K}(I, :).$$

This matrix is equivalent to the Nyström decomposition (14.1) with respect to a test matrix $\mathbf{X}$ whose range is $\operatorname{span}\{\boldsymbol{\delta}_i : i \in I\}$.

To obtain $\mathsf{K}\langle I\rangle$, the basic cost is nr kernel evaluations, which typically require $\mathcal{O}(nrd)$ operations for a d-dimensional input space $\mathcal{X}$. We typically do not form the pseudoinverse directly, but rather use the factored form of the Nyström approximation for downstream calculations.

For kernel problems, it is common to regularize the coordinate Nyström approximation. One approach replaces the core matrix $\mathsf{K}(I, I)$ with its truncated eigenvalue decomposition before computing the pseudoinverse; for example, see Drineas and Mahoney (2005). The RSVD algorithm (Section 11.2) has been proposed for this purpose (Li et al. 2015*a*). When it is computationally feasible, we recommend taking a truncated eigenvalue decomposition of the full Nyström approximation $\mathsf{K}\langle I\rangle$, rather than just the core; see Tropp et al. (2017*a*).

19.2.2. *Greedy selection of coordinates.* Recall from Section 14 that the error $\mathsf{K}/I := \mathsf{K} - \mathsf{K}\langle I\rangle$ in the Nyström decomposition is simply the Schur complement of K with respect to the coordinates in I.

This connection suggests that we should use a pivoted Cholesky method or a pivoted QR algorithm to select the coordinates I. These techniques lead to Nyström approximations with superior learning performance; for example, see (Fine and Scheinberg 2001, Bach and Jordan 2005) and (Bach 2013). Unfortunately the $\mathcal{O}(n^2 r)$ cost is prohibitive in applications. See Schölkopf and Smola (2001, Sec. 10.2) for some randomized strategies that can reduce the expense.

19.2.3. *Ridge leverage scores.* A natural approach to selecting the coordinate set I is to perform randomized sampling. To describe these approaches, we need to take a short detour.

Fix a regularization parameter $\tau > 0$. Consider the smoothed projector:

$$h_\tau(\mathsf{K}) := \mathsf{K}(\mathsf{K} + \tau n \mathsf{I})^{-1}.$$

The number of effective degrees of freedom at regularization level τ is

$$\nu_{\text{eff}} := \operatorname{trace} h_\tau(\mathsf{K}).$$

The maximum marginal number of degrees of freedom at regularization level τ is

$$\nu_{\text{mof}} := n \cdot \max_{i=1,\dots,n} (h_\tau(\mathsf{K}))_{ii}.$$

Observe that $\nu_{\text{eff}} \le \nu_{\text{mof}}$. The statistic ν_{mof} is analogous with the coherence that appears in our initial discussion of coordinate sampling (Section 9.6).

The *ridge leverage scores* at regularization level τ are the (normalized) diagonal entries of the smoothed projector:

$$p_i = \frac{(h_\tau(\mathsf{K}))_{ii}}{\nu_{\text{eff}}} \quad \text{for } i = 1, \dots, n.$$

Evidently, $(p_1, \dots, p_n)$ is a probability distribution. The ridge leverage scores and related quantities are expensive to compute directly, but there are efficient algorithms for approximating them well. These approximations suffice for applications. See Section 19.2.5 for more discussion.

Remark 19.6 (History)**.** *Bach (2013) identified the core role of the smoothed projector for KRR. Alaoui and Mahoney (2015) proposed the definition of the ridge leverage scores and described a simple method for approximating them. At present, the most practical algorithm for approximating ridge leverage scores appears in Rudi et al. (2018). Musco and Musco (2017) recognize that ridge leverage scores also have relevance for KPCA.*

19.2.4. *Uniform sampling.* The simplest way to select a set I of r coordinates for the Nyström approximation $\mathsf{K}\langle I\rangle$ is to draw the set uniformly at random. Although this approach seems naïve, it can be surprisingly effective in practice. The main failure mode occurs when there are a few significant observations that make outsize contributions to the kernel matrix; uniform sampling is likely to miss these influential data points.

Bach (2013) proves that we can achieve optimal learning guarantees for KRR with a uniformly sampled Nyström approximation. It suffices that the number r of coordinates is proportional to $\nu_{\text{mof}} \log n$. A similar result holds for KPCA.

Nyström approximation with uniform coordinate sampling was proposed by Williams and Seeger (2001). The theoretical and numerical performance of this approach has been studied in many subsequent works, including (Kumar et al. 2012, Gittens 2013, Bach 2013) and (Rudi et al. 2017).

19.2.5. *Sampling with ridge leverage scores.* Suppose that we have computed an approximation of the ridge leverage score distribution. We can construct a coordinate set I for the Nyström approximation $\mathbf{K}\langle I\rangle$ by sampling r coordinates independently at random from the ridge leverage score distribution. Properly implemented, this method is unlikely to miss influential observations.

Alaoui and Mahoney (2015) prove that we can achieve optimal learning guarantees for KRR by ridge leverage score sampling. It suffices that the number r of sampled coordinates is proportional to $\nu_{\mathrm{eff}} \log n$. This bound improves over the uniform sampling bound. Musco and Musco (2017) give related theoretical results for KPCA.

Effective algorithms for estimating ridge leverage scores are based on multilevel procedures that sequentially improve the ridge leverage score estimates. The basic idea is to start with a small uniform sample of coordinates, which we use to approximate the smoothed projector for a very large regularization parameter τ_0. From this smoothed projector, we estimate the ridge leverage scores at level τ_0. We then sample a larger set of coordinates non-uniformly using the approximate ridge leverage score distribution at level τ_0. These samples allow us to approximate the smoothed projector at level $\tau_1 = \mathrm{const.}\tau_0$ for a constant smaller than one. We obtain an estimate for the ridge leverage score distribution at level τ_1. This process is repeated. In this way, the sampling and the matrix approximation are intertwined. See (Musco and Musco 2017) and (Rudi et al. 2018).

Musco and Musco (2017) provide empirical evidence that ridge leverage score sampling is more efficient than uniform sampling for KPCA, including the cost of the ridge leverage score approximations. Likewise, Rudi et al. (2018) report empirical evidence that ridge leverage score sampling is more efficient than uniform sampling for KRR.

19.3. **Random features approximation of kernels.** A second approach to kernel approximation is based on the method of empirical approximation (Section 7). This technique constructs a random rank-one matrix that serves as an unbiased estimator for the kernel matrix. By averaging many copies of the estimator, we can obtain superior approximations of the kernel matrix. The individual rank-one components are called *random features*.

Neal (1996) proposed the idea of using empirical approximation for kernels arising in Gaussian process regression. Later, Rahimi and Recht (2008) and Rahimi and Recht (2009) developed empirical approximations for translation-invariant kernels and Mercer kernels, and they coined the term "random features." Our presentation is based on an abstract formulation of the random feature method from Tropp (2015, Sec. 6.5); see also Bach (2017).

In this subsection, we introduce the idea of a random feature map, along with some basic examples. We explain how to use random feature maps to construct empirical approximations of a kernel matrix, and we give a short analysis. Afterwards, we summarize two randomized NLA methods for improving the computational profile of random features.

19.3.1. *Random feature maps.* In many cases, a kernel function k on a domain $\mathcal{X}$ can be written as an expectation, and we can exploit this representation to obtain empirical approximations of the kernel matrix.

Let $\mathcal{W}$ be a probability space equipped with a probability measure ρ. Assume that there is a bounded function

$$\psi : \mathcal{X} \times \mathcal{W} \to \{z \in \mathbb{C} : |z| \leq b\}$$

with the reproducing property

$$k(\mathbf{x}, \mathbf{y}) = \int \psi(\mathbf{x}; \mathbf{w})\, \psi(\mathbf{y}; \mathbf{w})^*\, \rho(\mathrm{d}\mathbf{w}) \quad \text{for all } \mathbf{x}, \mathbf{y} \in \mathcal{X}. \tag{19.1}$$

The star * denotes the conjugate of a complex number. We call (ψ, ρ) a *random feature map* for the kernel function k. As we will see in Section 19.3.2, a kernel that admits a random feature map must be positive definite.

It is not obvious that we can equip kernels of practical interest with random feature maps, so let us offer a few concrete examples.

Example 19.7 (The inner product kernel)**.** *There are many ways to construct a random feature map for the inner product kernel on $\mathbb{F}^d$. One simple example is*

$$\psi(\mathbf{x}; \mathbf{w}) = \langle \mathbf{x}, \mathbf{w} \rangle \quad \textit{with } \mathbf{w} \sim \text{NORMAL}(\mathbf{0}, \mathbf{I}_d).$$

To check that this map satisfies the reproducing property (19.1)*, just note that $\mathbf{w}$ is isotropic: $\mathbb{E}[\mathbf{w}\mathbf{w}^*] = \mathbf{I}$. This formulation is closely related to the theory of random embeddings (Sections 8 and 9) and to approximate matrix multiplication (Section 7.3).*

Example 19.8 (Angular similarity kernel)**.** *For the angular similarity map defined in Example 19.2, we can construct a random feature map using an elegant fact from geometry. Indeed, the function*

$$\psi(\mathbf{x}; \mathbf{w}) = \operatorname{sgn} \langle \mathbf{x}, \mathbf{w} \rangle \quad \textit{with } \mathbf{w} \sim \text{UNIFORM}(\mathbb{S}^{d-1}(\mathbb{R}))$$

gives a random feature map for the angular similarity kernel. As a consequence, the angular similarity kernel is positive definite.

Example 19.9 (Translation-invariant kernels)**.** *A kernel function k on $\mathbb{F}^d$ is called* translation-invariant *if it has the form*

$$k(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x} - \mathbf{y}) \quad \textit{for all } \mathbf{x}, \mathbf{y} \in \mathbb{F}^d.$$

A classic result from analysis, Bochner's theorem (Bôchner 1933), gives a characterization of these kernels. A kernel is continuous, positive definite and translation-invariant if and only if it is the Fourier transform of a positive probability measure ρ on $\mathbb{F}^d$:

$$\phi(\mathbf{x} - \mathbf{y}) = c_\phi \int \mathrm{e}^{\mathrm{i}\langle \mathbf{x}, \mathbf{w} \rangle} \mathrm{e}^{-\mathrm{i}\langle \mathbf{y}, \mathbf{w} \rangle} \, \rho(\mathrm{d}\mathbf{w}).$$

The constant c_ϕ is a normalizing factor that depends only on ϕ, and i is the imaginary unit.

Bochner's theorem immediately delivers a random feature map for the translation-invariant kernel k:

$$\psi(\mathbf{x}; \mathbf{w}) = \sqrt{c_\phi}\, \mathrm{e}^{\mathrm{i}\langle \mathbf{x}, \mathbf{w} \rangle} \quad \textit{where } \mathbf{w} \sim \rho.$$

This was one of the original examples of a random feature map (Rahimi and Recht 2008). When working with data in $\mathbb{R}^d$, the construction can also be modified to avoid complex values.

The key example of a positive definite, translation-invariant kernel is the Gaussian kernel on $\mathbb{F}^d$, defined in Example 19.4. The Gaussian kernel is derived from the function

$$\phi(\mathbf{x}) = \mathrm{e}^{-\|\mathbf{x}\|^2/(2\sigma^2)} \quad \textit{where the bandwidth } \sigma > 0.$$

The associated random feature map is

$$\psi(\mathbf{x}; \mathbf{w}) = \mathrm{e}^{\mathrm{i}\langle \mathbf{x}, \mathbf{w} \rangle} \quad \textit{where} \quad \mathbf{w} \sim \text{NORMAL}(\mathbf{0}, \sigma^{-2}\mathbf{I}) \in \mathbb{F}^d.$$

This fact is both beautiful and useful because of the ubiquity of the Gaussian kernel in data analysis.

There are many other kinds of kernels that admit random feature maps. Random feature maps for dot product kernels were obtained in (Kar and Karnick 2012, Pham and Pagh 2013) and (Hamid, Xiao, Gittens and Decoste 2014). For nonstationary kernels, see (Samo and Roberts 2015) and (Ton, Flaxman, Sejdinovic and Bhatt 2018). Catalogs of examples appear in Rudi and Rosasco (2017, App. E) and Bach (2017).

19.3.2. *Random features and kernel matrix approximation.* We can use the random feature map to construct an empirical approximation of the kernel matrix $\mathbf{K} \in \mathbb{H}_n$ induced by the dataset $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$. To do so, we draw a random variable $\mathbf{w} \in \mathcal{W}$ with the distribution ρ. Then we form a random vector

$$\mathbf{z} = \begin{bmatrix} z_1 \\ \vdots \\ z_n \end{bmatrix} = \begin{bmatrix} \psi(\mathbf{x}_1; \mathbf{w}) \\ \vdots \\ \psi(\mathbf{x}_n; \mathbf{w}) \end{bmatrix} \in \mathbb{F}^n.$$

Note that we are using the *same* random variable $\mathbf{w}$ for each data point. A realization of the random vector $\mathbf{z}$ is called a *random feature*. The reproducing property (19.1) ensures that the random feature verifies the identity

$$(\mathbf{K})_{ij} = k(\mathbf{x}_i, \mathbf{x}_j) = \int \psi(\mathbf{x}_i; \mathbf{w})\, \psi(\mathbf{x}_j; \mathbf{w})^*\, \rho(\mathrm{d}\mathbf{w}) = \mathbb{E}[z_i \cdot z_j^*].$$

In matrix form,

$$\mathbf{K} = \mathbb{E}[\mathbf{z}\mathbf{z}^*].$$

Therefore, the random rank-one psd matrix $\mathbf{Z} = \mathbf{z}\mathbf{z}^*$ is an unbiased estimator for the kernel matrix. The latter display proves that a kernel k must be positive definite if it admits a random feature map.

To approximate the kernel matrix, we can average r copies of the rank-one estimator:

$$\bar{\mathbf{Z}}_r := \frac{1}{r} \sum\nolimits_{i=1}^{r} \mathbf{Z}_i \quad \text{where } \mathbf{Z}_i \sim \mathbf{Z} \text{ are iid.}$$

If the points in the input space $\mathcal{X}$ are parameterized by d numbers, each random feature typically requires $O(nd)$ arithmetic. The total cost of forming the approximation $\bar{\mathbf{Z}}_r$ is thus $O(rnd)$. When $r \ll n$, we can obtain substantial improvements over the direct approach of computing the kernel matrix $\mathbf{K}$ explicitly at a cost of $O(n^2 d)$.

19.3.3. *Analysis of the random feature approximation.* How many random features are enough to approximate the kernel matrix in spectral norm? Theorem 7.2 delivers bounds.

For simplicity, assume that the kernel satisfies $k(\mathbf{x}, \mathbf{x}) = 1$ for all $\mathbf{x} \in \mathcal{X}$; the angular similarity kernel and the Gaussian kernel both enjoy this property. For an accuracy parameter $\varepsilon > 0$, suppose that we select

$$r \geq 2b\varepsilon^{-2} \operatorname{intdim}(\mathbf{K}) \log(2n).$$

The number b is the uniform bound on the feature map ψ defined in (19.1), and the intrinsic dimension is defined in (2.1). Theorem 7.2 implies that the empirical approximation $\bar{\mathbf{Z}}_r$ satisfies

$$\frac{\mathbb{E}\, \|\bar{\mathbf{Z}}_r - \mathbf{K}\|}{\|\mathbf{K}\|} \leq \varepsilon + \varepsilon^2.$$

In other words, we achieve a relative error approximation of the kernel matrix in spectral norm when the number r of random features is proportional to the number of energetic dimensions in the range of the matrix $\mathbf{K}$.

This analysis is due to Lopez-Paz, Sra, Smola, Ghahramani and Schölkopf (2014); see also (Tropp 2015, Sec. 6.5). For learning applications, such as KPCA or KRR, this result suggests that we need about $O(n \log n)$ random features to obtain optimal generalization guarantees (where $\varepsilon = n^{-1/2}$). In fact, roughly $O(\sqrt{n}\, \log n)$ random features are sufficient to achieve optimal learning rates. This claim depends on involved arguments from learning theory that are outside the realm of linear algebra. For example, see (Sriperumbudur and Szabo 2015, Rudi and Rosasco 2017, Ullah, Mianjy, Marinov and Arora 2018, Szabo and Sriperumbudur 2019) and (Wang 2019).

19.3.4. *Randomized embeddings and random features.* Random feature approximations are faster than explicit computation of a kernel matrix. Even so, it takes a significant amount of effort to extract r random features and form an empirical approximation $\bar{\mathbf{Z}}_r$ of the kernel matrix. Several groups have proposed using structured random embeddings (Section 9) to accelerate this process; see (Pham and Pagh 2013, Le, Sarlós and Smola 2013) and (Hamid et al. 2014).

As an example, let us summarize a heuristic method, called *FFT Fastfood* (Le et al. 2013), for speeding up the computation of random features for the complex Gaussian kernel with bandwidth σ^2. Consider the matrix $\mathbf{X}$ formed from n observations in $\mathbb{C}^d$:

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^* \\ \vdots \\ \mathbf{x}_n^* \end{bmatrix} \in \mathbb{C}^{n \times d}.$$

Let $\mathbf{\Gamma} \in \mathbb{C}^{d \times d}$ be a matrix with iid complex $\textsc{normal}(0, \sigma^{-2})$ entries. Then we can simultaneously compute d random features $\mathbf{z}_1, \dots, \mathbf{z}_d$ for the Gaussian kernel by forming a matrix product and applying the exponential map:

$$\exp \cdot (\mathrm{i}\mathbf{X}\mathbf{\Gamma}) = \begin{bmatrix} \mathbf{z}_1 & \dots & \mathbf{z}_d \end{bmatrix} \in \mathbb{C}^{n \times d}.$$

We have written $\exp \cdot$ for the *entrywise* exponential. This procedure typically involves $O(nd^2)$ arithmetic.

The idea behind FFT Fastfood is to accelerate this computation by replacing the Gaussian matrix with a structured random matrix. This exchange is motivated by the observed universality properties of random embeddings. Consider a random matrix of the form

$$\mathbf{S} = \frac{1}{\sigma} \mathbf{E}\mathbf{\Pi}\mathbf{F} \in \mathbb{C}^{d \times d},$$

where $\mathbf{E}$ is a random sign flip, $\mathbf{\Pi}$ is a random permutation, and $\mathbf{F}$ is the discrete DFT. The FFT algorithm supports efficient matrix products with $\mathbf{S}$. Therefore, we can simultaneously extract d random features by computing $\exp \cdot (\mathrm{i}\mathbf{X}\mathbf{S}) \in \mathbb{C}^{n \times d}$. This procedure uses only $O(nd \log d)$ operations. To form r random features where $r > d$, we simply repeat the same process $\lceil r/d \rceil$ times.

Compared with using a Gaussian matrix product, FFT Fastfood gives a substantial reduction in arithmetic. Even so, the performance for learning is almost identical to a direct application of the random feature approximation.

19.3.5. *Random features and streaming matrix approximation.* Suppose that we wish to perform KPCA. The direct random features approach requires us to form the empirical approximation $\bar{\mathbf{Z}}_r$ of the kernel matrix $\mathbf{K}$ and to compute its rank-ℓ truncated eigenvalue decomposition. It is often the case that the desired number ℓ of principal components is far smaller than the number r of random features we need to obtain a suitable approximation of the kernel matrix. In this case, we can combine random features with streaming matrix approximation to make economies in storage and computation.

Let $\{\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3, \dots\} \subset \mathbb{F}^n$ be an iid sequence of random features for the kernel matrix $\mathbf{K} \in \mathbb{H}_n$. The empirical approximation $\bar{\mathbf{Z}}_r$ of the kernel matrix, obtained from the first r random features, follows the recursion

$$\bar{\mathbf{Z}}_0 = \mathbf{0} \quad \text{and} \quad \bar{\mathbf{Z}}_t = (1 - t^{-1})\bar{\mathbf{Z}}_{t-1} + t^{-1}\, \mathbf{z}_t \mathbf{z}_t^* \quad \text{for } t = 1, 2, 3, \dots.$$

This is a psd matrix, generated by a stream of linear updates. Therefore, we can track the evolution using a streaming Nyström approximation (Section 14).

Let $\mathbf{\Omega} \in \mathbb{F}^{n \times s}$ be a random test matrix, with $s > \ell$. By performing rank-one updates, we can efficiently maintain the sample matrices

$$\mathbf{Y}_t = \bar{\mathbf{Z}}_t \mathbf{\Omega} \in \mathbb{F}^{n \times s} \quad \text{for } t = 1, 2, 3, \dots.$$

After collecting a sufficient number r of samples, we can apply Algorithm 16 to $\mathbf{Y}_t$ to obtain a near-optimal rank-ℓ eigenvalue decomposition of the empirical approximation $\bar{\mathbf{Z}}_r$.

It usually suffices to take the sketch size s to be proportional to the rank ℓ of the truncated eigenvalue decomposition. In this case, the overall approach uses $O(\ell n)$ storage. We can generate and process r random features using $O((d+\ell)rn)$ arithmetic, where d is the dimension of $\mathcal{X}$. The subsequent cost of the Nyström approximation is $O(\ell^2 n)$ operations. The streaming random features approach has storage and arithmetic costs roughly ℓ/r times those of the direct random features approach. The streaming method can be combined with dimension reduction techniques (Section 19.3.4) for further acceleration.

Remark 19.10 (History)**.** *Ghashami, Perry and Phillips (2016*b*) proposed using a stream of random features to perform KPCA; their algorithm tracks the stream with the (deterministic) frequent directions sketch (Ghashami et al. 2016*a*). We have presented a new variant, based on randomized Nyström approximation, that is motivated by the work in Tropp et al. (2017*a*). Ullah et al. (2018) have develop a somewhat different streaming KPCA algorithm based on Oja's method (Oja 1982). At present, we lack a full empirical comparison of these alternatives.*

20. High-accuracy approximation of kernel matrices

In this section, we continue the discussion of kernel matrices that we started in Section 19, but we now consider the high-accuracy regime. In particular, given a kernel matrix $\mathbf{K}$, we seek an approximation $\mathbf{K}_{\text{approx}}$ for which $\|\mathbf{K} - \mathbf{K}_{\text{approx}}\|$ is small, say of relative accuracy 10^{-3} or 10^{-6}. This objective was not realistic for the applications discussed in Section 19, but it can be achieved in situations where we have access to fast techniques for evaluating the matrix-vector product $\mathbf{x} \mapsto \mathbf{K}\mathbf{x}$ (and also $\mathbf{x} \mapsto \mathbf{K}^*\mathbf{x}$ when $\mathbf{K}$ is not self-adjoint). The algorithms that we describe will build a data sparse approximation to $\mathbf{K}$ by using information in samples such as $\mathbf{K}\mathbf{x}$ and $\mathbf{K}^*\mathbf{x}$ for random vectors $\mathbf{x}$. These techniques are particularly well suited to problems that arise in modeling physical phenomena such as electromagnetic scattering, or the deformation of solid bodies; we describe how the fast matrix-vector application we need can be realized in Section 20.3.

As in Section 19, we say that a matrix $\mathbf{K} \in \mathbb{C}^{n\times n}$ is a *kernel matrix* if its entries are given by a formula such as

$$\mathbf{K}(i,j) = k(\mathbf{x}_i, \mathbf{x}_j), \tag{20.1}$$

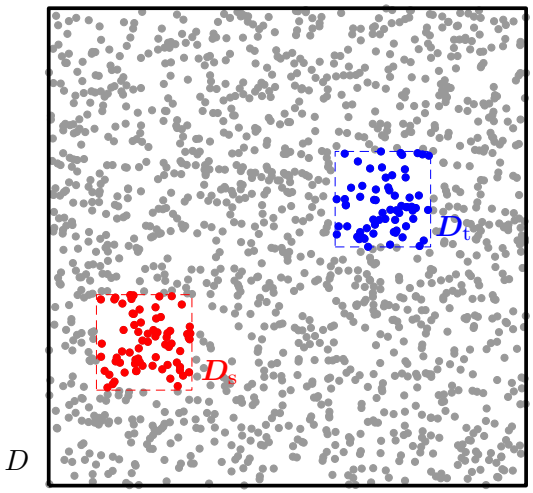
where $\{\mathbf{x}_i\}_{i=1}^n$ is a set of points in $\mathbb{R}^d$, and where $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{C}$ is a kernel function. The kernel matrices that we consider tend to have singular values that decay slowly or not at all, which rules out the possibility that $\mathbf{K}_{\text{approx}}$ could have low rank. Instead, we build an approximation $\mathbf{K}_{\text{approx}}$ that is tessellated into $O(n)$ blocks in such as way that each off-diagonal block has low rank. Figure 11(b) shows a representative tessellation pattern. We say that a matrix of this type is a *rank-structured hierarchical matrix.*

The purpose of determining a rank-structured approximation to an operator for which we already have fast matrix-vector multiplication techniques available is that the new representation can be used to rapidly execute a whole range of linear algebraic operations: matrix inversion, LU factorization, and even full spectral decompositions in certain cases.

This section is structured to provide a high-level description of the core ideas in Sections 20.1 – 20.3. Additional details follow in Sections 20.4 – 20.9.

20.1. **Separation of variables and low-rank approximation.** The reason that many kernel matrices can be tessellated into blocks that have low numerical rank is that the function $(\mathbf{x}, \mathbf{y}) \mapsto k(\mathbf{x}, \mathbf{y})$ is typically smooth as long as $\mathbf{x}$ and $\mathbf{y}$ are not close. To illustrate the connection, let us consider a computational domain D that holds a set of points $\{\mathbf{x}_i\}_{i=1}^n \subset \mathbb{R}^2$, as shown in Figure 10. Suppose further that D_{s} and D_{t} are two subdomains of D that are located a bit apart from each other, as shown in the figure. When the kernel function k is smooth, we can typically approximate it to high accuracy through an approximate separation of variables of the form

$$k(\mathbf{x}, \mathbf{y}) \approx \sum_{p=1}^{P} b_p(\mathbf{x})\, c_p(\mathbf{y}), \qquad \mathbf{x} \in D_{\text{t}},\ \mathbf{y} \in D_{\text{s}}. \tag{20.2}$$



A box D holding points $\{\mathbf{x}_i\}_{i=1}^n$ (the blue, red, and gray dots) that define a kernel matrix $\mathbf{K}(i,j) = k(\mathbf{x}_i, \mathbf{x}_j)$. The regions D_{s} (the red box) and D_{t} (the blue box) are separated enough that $k(\mathbf{x}, \mathbf{y})$ is smooth when $\mathbf{x} \in D_{\mathrm{t}}$ and $\mathbf{y} \in D_{\mathrm{s}}$. In consequence, $\mathbf{K}(I_{\mathrm{t}}, I_{\mathrm{s}})$ has low numerical rank, where I_{s} identifies the red points and I_{t} the blue.

FIGURE 10. The geometry discussed in Section 20.2.

Let I_{s} and I_{t} denote two index vectors that identify the points located in D_{s} and D_{t}, respectively (so that, for example, $i \in I_{\mathrm{s}}$ if and only if $\mathbf{x}_i \in D_{\mathrm{s}}$). Then combining the formula (20.1) with the separation of variables (20.2), we get

$$\mathbf{K}(i,j) \approx \sum_{p=1}^{P} b_p(\mathbf{x}_i)\, c_p(\mathbf{x}_j), \qquad i \in I_{\mathrm{t}},\ j \in I_{\mathrm{s}}. \tag{20.3}$$

Equation (20.3) is exactly the low-rank approximation to the block $\mathbf{K}(I_{\mathrm{t}}, I_{\mathrm{s}})$ that we seek. To be precise, (20.3) can be written as

$$\mathbf{K}(I_{\mathrm{t}}, I_{\mathrm{s}}) \approx \mathbf{B}\mathbf{C},$$

where $\mathbf{B}$ and $\mathbf{C}$ are defined via $\mathbf{B}(i,p) = b_p(\mathbf{x}_i)$ and $\mathbf{C}(p,j) = c_p(\mathbf{x}_j)$.

A separation of variables such as (20.2) is sometimes provided through analytic knowledge of the kernel function, as illustrated in Example 20.1. Perhaps more typically, all we know is that such a formula *should* in principle exist, for instance because we know that the matrix approximates a singular integral operator for which a Calderón-Zygmund decomposition must exist. It is then the task of the randomized algorithm to explicitly build the factors $\mathbf{B}$ and $\mathbf{C}$, given a computational tolerance.

Example 20.1 (Laplace kernel in two dimensions)**.** *A standard example of a kernel matrix in mathematical physics is the matrix* $\mathbf{K}$ *that maps a vector of electric source strengths* $\mathbf{q} = (q_j)_{j=1}^n$ *to a vector of potentials* $\mathbf{u} = (u_i)_{i=1}^n$*. When the set* $\{\mathbf{x}_i\}_{i=1}^n \subset \mathbb{R}^2$ *identifies the locations of both the source and the target points,* $\mathbf{K}$ *takes the form (20.1), for*

$$k(\mathbf{x}, \mathbf{y}) = \begin{cases} \log|\mathbf{x} - \mathbf{y}|, & \text{when } \mathbf{x} \neq \mathbf{y}, \\ 0, & \text{when } \mathbf{x} = \mathbf{y}. \end{cases}$$

There is a well-known result from potential theory that provides the required separation of variables. Expressing $\mathbf{x}$ *and* $\mathbf{y}$ *in polar coordinates with respect to some expansion center* $\mathbf{c}$*, so that* $\mathbf{x} - \mathbf{c} = r(\cos\theta, \sin\theta)$ *and* $\mathbf{y} - \mathbf{c} = r'(\cos\theta', \sin\theta')$*, the separation of variables (known as a multipole expansion)*

$$k(\mathbf{x}, \mathbf{y}) = \log(r) - \sum_{p=1}^{\infty} \frac{1}{p} \left(\frac{r'}{r}\right)^p \big(\cos(p\theta)\cos(p\theta') + \sin(p\theta)\sin(p\theta')\big) \tag{20.4}$$

is valid whenever $r' < r$.

20.2. **Rank-structured matrices and randomized compression.** The observation that the off-diagonal blocks of a kernel matrix often have low numerical rank underpins many "fast" algorithms in computational physics. In particular, it is the foundation of the Barnes-Hut (Barnes and Hut 1986) and fast multipole methods (Greengard and Rokhlin 1987) for evaluating all $O(n^2)$ pairwise interactions between n electrically charged particles in linear or close-to-linear complexity. These methods were generalized by Hackbusch and co-workers, who developed the $\mathcal{H}$- and $\mathcal{H}^2$-matrix frameworks (Hackbusch 1999, Grasedyck and Hackbusch 2003, Hackbusch, Khoromskij and Sauter 2002). These explicitly linear algebraic formulations enable fast algorithms not only for matrix-vector multiplication but also for matrix inversion, LU factorization, matrix-matrix multiplication and many more.

A fundamental challenge that arises when rank-structured matrix formulations are used is how to find the data sparse representation of the operator in the first place. The straightforward approach would be to form the full matrix, and then loop over all the compressible off diagonal blocks and compress them using, for example, a singular value decomposition. The cost of such a process is necessarily at least $O(n^2)$, which is rarely affordable. Using randomized compression techniques, it turns out to be possible to compress all the off-diagonal blocks *jointly*, without any need for sampling each block individually. In this section, we describe two such methods. Both require the user to supply a fast algorithm for applying the full matrix (and its adjoint) to vectors; see Section 20.3. The two methods have different computational profiles.

(a) The technique described in Sections 20.4 and 20.5 is a true "black-box" technique that interacts with $\mathbf{K}$ only through the matrix-vector multiplication. It has storage complexity $O(n \log n)$ and it requires $O(\log n)$ applications of $\mathbf{K}$ to a random matrix of size $n \times (r + p)$ where r is an upper bound on the ranks of the off-diagonal blocks, and p is a small over-sampling parameter.

(b) The technique described in Sections 20.6 and 20.7 attains true linear $O(n)$ complexity, and requires only a single application of $\mathbf{K}$ and $\mathbf{K}^*$ to a random matrix of size $n \times (r + p)$, where r and p are as in (a). Its drawbacks are that it requires evaluation of $O(rn)$ individual matrix entries, and that it works only for a smaller class of matrices.

Remark 20.2 (Scope of Section 20)**.** *To keep the presentation as uncluttered by burdensome notation as possible, in this survey we restrict attention to two basic "formats" for representing a rank-structured hierarchical matrix. In Sections 20.4 and 20.5, we use the hierarchically off-diagonal low rank (HODLR) format and in Sections 20.6 and 20.7 we use the hierarchically block separable (HBS) format (sometimes referred to as hierarchically semi separable (HSS) matrices). The main limitation of these formats is that they require all off-diagonal blocks of the matrix to have low numerical rank. This is realistic only when the points* $\{\mathbf{x}_i\}_{i=1}^n$ *are restricted to a low-dimensional manifold. In practical applications, one sometimes has to leave a larger part of the matrix uncompressed, to avoid attempting to impose a separation of variables such as (20.2) on a kernel* $k = k(\mathbf{x}, \mathbf{y})$ *when* $\mathbf{x}$ *and* $\mathbf{y}$ *are too close. This is done through enforcing what is called a "strong admissibility condition" (in contrast to the "weak admissibility condition" of the HODLR and HBS formats), as was done in the original Barnes-Hut and fast multipole methods.*

Remark 20.3 (Alternative compression strategies)**.** *Let us briefly describe what alternatives to randomized compression exist. The original papers on* $\mathcal{H}$*-matrices used Taylor approximations to derive a separation of variables, but this works only when the kernel is given analytically. It also tends to be quite expensive. The adaptive cross-approximation (ACA) technique of (Kurz, Rain and Rjasanow 2002) and (Bebendorf and Grzhibovskis 2006) relies on using "natural basis" vectors (see Section 13.1) that are found using semi-heuristic techniques. The method can work very well in practice, but is not guaranteed to provide an accurate factorization. When the kernel matrix comes from mathematical physics, specialized techniques that exploit mathematical properties of the kernel function often perform well*

(Martinsson and Rokhlin 2005). For a detailed discussion of compression of rank-structured matrices, we refer to (Martinsson 2019, Ch. 17).

20.3. **Computational environments.** The compression techniques that we describe rely on the user providing a fast algorithm for applying the operator to be compressed to vectors. Representative environments where such fast algorithms are available include the following.

Matrix-matrix multiplication: Suppose that $\mathbf{K} = \mathbf{BC}$, where $\mathbf{B}$ and $\mathbf{C}$ are matrices that can rapidly be applied to vectors. Then the randomized compression techniques allow us to compute their product.

Compression of boundary integral operators: It is often possible to reformulate a boundary value problem involving an elliptic partial different operator, such as the Laplace or the Helmholtz operators, as an equivalent boundary integral equation (BIE), see (Martinsson 2019, Part III). When such a BIE is discretized, the result is a kernel matrix which can be applied rapidly to vectors using fast summation techniques such as the fast multipole method (Greengard and Rokhlin 1987). Randomized compression techniques allow us to build an approximation to the matrix that can be factorized or inverted, thus enabling direct (as opposed to iterative) solvers.

Dirichlet-to-Neumann (DtN) operators: A singular integral operator of central importance in engineering, physics, and scientific computing is the DtN operator which maps given Dirichlet data for an elliptic boundary value problem to the boundary fluxes of the corresponding solution. The kernel of the DtN operator is rarely known analytically, but the operator can be applied through a fast solver for the PDE, such as, e.g., a finite-element discretization combined with a multigrid solver. The randomized techniques described allow for the DtN operator to be built and stored explicitly.

Frontal matrices in sparse direct solvers: Suppose that $\mathbf{S}$ is a large sparse matrix arising from the discretization of an elliptic PDEs. A common technique for solving $\mathbf{Sx} = \mathbf{b}$ is to compute an LU factorization of the matrix $\mathbf{S}$. There are ways to do this that preserve sparsity as far as possible, but in the course of the factorization procedure, certain dense matrices of increasing size will need to be factorized. These matrices turn out to be kernel matrices with kernels that are not known explicitly, but that can be built using the techniques described here (Xia 2013, Ghysels et al. 2017).

20.4. **Hierarchically off-diagonal low-rank matrices.** In order to illustrate how randomized methods can be used to construct data sparse representations of rank-structured matrices, we will describe a particularly simple 'format' in this section that is often referred to as the hierarchically off-diagonal low-rank (HODLR) format. This is a basic format that works well when the points are organized on a one- or two-dimensional manifold.

The first step towards defining the HODLR format is to build a binary tree on the index vector $I = [1, 2, 3, \ldots, n]$ through a process that is illustrated in Figure 11(a). With any node τ in the tree, we associate an index vector $I_\tau \subseteq I$. The root of the tree is given the index $\tau = 1$, and we associate it with the full index vector, so that $I_1 = I$. At the next finer level of the tree, we split I_1 into two parts I_2 and I_3 so that $I_1 = I_2 \cup I_3$ forms a disjoint partition. Then continue splitting the index vectors until each remaining index vector is "short". (Exactly what "short" means is application-dependent, but one may think of a short vector as holding a few hundred indices or so.) We let ℓ denote a *level* of the tree, with $\ell = 0$ denoting the root, so that level ℓ holds 2^ℓ nodes. We use the terms *parent* and *child* in the obvious way, and say that a pair of nodes $\{\alpha, \beta\}$ forms a *sibling pair* if they have the same parent. A *leaf* node is of course a node that has no children.

The binary tree that we defined induces a natural tessellation of the kernel matrix $\mathbf{K}$ into $O(n)$ blocks. Figure 11(b) shows the tessellation that follows from the tree in Figure 11(a). Each parent node τ in the tree gives rise to two off-diagonal blocks that both have low numerical rank. Letting $\{\alpha, \beta\}$ denote the children of τ, these two blocks are

$$\mathbf{K}_{\alpha,\beta} = \mathbf{K}(I_\alpha, I_\beta) \qquad \text{and} \qquad \mathbf{K}_{\beta,\alpha} = \mathbf{K}(I_\beta, I_\alpha). \tag{20.5}$$

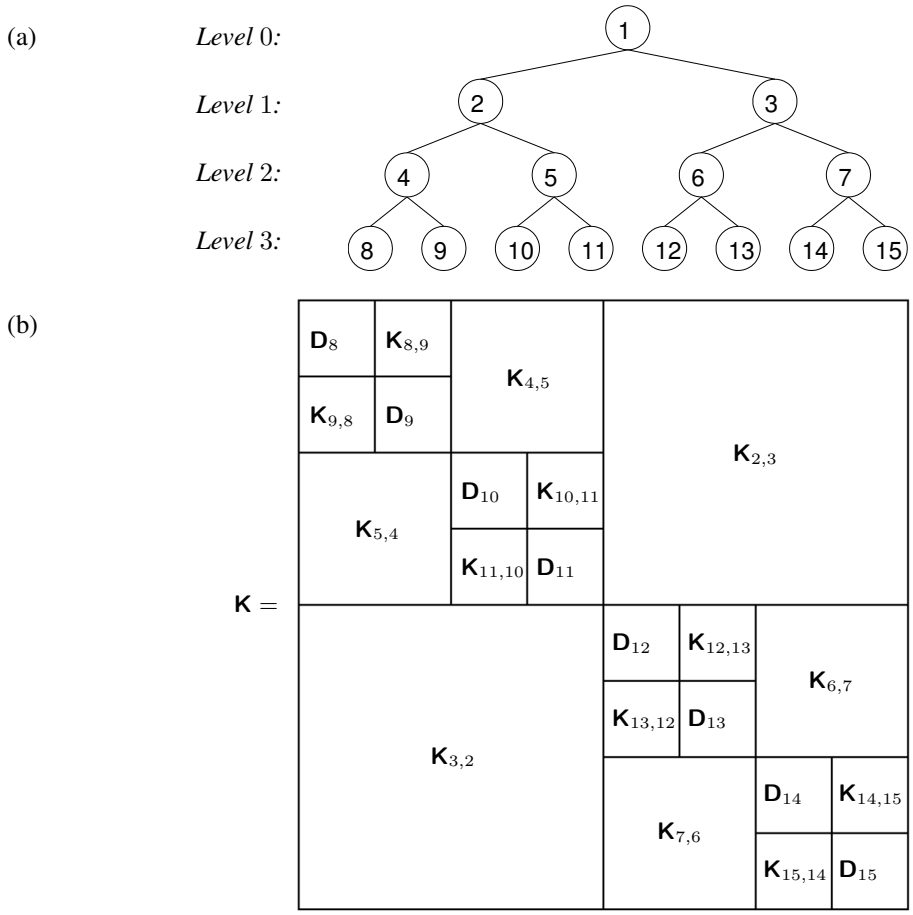


FIGURE 11. (a) A binary tree with three levels; see Section 20.4. Each node τ in the tree owns an index vector I_τ that is a subset of the full index vector $I = 1 : n$. For the root node, we set $I_1 = I$. Each split in the tree represents a disjoint partition of the corresponding index vectors, so that, for example, $I_1 = I_2 \cup I_3$ and $I_7 = I_{14} \cup I_{15}$. (b) A matrix $\mathbf{K}$ tessellated according to the tree shown in (a).

For each leaf node τ, we define a corresponding diagonal block as

$$\mathbf{D}_\tau = \mathbf{K}(I_\tau, I_\tau). \tag{20.6}$$

The disjoint partition of $\mathbf{K}$ into blocks is now formed by all sibling pairs, as defined in (20.5), together with all diagonal blocks, as defined by (20.6). When each off-diagonal block in this tessellation has low rank, we say that $\mathbf{K}$ is a *hierarchically off-diagonal low-rank (HODLR)* matrix.

20.5. **Compressing a rank-structured hierarchical matrix through the matrix-vector multiplication only.** In this section, we describe a randomized technique for computing a data sparse representation of a matrix that is compressible in the "HODLR" format that we introduced in Section 20.4. This technique interacts with $\mathbf{K}$ only through the application of $\mathbf{K}$ and $\mathbf{K}^*$ to vectors, and is in this sense a "black-box" technique. In particular, we do not need the ability to evaluate individual entries of $\mathbf{K}$. The following theorem summarizes the main result:

Theorem 20.4. *Let $\mathbf{K}$ be a HODLR matrix associated with a fully populated binary tree on the index vector, as described in Section 20.4. Suppose that the tree has L levels, that each off-diagonal block has rank at most k, and that each leaf node in the tree holds at most ck indices for some fixed number c. Then the diagonal blocks $\mathbf{D}_\tau$, as well as rank-k factorizations of all*

sibling interaction matrices $\mathbf{K}_{\alpha,\beta}$ can be computed by a randomized algorithm with cost at most

$$T_{\text{total}} = T_{\text{matvec}} \times (4L + c)k + T_{\text{flop}} \times O(L^2 k^2 n),$$

where T_{matvec} is the cost of applying either $\mathbf{K}$ or $\mathbf{K}^*$ to a vector, and T_{flop} is the cost of a floating point operation.

The proof of the theorem consists of an explicit algorithm for building all matrices that is often referred to as the "peeling algorithm". It was originally published in (Lin, Lu and Ying 2011), with later modifications proposed in (Martinsson 2016). The proof below is based on (Martinsson 2019, Sec. 17.4). We give the proof for the case described in the theorem involving a matrix whose off-diagonal blocks have exact rank k. In practical applications, it is of course more typical to have the off-diagonal blocks be only of approximate low rank. In this case, the exact same algorithm can be used, but the number of samples drawn should be increased from k to $k + p$ for some modest oversampling parameter p.

Proof. The algorithm is a "top-down" technique that compresses the largest blocks first, and then moves on to process one level at a time of successively smaller blocks. In describing the technique, we assume that the blocks are numbered as shown in Figure 11(a).

In the first step of the algorithm, we build approximations to the two largest blocks $\mathbf{K}_{2,3}$ and $\mathbf{K}_{3,2}$, shown in red in Figure 12(a). To do this, we form a random matrix $\mathbf{\Omega}$ of size $n \times 2k$ and then use the matrix-vector multiplication to form a sample matrix

$$\underset{n\times 2k}{\mathbf{Y}} = \underset{n\times n}{\mathbf{K}}\ \underset{n\times 2k}{\mathbf{\Omega}}.$$

The key idea of the peeling algorithm is to insert zero blocks in the random matrix $\mathbf{\Omega}$ in the pattern shown in Figure 12(a). The zero blocks permit us to extract "pure" samples from the column spaces of the blocks $\mathbf{K}_{2,3}$ and $\mathbf{K}_{3,2}$. For instance, the blocks labeled $\mathbf{Y}_2$ and $\mathbf{Y}_3$ in the figure are given by the formulas

$$\underset{\frac{n}{2}\times k}{\mathbf{Y}_2} = \underset{\frac{n}{2}\times\frac{n}{2}}{\mathbf{K}_{2,3}}\ \underset{\frac{n}{2}\times k}{\mathbf{\Omega}_3} \qquad \text{and} \qquad \underset{\frac{n}{2}\times k}{\mathbf{Y}_3} = \underset{\frac{n}{2}\times\frac{n}{2}}{\mathbf{K}_{3,2}}\ \underset{\frac{n}{2}\times k}{\mathbf{\Omega}_2}$$

By orthonormalizing the matrices $\mathbf{Y}_2$ and $\mathbf{Y}_3$, we obtain ON bases $\mathbf{U}_2$ and $\mathbf{U}_3$ for the off-diagonal blocks $\mathbf{K}_{2,3}$ and $\mathbf{K}_{3,2}$. In order to complete the factorization of $\mathbf{K}_{2,3}$ and $\mathbf{K}_{3,2}$, we will perform an operation that is the equivalent of "Stage B" in Section 11.2. To this end, we form a test matrix by interlacing the matrices $\mathbf{U}_2$ and $\mathbf{U}_3$ with zero blocks, to form the $n \times 2k$ matrix shown in Figure 13. Applying $\mathbf{K}^*$ to this test matrix, we get the sample matrices

$$\mathbf{Z}_2 = \mathbf{K}_{3,2}^* \mathbf{U}_3, \qquad \text{and} \qquad \mathbf{Z}_3 = \mathbf{K}_{2,3}^* \mathbf{U}_2.$$

Since $\mathbf{U}_2$ holds an ON-basis for the column space of $\mathbf{K}_{2,3}$, it follows that

$$\mathbf{K}_{2,3} = \mathbf{U}_2 \mathbf{U}_2^* \mathbf{K}_{2,3} = \mathbf{U}_2 \mathbf{Z}_3^*,$$

which establishes the low rank factorization of $\mathbf{K}_{2,3}$.[2] The block $\mathbf{K}_{3,2}$ is of course factorized analogously.

At the second step of the algorithm, the objective is to build approximations to the sibling matrices at the next finer level, shown in red in Figure 12(b). We use a random matrix $\mathbf{\Omega}$ of the same size, $n \times 2k$, as in the previous step, but now with four zero blocks, as shown in the figure. Looking at the sample matrix $\mathbf{Y}_4$, we find that it takes the form

$$\underset{\frac{n}{4}\times k}{\mathbf{Y}_4} = \underset{\frac{n}{4}\times\frac{n}{4}}{\mathbf{K}_{4,5}}\ \underset{\frac{n}{4}\times k}{\mathbf{\Omega}_5} + \underset{\frac{n}{4}\times k}{\mathbf{B}_4},$$

[2] One may if desired continue the factorization process and form an "economy size" SVD of $\mathbf{Z}_3^*$ so that $\mathbf{Z}_3^* = \hat{\mathbf{U}}_2 \mathbf{\Sigma}_{2,3} \mathbf{V}_3^*$. This results in the factorization $\mathbf{K}_{2,3} = \big(\mathbf{U}_2 \hat{\mathbf{U}}_2\big) \mathbf{\Sigma}_{2,3} \mathbf{V}_3^*$, which is an SVD of $\mathbf{K}_{2,3}$. However, for purposes of establishing the theorem, we may leave $\mathbf{Z}_3$ alone, and let $\{\mathbf{U}_2, \mathbf{Z}_3\}$ be our "compressed" representation of $\mathbf{K}_{2,3}$.

(a) In the first step of the algorithm, the sibling interaction matrices $\mathbf{K}_{2,3}$ and $\mathbf{K}_{3,2}$ on level 1, shown in red, are compressed.

(b) In the second step, the four sibling interaction matrices on level 2, shown in red, are compressed. In this step, we exploit that we now possess factorizations of the gray blocks.

(c) In the third step, the eight sibling interaction matrices on level 8, shown in red, are compressed. We again exploit that we possess factorizations of the gray blocks.

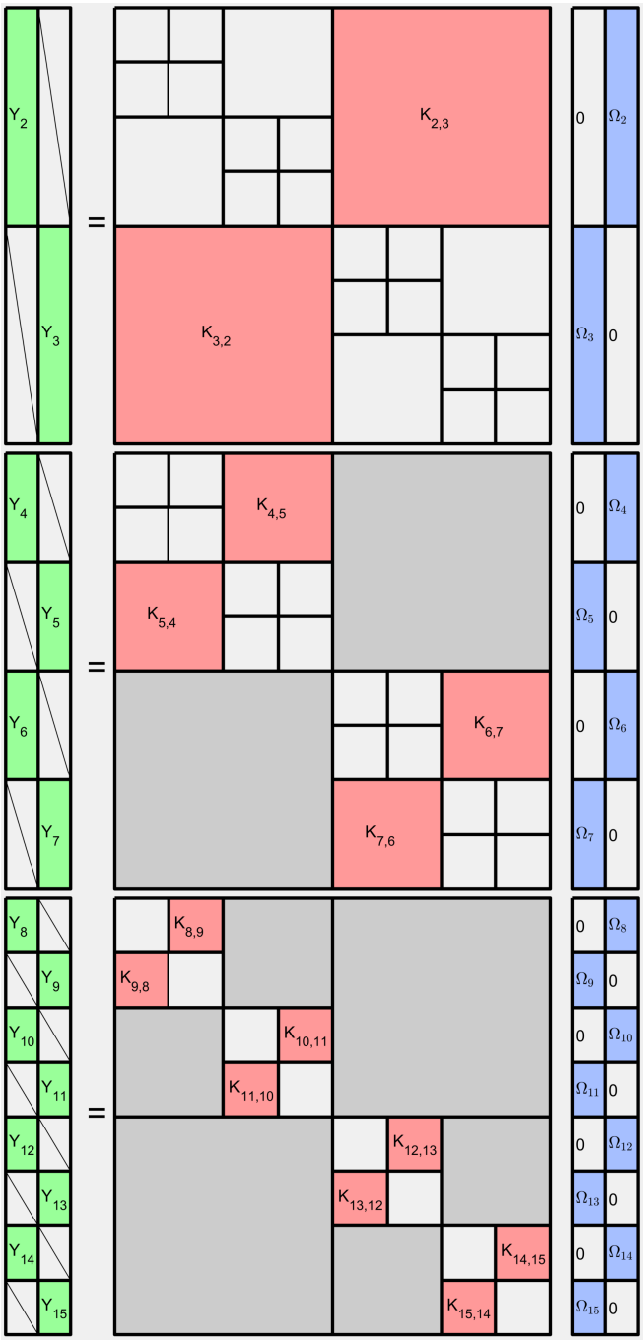


FIGURE 12. The "peeling algorithm" for computing a HODLR representation of a matrix described in Section 20.5.

where $\mathbf{B}_4$ represents contributions from interactions between the block $\mathbf{K}_{2,3}$ and the random matrix and $\mathbf{\Omega}_7$. But now observe that we at this point are in possession of a low rank approximation to $\mathbf{K}_{2,3}$. This allows us to compute $\mathbf{B}_4$ explicitly, and then obtain a "pure" sample from the column space of $\mathbf{K}_{4,5}$ by subtracting this contribution out:

$$\mathbf{Y}_4 - \mathbf{B}_4 = \mathbf{K}_{4,5}\mathbf{\Omega}_5.$$

By orthonormalizing the matrix $\mathbf{Y}_4 - \mathbf{B}_4$, we obtain the ON matrix $\mathbf{U}_4$ whose columns span the column space of $\mathbf{K}_{4,5}$. The same procedure of course also allows us to build bases for the

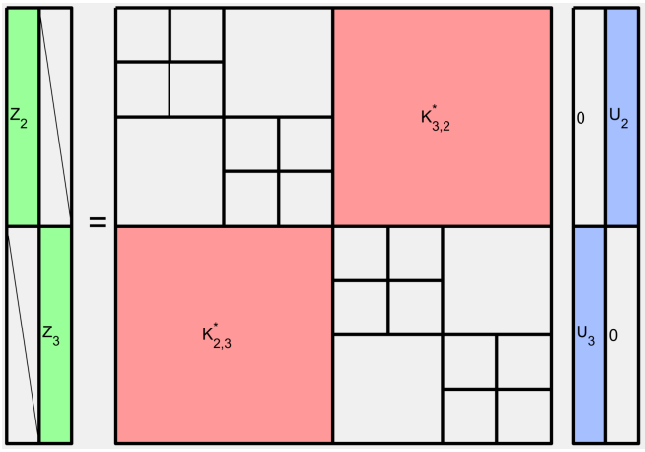


FIGURE 13. The process used to complete the factorization of the blocks $\mathbf{K}_{2,3}$ and $\mathbf{K}_{3,2}$ once the basis matrices $\mathbf{U}_2$ and $\mathbf{U}_3$ have been constructed. We put $\mathbf{U}_2$ and $\mathbf{U}_3$ into the $n \times 2k$ test matrix shown, and form the sample matrix that holds $\mathbf{Z}_2$ and $\mathbf{Z}_3$ by applying $\mathbf{K}^*$.

column spaces of all matrices at this level, and we can then extract bases for the row spaces and the cores of the matrices by sampling $\mathbf{K}^*$.

At the third step of the algorithm, the object is to compress the blocks marked in red in Figure 12(c). The random matrix used remains of size $n \times 2k$ but now contains 8 zero blocks, as shown in the figure. The sample matrix $\mathbf{Y}_8$ takes the form

$$\underset{\frac{n}{8}\times k}{\mathbf{Y}_8} = \underset{\frac{n}{8}\times\frac{n}{8}}{\mathbf{K}_{8,9}} \underset{\frac{n}{8}\times k}{\boldsymbol{\Omega}_9} + \underset{\frac{n}{8}\times k}{\mathbf{B}_8}$$

where $\mathbf{B}_8$ represents contributions from off-diagonal blocks at the coarser levels, shown in gray in the figure. Since we already possess data sparse representations of these blocks, we can subtract their contributions out, just as at the previous level.

Once all levels have been processed, we hold low-rank factorizations of all off-diagonal blocks, and all that remains is to extract the diagonal matrices $\mathbf{D}_\tau$ for all leaf nodes τ. Let us for simplicity assume that all such block are of the same size $m \times m$. We then form a test matrix by stacking n/m copies of an $m \times m$ identity matrix atop of each other and applying $\mathbf{K}$ to this test matrix. We subtract off the contributions from all the off-diagonal blocks using the representation we have on hand, and are then left with a sample matrix consisting of all the blocks $\mathbf{D}_\tau$ stacked on top of each other. □

20.6. **A linear complexity data sparse format.** The "HODLR" data sparse matrix format discussed in Sections 20.4 and 20.5 is simple to use, and is in many applications efficient enough. However, its storage complexity is at least $O(n \log n)$ as the matrix size n grows. We will next describe how the logarithmic factor can be eliminated. To keep the presentation concise, we focus on the asymptotic storage requirements, although similar estimates hold for the asymptotic flop count.

To start, let us first investigate why there are logarithmic factors in the storage requirement in the HODLR format. Suppose that for a sibling pair $\{\alpha, \beta\}$ the corresponding off-diagonal block $\mathbf{K}_{\alpha,\beta}$ is of size $m \times m$, has rank k, and is stored in the form of a factorization

$$\underset{m\times m}{\mathbf{K}_{\alpha,\beta}} = \underset{m\times k}{\mathbf{U}_\alpha^{\text{long}}} \underset{k\times k}{\tilde{\mathbf{K}}_{\alpha,\beta}} \underset{k\times m}{\left(\mathbf{V}_\beta^{\text{long}}\right)^*}$$

where $\mathbf{U}_\alpha^{\text{long}}$ and $\mathbf{V}_\beta^{\text{long}}$ are two matrices whose columns form bases for the column and row space of $\mathbf{K}_{\alpha,\beta}$, respectively.[3] We see that $\mathbf{K}_{\alpha,\beta}$ can be stored using about $2mk$ floating point

[3]The matrices $\mathbf{U}_\alpha^{\text{long}}$ and $\mathbf{V}_\beta^{\text{long}}$ were denoted $\mathbf{U}_\alpha$ and $\mathbf{V}_\beta$ in Sections 20.4 and 20.5.

numbers (where we ignore $O(k^2)$ terms, since typically $k \ll m$). Now let us consider how many floats are required to store all sibling interaction matrices on a given level ℓ in the tree. There are 2^ℓ such matrices, and they each have size $m \approx 2^{-\ell}n$, resulting in a total of $2^\ell \times 2 \times 2^{-\ell}nk = 2nk$ floats. Since there are $O(\log(n))$ levels in the tree, it follows that just the task of storing all the basis matrices requires $O(kn\log(n))$ storage.

A standard technique for overcoming the problem of storing all the "long" basis matrices is to assume that the basis matrices on one level can be formed through slight modifications to the basis matrices on the next finer level. To be precise, we assume that if τ is a node in the tree with children α and β, then there exists a "short" basis matrix $\mathbf{U}_\tau$ of size $2k \times k$ such that

$$\underset{m \times m}{\mathbf{U}_\tau^{\text{long}}} = \underset{m \times 2k}{\begin{bmatrix} \mathbf{U}_\alpha^{\text{long}} & \mathbf{0} \\ \mathbf{0} & \mathbf{U}_\beta^{\text{long}} \end{bmatrix}} \underset{2k \times k}{\mathbf{U}_\tau} \tag{20.7}$$

(with of course an analogous statement holding for the "V" matrices holding bases for the row spaces). The point is that if we already have the "long" basis matrices for the children available, then the long basis matrices for the parent can be formed using the information in the small matrix $\mathbf{U}_\tau$. By applying this argument recursively, we see that for any parent node τ, all that needs to be stored explicitly are small matrices of size $2k \times k$.

To illustrate the idea, suppose that we use this technique for storing basis matrices to the tree with three levels described in Section 20.4. Then for each of the 8 leaf boxes, we compute basis matrices of size $n/8 \times k$ and store these. At the next coarser level, the long basis matrix for a box such as $\tau = 4$ can be expressed through the formula

$$\underset{(n/4) \times m}{\mathbf{U}_4^{\text{long}}} = \underset{(n/4) \times 2k}{\begin{bmatrix} \mathbf{U}_8^{\text{long}} & \mathbf{0} \\ \mathbf{0} & \mathbf{U}_9^{\text{long}} \end{bmatrix}} \underset{2k \times k}{\mathbf{U}_4},$$

so that only the $2k \times k$ matrix $\mathbf{U}_4$ needs to be stored. At the next coarser level still, the long basis matrix $\mathbf{U}_2^{\text{long}}$ is expressed as

$$\underset{(n/2) \times m}{\mathbf{U}_2^{\text{long}}} = \underset{(n/2) \times 4k}{\begin{bmatrix} \mathbf{U}_8^{\text{long}} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{U}_9^{\text{long}} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{U}_{10}^{\text{long}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{U}_{11}^{\text{long}} \end{bmatrix}} \underset{4k \times 2k}{\begin{bmatrix} \mathbf{U}_4 & \mathbf{0} \\ \mathbf{0} & \mathbf{U}_5 \end{bmatrix}} \underset{2k \times k}{\mathbf{U}_2}.$$

We say that a matrix that can be represented using nested basis matrices in the manner described is a *hierarchically block separable (HBS)* matrix. (This format is very closely related to the *hierarchically semi separable (HSS)* matrix format described in (Xia, Chandrasekaran, Gu and Li 2010, Chandrasekaran, Gu and Lyons 2005).) When the HBS format is used, it is natural to assume that each leaf index vector holds $O(k)$ indices, which means that there are overall $O(n/k)$ nodes in the tree. Since we only need $O(k^2)$ storage per node, we see that the overall storage requirements improve from $O(kn\log(n))$ to $O(kn)$.

There is a price to be paid for using "nested" basis matrices, and it is that the rank k required to reach a requested precision ε typically is higher when nested basis matrices are used, in comparison to the HODLR format. To be precise, one can show that for the relation (20.7) to hold, it is necessary and sufficient that for a node α, the columns of $\mathbf{U}_\alpha^{\text{long}}$ must span the column space of the matrix $\mathbf{K}(I_\alpha, I_\alpha^{\text{c}})$ (where $I_\alpha^{\text{c}} = I \backslash I_\alpha$). In contrast, in the HODLR format, they only need to span the columns of $\mathbf{K}(I_\alpha, I_\beta)$, where β is the sibling of α. This is easier to do since I_β is a subset of the index vector I_α^{c}.

20.7. **A linear complexity randomized compression technique.** We saw in Section 20.6 that the so called "HBS" data sparse format allows us to store a matrix using $O(kn)$ floating point numbers, without any factor of $\log(n)$. But is it also possible to compute such a representation in optimal linear complexity? The answer is yes, and a number of deterministic techniques that work in specific environments have been proposed (Martinsson 2019, Ch. 17).

A linear complexity randomized technique was proposed in (Martinsson 2011). This method is based on analyzing samples of the matrix obtained through the application of $\mathbf{K}$ and $\mathbf{K}^*$ to Gaussian random vectors, but is not quite a black-box method, as it also requires the ability to evaluate $O(kn)$ individual elements of $\mathbf{K}$ itself. This is not always possible, of course, but when it is, the technique is lightning fast in practice. To be precise, (Martinsson 2011) establishes the following:

Theorem 20.5. *Let $\mathbf{K}$ be an $n \times n$ matrix that is compressible in the HBS format described in Section 20.6, for some rank k. Let T_{apply} denote the time it takes to evaluate the two products*

$$\mathbf{Y} = \mathbf{K}\mathbf{G}, \tag{20.8}$$

$$\mathbf{Z} = \mathbf{K}^*\mathbf{G}, \tag{20.9}$$

where $\mathbf{G}$ is an $n \times k$ matrix drawn from a Gaussian distribution. Then a full HBS representation of $\mathbf{K}$ can be computed at cost bounded by

$$T_{\text{total}} = T_{\text{apply}} + T_{\text{entry}} \times O(nk) + T_{\text{flop}} \times O(nk^2),$$

where T_{flop} is the cost of a floating point operation, and where T_{entry} is the time it takes to evaluate an individual entry of $\mathbf{K}$

When the off-diagonal blocks are only approximately of rank k, we do some over sampling as usual, and replace the number k in the theorem by $k + p$ for some small number p. This of course results in an approximate HBS representation of $\mathbf{K}$.

The proof of Theorem 20.5 is an algorithm that explicitly builds the data sparse representation of the matrix within the specified time budget. The algorithm consists of a pass through all nodes in the hierarchical tree, going from smaller boxes to larger; "bottom-up", as opposed to the "top-down" method for HODLR matrices in Section 20.5. We provide an outline of the proof below. This outline is written to convey the main ideas without introducing cumbersome notation; for full details, see the original article (Martinsson 2011) or (Martinsson 2019, Sec. 17.4).

Proof. To describe the algorithm that establishes Theorem 20.5, we will walk through how it applies to a matrix tessellated in accordance with the hierarchical tree in Figure 11(a). We start by showing how we can use the information in the matrix $\mathbf{Y}$ defined by (20.8) to build the basis matrix $\mathbf{U}_8$. As we saw in Section 20.6, we need the columns of $\mathbf{U}_8$ to span the columns in the submatrix $\mathbf{K}(I_8, I_8^{\text{c}})$. We achieve this by constructing the sample matrix

$$\mathbf{Y}_8 := \mathbf{K}(I_8, I_8^{\text{c}})\mathbf{G}(I_8^{\text{c}}, :).$$

Since $\mathbf{G}(I_8^{\text{c}}, :)$ is a Gaussian random matrix of the appropriate size, the columns of $\mathbf{Y}_8$ will form an approximate basis for the column space of $\mathbf{K}(I_8, I_8^{\text{c}})$. But how do we form $\mathbf{Y}_8$ from the sample matrix $\mathbf{Y}$ defined by (20.8)? The method we use is illustrated in Figure 14(a), where the block $\mathbf{K}(I_8, I_8^{\text{c}})$ consists of the gray block inside the red rectangle. We see that in order to isolate the product between $\mathbf{K}(I_8, I_8^{\text{c}})$ and $\mathbf{G}(I_8^{\text{c}}, :)$, all we need to do is to subtract the contribution from the diagonal block $\mathbf{K}(I_8, I_8)$ (marked in white in the figure). In other words,

$$\mathbf{Y}_8 = \mathbf{K}(:, I_8)\mathbf{G} - \mathbf{K}(I_8, I_8)\mathbf{G}(I_8, :) = \mathbf{Y}(I_8, :) - \mathbf{K}(I_8, I_8)\mathbf{G}(I_8, :). \tag{20.10}$$

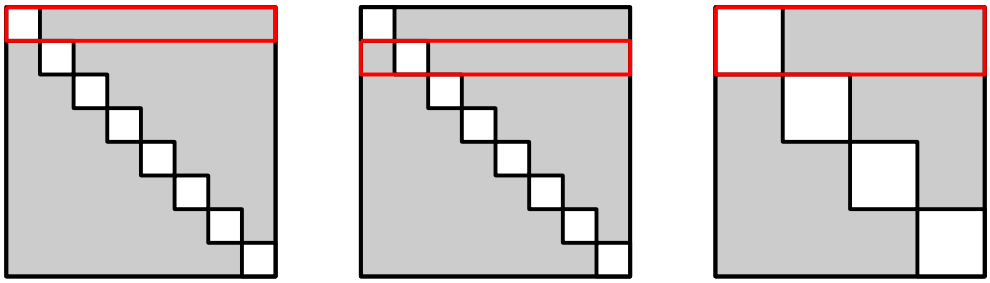
The formula (20.10) can be evaluated inexpensively since the block $\mathbf{K}(I_8, I_8)$ is small, and can be explicitly formed since we assume that we have access to individual entries of the matrix $\mathbf{K}$. Once $\mathbf{Y}_8$ has been formed, we compute its row interpolatory decomposition to form a basis matrix $\mathbf{U}_8$ and an index vector $\tilde{I}_8^{\text{row}}$ such that

$$\mathbf{K}(I_8, I_8^{\text{c}}) = \mathbf{U}_8\, \mathbf{K}(\tilde{I}_8^{\text{row}}, I_8^{\text{c}}).$$

In an entirely analogous way, we form a sample matrix $\mathbf{Y}_9$ whose columns span the block $\mathbf{K}(I_9, I_9^{\text{c}})$ through the formula

$$\mathbf{Y}_9 = \mathbf{Y}(I_9, :) - \mathbf{K}(I_9, I_9)\mathbf{G}(I_9, :),$$

cf. Figure 14(b), and then compute the row ID $\{\mathbf{U}_9, \tilde{I}_9^{\text{row}}\}$ of $\mathbf{Y}_9$.



(a) (b) (c)

FIGURE 14. The algorithm that establishes Theorem 20.5 is illustrated using a matrix $\mathbf{K}$ tessellated in accordance with the tree shown in Figure 11(a). (a) The block $\mathbf{K}(I_8, :)$ is marked with a red rectangle. Within the red rectangle, the shaded area represents the block $\mathbf{K}(I_8, I_8^{\mathrm{c}})$. (b) The block $\mathbf{K}(I_9, :)$ is marked with a red rectangle, with $\mathbf{K}(I_9, I_9^{\mathrm{c}})$ shaded. (c) The block $\mathbf{K}(I_4, :)$ is marked with a red rectangle, with $\mathbf{K}(I_4, I_4^{\mathrm{c}})$ shaded.

The basis matrices $\mathbf{V}_\tau$ that span the row spaces of the offdiagonal blocks for any leaf τ are built through the same procedure, but starting with the sample matrix $\mathbf{Z}$ defined by (20.9). For instance, we form the sample matrix

$$\mathbf{Z}_8 = \mathbf{K}(I_8, :)^*\mathbf{G} - \mathbf{K}(I_8, I_8)^*\mathbf{G}(I_8, :) = \mathbf{Z}(I_8, :) - \mathbf{K}(I_8, I_8)^*\mathbf{G}(I_8, :), \tag{20.11}$$

and then compute the ID of $\mathbf{Z}_8$ to build a basis matrix $\mathbf{V}_8$ and an index vector $\tilde{I}_8^{\mathrm{col}}$ such that

$$\mathbf{K}(I_8^{\mathrm{c}}, I_8) = \mathbf{K}(I_8^{\mathrm{c}}, \tilde{I}_8^{\mathrm{col}})\mathbf{V}_8^*.$$

The choice to use the interpolatory decomposition in factorizing the off-diagonal blocks is essential to making the linear complexity compression scheme work. In particular, observe that once we have formed the row and column IDs of all the leaf boxes, we automatically obtain rank-k factorizations of all the corresponding sibling interaction matrices. For instance, if $\{\alpha, \beta\}$ is a sibling pair consisting of two leaf nodes, then the $m \times m$ sibling interaction matrix $\mathbf{K}(I_\alpha, I_\beta)$ admits the factorization

$$\underset{m \times m}{\mathbf{K}(I_\alpha, I_\beta)} = \underset{m \times k}{\mathbf{U}_\alpha} \quad \underset{k \times k}{\mathbf{K}(\tilde{I}_\alpha^{\mathrm{row}}, \tilde{I}_\beta^{\mathrm{col}})} \quad \underset{k \times m}{\mathbf{V}_\beta^*}. \tag{20.12}$$

In order to evaluate (20.12), we merely need to form the matrix $\mathbf{K}(\tilde{I}_\alpha^{\mathrm{row}}, \tilde{I}_\beta^{\mathrm{col}})$.

Once the leaf nodes have all been processed, we proceed to the next coarser level. Consider for instance the node $\tau = 4$ with children $\alpha = 8$ and $\beta = 9$. Our task is in principle to build the long basis matrix $\mathbf{U}_4^{\mathrm{long}}$ that spans the columns of $\mathbf{K}(I_4, I_4^{\mathrm{c}})$, which is the shaded matrix inside the red rectangle in Figure 14(c). (We say "in principle" since it will not actually be explicitly formed.) To this end, we define the sample matrix

$$\mathbf{Y}_4 := \mathbf{K}(I_4, I_4^{\mathrm{c}})\mathbf{G}(I_4^{\mathrm{c}}, :).$$

The idea is now to repeat the same technique that we used for the leaf boxes, and write

$$\mathbf{Y}_4 = \mathbf{Y}(I_4, :) - \mathbf{K}(I_4, I_4)\mathbf{G}(I_4, :). \tag{20.13}$$

It turns out that we can evaluate (20.13) very efficiently: The block $\mathbf{K}(I_4, I_4)$ is made up of the two diagonal blocks $\mathbf{K}(I_8, I_8)$ and $\mathbf{K}(I_9, I_9)$, and the two off-diagonal blocks $\mathbf{K}(I_8, I_9)$ and $\mathbf{K}(I_9, I_8)$. Now observe that at this point in the execution of the algorithm, we have compressed representations of all these blocks available. Using this information, we form a short sample matrix $\tilde{\mathbf{Y}}_4$ of size $2k \times k$ that we can compress to build the basis matrix $\mathbf{U}_4$ and the associated index vector $\tilde{I}_4^{\mathrm{row}}$ in an ID of $\tilde{\mathbf{Y}}_4$. □

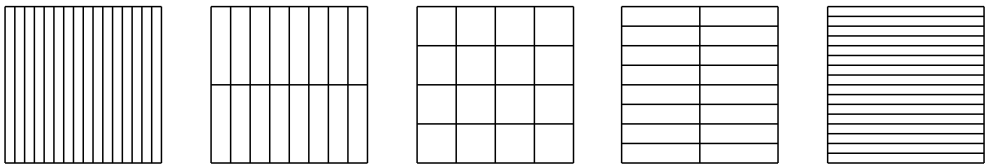


FIGURE 15. Illustration of the *butterfly* rank-structured matrix format described in Section 20.8. The figures show the blocks that must all be of numerically low rank for a butterfly matrix arising from a binary tree with 16 leaf nodes.

20.8. **Butterfly matrices.** An interesting class of rank-structured matrices arises from a generalization of the discrete Fourier transform. These so called "butterfly matrices" provide a data sparse format for many matrices that appear in the analysis of neural networks, wave propagation problems, and signal processing (Dao, Gu, Eichhorn, Rudra and R 2019, O'Neil 2007, Cands, Demanet and Ying 2009). This format involves an additional complication in comparison to the HODLR and HBS formats in that it requires $O(\log n)$ different tessellations of the matrix, as illustrated for a simple case in Figure 15. It can be demonstrated that when all of the resulting submatrices are of numerically low rank, the matrix as a whole can be written (approximately) as a product of $O(\log n)$ sparse matrices, in a manner analogous to the butterfly representation of an FFT (Briggs and Henson 1995, Sec. 10.4).

Randomization has proven to be a powerful tool for finding butterfly representations of matrices. The techniques involved are more complex than the methods described in Sections 20.4 – 20.6 due to the multiplicative nature of the representation, and typically involve iterative refinement rather than direct approximation (Li, Yang, Martin, Ho and Ying 2015*b*, Li and Yang 2017, Li, Yang and Ying 2018). Similar techniques played an essential role in a recent ground breaking paper (Guo, Liu, Hu and Michielssen 2017) that exploits butterfly representations to directly solve linear systems arising from the modeling of scattering problems in the high frequency regime.

20.9. **Applications of rank-structured matrices in data analysis.** The machinery for working with rank-structured hierarchical matrices that we have described can be used also for the kernel matrices discussed in Section 19 that arise in machine learning and computational statistics. For instance, Ambikasaran, Foreman-Mackey, Greengard, Hogg and ONeil (2015) demonstrate that data sparse formats of this type can be very effective for simulating Gaussian processes in low dimensional spaces.

When the underlying dimension grows, the techniques that we have described so far become uncompetitive. Even $d = 4$ would be considered a stretch. Fortunately, significant progress has recently been made towards extending the essential ideas to higher dimensions. For instance, March, Xiao and Biros (2015) describe a technique that is designed to uncover intrinsic lower dimensional structures that are often present in sets of points $\{\mathbf{x}_i\}_{i=1}^n$ that ostensibly live in higher dimensional spaces. The idea is to use randomized algorithms both for organizing the points into a hierarchical tree (that induces the tessellation of the matrix) and for computing low rank approximations to the resulting admissible blocks. The authors report promising numerical results for a wide selection of kernel matrices.

In this context, it is as we saw in Section 19 rarely possible to execute a matrix-vector multiplication, or even to evaluate more than a tiny fraction of the entries of the matrix. This means on the one hand that sampling must form an integral part of the compression strategy, and on the other that firm performance guarantees are typically not available. The saving grace is that when a kernel matrix is used for learning and data analysis, a rough approximation is often sufficient.

Remark 20.6 (Geometry oblivious methods)**.** *A curious observation is that techniques developed for kernel matrices appear to also be applicable for certain symmetric positive definite (pd) matrices that are not explicitly presented as kernel matrices. This is a consequence of the*

well known fact that any pd matrix $\mathbf{K}$ *admits a factorization*

$$\mathbf{K} = \mathbf{G}^*\mathbf{G} \tag{20.14}$$

for a so called "Gramian matrix" $\mathbf{G}$*. (If the eigenvalue decomposition of* $\mathbf{K}$ *takes the form* $\mathbf{K} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^*$*, then a matrix of the form* $\mathbf{G} = \mathbf{V}\mathbf{\Lambda}^{1/2}\mathbf{U}^*$ *is a Gramian if and only if* $\mathbf{V}$ *is unitary.) The factorization (20.14) says that the entries of* $\mathbf{K}$ *are formed by the inner products between the columns of* $\mathbf{G}$*,*

$$\mathbf{K}(i,j) = \langle \mathbf{g}_i,\, \mathbf{g}_j \rangle = k(\mathbf{g}_i, \mathbf{g}_i), \tag{20.15}$$

where $\mathbf{g}_i$ *is the* i*'th column of* $\mathbf{G}$ *(the "Gram vector") and where the* k *is the inner product kernel of Example 19.7. At this point, it becomes plausible that the techniques of (March et al. 2015) for kernel matrices associated with points in high dimensional spaces may apply to certain pd matrices. The key to make this work is the observation that it is not necessary to explicitly form the Gram factors* $\mathbf{G}$*. All that is needed in order to organize the points* $\{\mathbf{g}_i\}_{i=1}^n$ *are relative distances and angles between the points, and we can evaluate these from the matrix entries of* $\mathbf{K}$*, via the formula*

$$\|\mathbf{g}_i - \mathbf{g}_j\|^2 = \|\mathbf{g}_i\|^2 - 2Re\,\langle \mathbf{g}_i,\, \mathbf{g}_j \rangle + \|\mathbf{g}_j\|^2 = \mathbf{K}(i,i) - 2Re\,\mathbf{K}(i,j) + \mathbf{K}(j,j).$$

*The resulting technique was presented in (Yu, Levitt, Reiz and Biros 2017*a*) as a "geometry oblivious FMM (GOFMM)", along with numerical evidence of its usefulness for important classes of matrices.*

REFERENCES

D. Achlioptas (2003), Database-friendly random projections: Johnson-Lindenstrauss with binary coins, Vol. 66, pp. 671–687. Special issue on PODS 2001 (Santa Barbara, CA).

D. Achlioptas and F. McSherry (2001), Fast computation of low rank matrix approximations, in *Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing*, ACM, New York, pp. 611–618.

D. Achlioptas and F. McSherry (2007), 'Fast computation of low-rank matrix approximations', *J. ACM* **54**(2), Art. 9, 19.

R. Ahlswede and A. Winter (2002), 'Strong converse for identification via quantum channels', *IEEE Trans. Inform. Theory* **48**(3), 569–579.

N. Ailon and B. Chazelle (2006), Approximate nearest neighbors and the fast johnson-lindenstrauss transform, in *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, ACM, pp. 557–563.

N. Ailon and B. Chazelle (2009), 'The fast Johnson-Lindenstrauss transform and approximate nearest neighbors', *SIAM J. Comput.* **39**(1), 302–322.

A. Alaoui and M. W. Mahoney (2015), Fast randomized kernel ridge regression with statistical guarantees, in *Advances in Neural Information Processing Systems 28* (C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama and R. Garnett, eds), Curran Associates, Inc., pp. 775–783.

N. Alon, P. B. Gibbons, Y. Matias and M. Szegedy (2002), Tracking join and self-join sizes in limited storage, Vol. 64, pp. 719–747. Special issue on PODS 1999 (Philadelphia, PA).

N. Alon, Y. Matias and M. Szegedy (1999), The space complexity of approximating the frequency moments, Vol. 58, pp. 137–147. Twenty-eighth Annual ACM Symposium on the Theory of Computing (Philadelphia, PA, 1996).

S. Ambikasaran, D. Foreman-Mackey, L. Greengard, D. W. Hogg and M. ONeil (2015), 'Fast direct methods for gaussian processes', *IEEE transactions on pattern analysis and machine intelligence* **38**(2), 252–265.

D. Amelunxen, M. Lotz, M. B. McCoy and J. A. Tropp (2014), 'Living on the edge: phase transitions in convex programs with random data', *Inf. Inference* **3**(3), 224–294.

T. Ando (2005), *The Schur complement and its applications*, Vol. 4 of *Numerical Methods and Algorithms*, Springer-Verlag, New York, chapter Schur complements and matrix inequalities: Operator-theoretic approach, pp. 137–162.

M. A. Arcones and E. Gine (1992), 'On the bootstrap of u and v statistics', *Ann. Statist.* **20**(2), 655–674.

B. Arras, M. Bachmayr and A. Cohen (2019), 'Sequential sampling for optimal weighted least squares approximations in hierarchical spaces', *SIAM J. Math. Data Sci.* **1**(1), 189–207.

H. Avron (2018), 'Randomized Riemannian preconditioning for quadratically constrained problems'. Slides, Workshop on Randomized Numerical Linear Algebra, Simons Institute, UC-Berkeley.

H. Avron, K. Clarkson and D. Woodruff (2017), 'Faster kernel ridge regression using sketching and preconditioning', *SIAM Journal on Matrix Analysis and Applications* **38**(4), 1116–1138.

H. Avron, A. Druinsky and A. Gupta (2015), 'Revisiting asynchronous linear solvers: Provable convergence rate through randomization', *Journal of the ACM (JACM)* **62**(6), 51.

H. Avron, M. Kapralov, C. Musco, C. Musco, A. Velingker and A. Zandieh (2019), A universal sampling method for reconstructing signals with simple fourier transforms, in *Proceedings of the 51st Annual ACM SIGACT*

Symposium on Theory of Computing, STOC 2019, Association for Computing Machinery, New York, NY, USA, pp. 1051–1063.

H. Avron, P. Maymounkov and S. Toledo (2010), 'Blendenpik: Supercharging lapack's least-squares solver', *SIAM Journal on Scientific Computing* **32**(3), 1217–1236.

M. Baboulin, J. J. Dongarra, A. Rémy, S. Tomov and I. Yamazaki (2017), 'Solving dense symmetric indefinite systems using gpus', *Concurrency and Computation: Practice and Experience*.

M. Baboulin, X. S. Li and F.-H. Rouet (2014), Using random butterfly transformations to avoid pivoting in sparse direct methods, in *International Conference on High Performance Computing for Computational Science*, Springer, pp. 135–144.

F. Bach (2013), Sharp analysis of low-rank kernel matrix approximations, in *Conference on Learning Theory*, pp. 185–209.

F. Bach (2017), 'On the equivalence between kernel quadrature rules and random feature expansions', *Journal of Machine Learning Research* **18**(21), 1–38.

F. R. Bach and M. Jordan (2005), Predictive low-rank decomposition for kernel methods, in *ICML '05: Proceedings of the 22Nd International Conference on Machine Learning*, ACM, New York, NY, USA.

Z. Bai and J. W. Silverstein (2010), *Spectral analysis of large dimensional random matrices*, Springer Series in Statistics, second edn, Springer, New York.

Z. Bai, J. Demmel, J. Dongarra, A. Ruhe and H. van der Vorst (1987), *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide (Software, Environments and Tools)*, 1st edn, SIAM, Philadelphia, PA.

P. Baldi and R. Vershynin (2019), 'Polynomial threshold functions, hyperplane arrangements, and random tensors', *SIAM J. Math. Data Sci.* **1**(4), 699–729.

G. Ballard, J. Demmel, I. Dumitriu and A. Rusciano (2019), 'A generalized randomized rank-revealing factorization', *arXiv preprint arXiv:1909.06524*.

J. Banks, J. G. Vargas, A. Kulkarni and N. Srivastava (2019), 'Pseudospectral shattering, the sign function, and diagonalization in nearly matrix multiplication time'.

J. Barnes and P. Hut (1986), 'A hierarchical $o(n \log n)$ force-calculation algorithm', *Nature*.

J. Batson, D. A. Spielman and N. Srivastava (2014), 'Twice-Ramanujan sparsifiers', *SIAM Rev.* **56**(2), 315–334.

M. Bebendorf and R. Grzhibovskis (2006), 'Accelerating galerkin bem for linear elasticity using adaptive cross approximation', *Mathematical Methods in the Applied Sciences* **29**(14), 1721–1747.

R. Bhatia (1997), *Matrix analysis*, Vol. 169 of *Graduate Texts in Mathematics*, Springer-Verlag, New York.

C. H. Bischof and G. Quintana-Ortí (1998), 'Computing rank-revealing QR factorizations of dense matrices', *ACM Transactions on Mathematical Software* **24**(2), 226–253.

S. Bôchner (1933), 'Monotone Funktionen, Stieltjessche Integrale und harmonische Analyse', *Math. Ann.* **108**(1), 378–410.

L. Bottou (2010), Large-scale machine learning with stochastic gradient descent, in *Proceedings of COMPSTAT'2010* (Y. Lechevallier and G. Saporta, eds), Physica-Verlag HD, Heidelberg, pp. 177–186.

S. Boucheron, G. Lugosi and P. Massart (2013), *Concentration inequalities*, Oxford University Press, Oxford. A nonasymptotic theory of independence, With a foreword by Michel Ledoux.

J. Bourgain, S. Dirksen and J. Nelson (2015), 'Toward a unified theory of sparse dimensionality reduction in Euclidean space', *Geom. Funct. Anal.* **25**(4), 1009–1088.

C. Boutsidis, M. W. Mahoney and P. Drineas (2009), An improved approximation algorithm for the column subset selection problem, in *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SIAM, Philadelphia, PA, pp. 968–977.

C. Boutsidis, D. P. Woodruff and P. Zhong (2016), Optimal principal component analysis in distributed and streaming models, in *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, ACM, pp. 236–249.

W. L. Briggs and V. E. Henson (1995), *The DFT: An Owner's Manual for the Discrete Fourier Transform*, Society for Industrial and Applied Mathematics.

K. Bringmann and K. Panagiotou (2017), 'Efficient sampling methods for discrete distributions', *Algorithmica* **79**(2), 484–508.

E. Cands, L. Demanet and L. Ying (2009), 'A fast butterfly algorithm for the computation of fourier integral operators', *Multiscale Modeling & Simulation* **7**(4), 1727–1750.

B. Carl (1985), 'Inequalities of Bernstein-Jackson-type and the degree of compactness of operators in Banach spaces', *Ann. Inst. Fourier (Grenoble)* **35**(3), 79–118.

Y. Carmon, J. C. Duchi, A. Sidford and K. Tian (2019), A rank-1 sketch for matrix multiplicative weights, in *Proceedings of the Thirty-Second Conference on Learning Theory* (A. Beygelzimer and D. Hsu, eds), Vol. 99 of *Proceedings of Machine Learning Research*, PMLR, Phoenix, USA, pp. 589–623.

S. Chandrasekaran, M. Gu and W. Lyons (2005), 'A fast adaptive solver for hierarchically semiseparable representations', *Calcolo* **42**(3-4), 171–185.

V. Chandrasekaran, B. Recht, P. A. Parrilo and A. S. Willsky (2012), 'The convex geometry of linear inverse problems', *Found. Comput. Math.* **12**(6), 805–849.

M. Charikar, K. Chen and M. Farach-Colton (2004), Finding frequent items in data streams, Vol. 312, pp. 3–15. Automata, languages and programming.

L. H. Y. Chen, L. Goldstein and Q.-M. Shao (2011), *Normal approximation by Stein's method*, Probability and its Applications (New York), Springer, Heidelberg.

X. Chen and E. Price (2019), Active regression via linear-sample sparsification, in *Proceedings of the Thirty-Second Conference on Learning Theory* (A. Beygelzimer and D. Hsu, eds), Vol. 99 of *Proceedings of Machine Learning Research*, PMLR, Phoenix, USA, pp. 663–695.

Z. Chen and J. J. Dongarra (2005), 'Condition numbers of Gaussian random matrices', *SIAM J. Matrix Anal. Appl.* **27**(3), 603–620.

H. Cheng, Z. Gimbutas, P. Martinsson and V. Rokhlin (2005), 'On the compression of low rank matrices', *SIAM Journal of Scientific Computing* **26**(4), 1389–1404.

K. L. Clarkson and D. P. Woodruff (2009), Numerical linear algebra in the streaming model, in *Proceedings of the 41st annual ACM symposium on Theory of computing*, ACM, pp. 205–214.

K. L. Clarkson and D. P. Woodruff (2013), Low rank approximation and regression in input sparsity time, in *STOC'13—Proceedings of the 2013 ACM Symposium on Theory of Computing*, ACM, New York, pp. 81–90.

A. Cohen and G. Migliorati (2017), 'Optimal weighted least-squares methods', *SMAI J. Comput. Math.* **3**, 181–203.

A. Cohen, M. A. Davenport and D. Leviatan (2013), 'On the stability and accuracy of least squares approximations', *Found. Comput. Math.* **13**(5), 819–834.

E. Cohen and D. D. Lewis (1999), 'Approximating matrix multiplication for pattern recognition tasks', *Journal of Algorithms* **30**(2), 211 – 252.

M. B. Cohen (2016), Nearly tight oblivious subspace embeddings by trace inequalities, in *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, ACM, New York, pp. 278–287.

M. B. Cohen, R. Kyng, G. L. Miller, J. W. Pachocki, R. Peng, A. B. Rao and S. C. Xu (2014), Solving sdd linear systems in nearly mlog1/2n time, in *Proceedings of the Forty-sixth Annual ACM Symposium on Theory of Computing*, STOC '14, ACM, New York, NY, USA, pp. 343–352.

J. Cullum and W. E. Donath (1974), A block generalization of the s-step Lanczos algorithm, Technical Report Report RC 4845 (21570), IBM Thomas J. Watson Research Center, Yorktown Heights, New York.

T. Dao, A. Gu, M. Eichhorn, A. Rudra and C. R (2019), 'Learning fast algorithms for linear transforms using butterfly factorizations'.

K. R. Davidson and S. J. Szarek (2001), Local operator theory, random matrices and Banach spaces, in *Handbook of the geometry of Banach spaces, Vol. I*, North-Holland, Amsterdam, pp. 317–366.

T. A. Davis, S. Rajamanickam and W. M. Sid-Lakhdar (2016), 'A survey of direct methods for sparse linear systems', *Acta Numerica*.

J. Demmel, I. Dumitriu and O. Holtz (2007), 'Fast linear algebra is stable', *Numerische Mathematik* **108**(1), 59–91.

J. Demmel, L. Grigori, M. Hoemmen and J. Langou (2012), 'Communication-optimal parallel and sequential QR and LU factorizations', *SIAM J. Sci. Comput.* **34**(1), A206–A239.

J. W. Demmel, L. Grigori, M. Gu and H. Xiang (2015), 'Communication avoiding rank revealing qr factorization with column pivoting', *SIAM Journal on Matrix Analysis and Applications* **36**(1), 55–89.

J. D. Dixon (1983), 'Estimating extremal eigenvalues and condition numbers of matrices', *SIAM J. Numer. Anal.* **20**(4), 812–814.

E. Dobriban and S. Liu (2018), 'Asymptotics for sketching in least squares regression'.

P. Drineas and M. Mahoney (2018), Lectures on randomized linear algebra, in *The Mathematics of Data* (M. Mahoney, J. Duchi and A. Gilbert, eds), Vol. 25, American Mathematical Society, chapter 4, pp. 1 – 48. IAS/Park City Mathematics Series.

P. Drineas and M. W. Mahoney (2005), 'On the Nyström method for approximating a Gram matrix for improved kernel-based learning', *J. Mach. Learn. Res.* **6**, 2153–2175.

P. Drineas, R. Kannan and M. W. Mahoney (2006*a*), 'Fast Monte Carlo algorithms for matrices. I. Approximating matrix multiplication', *SIAM J. Comput.* **36**(1), 132–157.

P. Drineas, R. Kannan and M. W. Mahoney (2006*b*), 'Fast Monte Carlo algorithms for matrices. II. Computing a low-rank approximation to a matrix', *SIAM J. Comput.* **36**(1), 158–183.

P. Drineas, R. Kannan and M. W. Mahoney (2006*c*), 'Fast Monte Carlo algorithms for matrices. III. Computing a compressed approximate matrix decomposition', *SIAM J. Comput.* **36**(1), 184–206.

P. Drineas, M. W. Mahoney and S. Muthukrishnan (2006*d*), Subspace sampling and relative-error matrix approximation: column-based methods, in *Approximation, randomization and combinatorial optimization*, Vol. 4110 of *Lecture Notes in Comput. Sci.*, Springer, Berlin, pp. 316–326.

P. Drineas, M. W. Mahoney and S. Muthukrishnan (2008), 'Relative-error CUR matrix decompositions', *SIAM J. Matrix Anal. Appl.* **30**(2), 844–881.

J. A. Duersch and M. Gu (2017), 'Randomized qr with column pivoting', *SIAM Journal on Scientific Computing* **39**(4), C263–C291.

J. Duersch and M. Gu (2015), 'True blas-3 performance qrcp using random sampling'. arXiv preprint #1509.06820.

A. S. Edelman (1989), *Eigenvalues and condition numbers of random matrices*, ProQuest LLC, Ann Arbor, MI. Thesis (Ph.D.)–Massachusetts Institute of Technology.

B. Efron (1982), *The jackknife, the bootstrap and other resampling plans*, Vol. 38 of *CBMS-NSF Regional Conference Series in Applied Mathematics*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, Pa.

D. Feldman, M. Volkov and D. Rus (2016), Dimensionality reduction of massive sparse datasets using coresets, in *Advances in Neural Information Processing Systems*, pp. 2766–2774.

Y. Feng, J. Xiao and M. Gu (2018), 'Low-rank matrix approximations with flip-flop spectrum-revealing qr factorization', *arXiv preprint arXiv:1803.01982*.

R. D. Fierro, P. C. Hansen and P. S. K. Hansen (1999), 'Utv tools: Matlab templates for rank-revealing utv decompositions', *Numerical Algorithms* **20**(2-3), 165–194.

S. Fine and K. Scheinberg (2001), 'Efficient SVM training using low-rank kernel representation', *Journal of Machine Learning Research* **2**, 243–264.

J. K. Fitzsimons, M. A. Osborne, S. J. Roberts and J. F. Fitzsimons (2018), Improved stochastic trace estimators using mutually unbiased bases, in *Uncertainty in Artificial Intelligence: Proceedings of the Thirty-Fourth Conference* (A. Globerson and R. Silva, eds), AUAI Press, Monterey, CA.

S. Foucart and H. Rauhut (2013), *A mathematical introduction to compressive sensing*, Applied and Numerical Harmonic Analysis, Birkhäuser/Springer, New York.

A. Frieze, R. Kannan and S. Vempala (2004), 'Fast Monte-Carlo algorithms for finding low-rank approximations', *J. ACM* **51**(6), 1025–1041.

M. Ghashami, E. Liberty, J. M. Phillips and D. P. Woodruff (2016*a*), 'Frequent directions: simple and deterministic matrix sketching', *SIAM J. Comput.* **45**(5), 1762–1792.

M. Ghashami, D. J. Perry and J. Phillips (2016*b*), Streaming kernel principal component analysis, in *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics* (A. Gretton and C. C. Robert, eds), Vol. 51 of *Proceedings of Machine Learning Research*, PMLR, Cadiz, Spain, pp. 1365–1374.

P. Ghysels, X. S. Li, C. Gorman and F.-H. Rouet (2017), A robust parallel preconditioner for indefinite systems using hierarchical matrices and randomized sampling, in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, IEEE, pp. 897–906.

A. Gionis, P. Indyk and R. Motwani (1999), Similarity search in high dimensions via hashing, in *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 518–529.

D. A. Girard (1989), 'A fast "Monte Carlo cross-validation" procedure for large least squares problems with noisy data', *Numer. Math.* **56**(1), 1–23.

A. Gittens (2013), *Topics in Randomized Numerical Linear Algebra*, ProQuest LLC, Ann Arbor, MI. Thesis (Ph.D.)–California Institute of Technology.

L. Goldstein, I. Nourdin and G. Peccati (2017), 'Gaussian phase transitions and conic intrinsic volumes: Steining the Steiner formula', *Ann. Appl. Probab.* **27**(1), 1–47.

G. H. Golub and G. Meurant (1994), Matrices, moments and quadrature, in *Numerical analysis 1993 (Dundee, 1993)*, Vol. 303 of *Pitman Res. Notes Math. Ser.*, Longman Sci. Tech., Harlow, pp. 105–156.

G. H. Golub and G. Meurant (2010), *Matrices, moments and quadrature with applications*, Princeton Series in Applied Mathematics, Princeton University Press, Princeton, NJ.

G. H. Golub and R. Underwood (1977), The block Lanczos method for computing eigenvalues, in *Mathematical software, III (Proc. Sympos., Math. Res. Center, Univ. Wisconsin, Madison, Wis., 1977)*, Academic Press, New York, pp. 361–377. Publ. Math. Res. Center, No. 39.

G. H. Golub and C. F. Van Loan (2013), *Matrix computations*, Johns Hopkins Studies in the Mathematical Sciences, fourth edn, Johns Hopkins University Press, Baltimore, MD.

G. H. Golub, F. T. Luk and M. L. Overton (1981), 'A block Lánczos method for computing the singular values of corresponding singular vectors of a matrix', *ACM Trans. Math. Software* **7**(2), 149–169.

I. J. Good (1977), 'A new formula for k-statistics', *Ann. Statist.* **5**(1), 224–228.

A. Gopal and P.-G. Martinsson (2018), 'The PowerURV algorithm for computing rank-revealing full factorizations'.

Y. Gordon (1988), On Milman's inequality and random subspaces which escape through a mesh in $\mathbf{R}^n$, in *Geometric aspects of functional analysis (1986/87)*, Vol. 1317 of *Lecture Notes in Math.*, Springer, Berlin, pp. 84–106.

S. A. Goreinov, I. V. Oseledets, D. V. Savostyanov, E. E. Tyrtyshnikov and N. L. Zamarashkin (2010), How to find a good submatrix, in *Matrix Methods: Theory, Algorithms And Applications: Dedicated to the Memory of Gene Golub*, World Scientific, pp. 247–256.

S. Goreinov, N. Zamarashkin and E. Tyrtyshnikov (1997), 'Pseudo-skeleton approximations by matrices of maximal volume', *Mathematical Notes*.

R. Gower and P. Richtarik (2015*a*), 'Randomized iterative methods for linear systems', *SIAM Journal on Matrix Analysis and Applications* **36**(4), 1660–1690.

R. Gower, F. Hanzely, P. Richtarik and S. U. Stich (2018), Accelerated stochastic matrix inversion: General theory and speeding up bfgs rules for faster second-order optimization, in *Advances in Neural Information Processing Systems 31* (S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi and R. Garnett, eds), Curran Associates, Inc., pp. 1619–1629.

R. M. Gower and P. Richtárik (2015*b*), 'Randomized iterative methods for linear systems', *SIAM J. Matrix Anal. Appl.* **36**(4), 1660–1690.

L. Grasedyck and W. Hackbusch (2003), 'Construction and arithmetics of $\mathcal{H}$-matrices', *Computing* **70**(4), 295–334.

S. Gratton and D. Titley-Peloquin (2018), 'Improved bounds for small-sample estimation', *SIAM J. Matrix Anal. Appl.* **39**(2), 922–931.

L. Greengard and V. Rokhlin (1987), 'A fast algorithm for particle simulations', *J. Comput. Phys.* **73**(2), 325–348.

G. R. Grimmett and D. R. Stirzaker (2001), *Probability and random processes*, third edn, Oxford University Press, New York.

M. Gu (2015), 'Subspace iteration randomization and singular value problems', *SIAM J. Sci. Comput.* **37**(3), A1139–A1173.

M. Gu and S. C. Eisenstat (1996), 'Efficient algorithms for computing a strong rank-revealing QR factorization', *SIAM J. Sci. Comput.* **17**(4), 848–869.

H. Guo, Y. Liu, J. Hu and E. Michielssen (2017), 'A butterfly-based direct integral-equation solver using hierarchical lu factorization for analyzing scattering from electrically large conducting objects', *IEEE Transactions on Antennas and Propagation* **65**(9), 4742–4750.

W. Hackbusch (1999), 'A sparse matrix arithmetic based on H-matrices; Part I: Introduction to H-matrices', *Computing* **62**, 89–108.

W. Hackbusch, B. Khoromskij and S. Sauter (2002), On $\mathcal{H}^2$-matrices, in *Lectures on Applied Mathematics*, Springer Berlin, pp. 9–29.

N. Halko, P. G. Martinsson and J. A. Tropp (2011*a*), 'Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions', *SIAM Rev.* **53**(2), 217–288.

N. Halko, P.-G. Martinsson, Y. Shkolnisky and M. Tygert (2011*b*), 'An algorithm for the principal component analysis of large data sets', *SIAM J. Sci. Comput.* **33**(5), 2580–2594.

R. Hamid, Y. Xiao, A. Gittens and D. Decoste (2014), Compact random feature maps, in *Proceedings of the 31st International Conference on Machine Learning* (E. P. Xing and T. Jebara, eds), Vol. 32 of *Proceedings of Machine Learning Research*, PMLR, Bejing, China, pp. 19–27.

J. Hampton and A. Doostan (2015), 'Coherence motivated sampling and convergence analysis of least squares polynomial chaos regression', *Computer Methods in Applied Mechanics and Engineering* **290**, 73 – 97.

P. Hennig and M. A. Osborne (2019), 'probabilistic-numerics.org'.

M. R. Hestenes and E. Stiefel (1952), *Methods of conjugate gradients for solving linear systems*, Vol. 49, NBS Washington, DC.

R. A. Horn and C. R. Johnson (2013), *Matrix analysis*, second edn, Cambridge University Press, Cambridge.

M. F. Hutchinson (1990), 'A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines', *Comm. Statist. Simulation Comput.* **19**(2), 433–450.

P. Indyk and R. Motwani (1999), Approximate nearest neighbors: towards removing the curse of dimensionality, in *STOC '98 (Dallas, TX)*, ACM, New York, pp. 604–613.

R. Jin, T. G. Kolda and R. Ward (2019), 'Faster johnson-lindenstrauss transforms via kronecker products'.

W. B. Johnson and J. Lindenstrauss (1984), Extensions of Lipschitz mappings into a Hilbert space, in *Conference in modern analysis and probability (New Haven, Conn., 1982)*, Vol. 26 of *Contemp. Math.*, Amer. Math. Soc., Providence, RI, pp. 189–206.

I. M. Johnstone (2001), 'On the distribution of the largest eigenvalue in principal components analysis', *Ann. Statist.* **29**(2), 295–327.

W. D. Joubert and G. F. Carey (1991), Parellelizable restarted iterative methods for nonsymmetric linear systems, Center for Numerical Analysis Report CNA-251, UT-Austin.

M. Kac (1956), Foundations of kinetic theory, in *Proceedings of the Third Berkeley Symposium on Mathematical Statistics and Probability, 1954–1955, vol. III*, University of California Press, Berkeley and Los Angeles, pp. 171–197.

W. Kahan (1966), 'Numerical linear algebra', *Canadian Math. Bull* **9**(6), 757–801.

R. Kannan and S. Vempala (2017), 'Randomized algorithms in numerical linear algebra', *Acta Numerica* **26**, 95–135.

P. Kar and H. Karnick (2012), Random feature maps for dot product kernels, in *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics* (N. D. Lawrence and M. Girolami, eds), Vol. 22 of *Proceedings of Machine Learning Research*, PMLR, La Palma, Canary Islands, pp. 583–591.

S. P. Kasiviswanathan, M. Rudelson, A. Smith and J. Ullman (2010), The price of privately releasing contingency tables and the spectra of random matrices with correlated rows, in *STOC'10—Proceedings of the 2010 ACM International Symposium on Theory of Computing*, ACM, New York, pp. 775–784.

W. Kong and G. Valiant (2017), 'Spectrum estimation from samples', *Ann. Statist.* **45**(5), 2218–2247.

V. S. Koroljuk and Y. V. Borovskich (1994), *Theory of U-statistics*, Vol. 273 of *Mathematics and its Applications*, Kluwer Academic Publishers Group, Dordrecht. Translated from the 1989 Russian original by P. V. Malyshev and D. V. Malyshev and revised by the authors.

F. Krahmer and R. Ward (2011), 'New and improved johnson-lindenstrauss embeddings via the restricted isometry property', *SIAM J. Math. Anal.* **43**(3), 1269–1281.

D. Kressner, M. Steinlechner and B. Vandereycken (2016), 'Preconditioned low-rank riemannian optimization for linear systems with tensor product structure', *SIAM Journal on Scientific Computing* **38**(4), A2018–A2044.

J. Kuczyński and H. Woźniakowski (1992), 'Estimating the largest eigenvalue by the power and Lanczos algorithms with a random start', *SIAM J. Matrix Anal. Appl.* **13**(4), 1094–1122.

R. Kueng (2019), 2-designs minimize variance of trace estimators, Unpublished manuscript.

S. Kumar, M. Mohri and A. Talwalkar (2012), 'Sampling methods for the Nyström method', *J. Mach. Learn. Res.* **13**, 981–1006.

S. Kurz, O. Rain and S. Rjasanow (2002), 'The adaptive cross-approximation technique for the 3d boundary-element method', *IEEE transactions on Magnetics* **38**(2), 421–424.

R. Kyng (2017), *Approximate Gaussian elimination*, ProQuest LLC, Ann Arbor, MI. Thesis (Ph.D.)–Yale University.

R. Kyng and S. Sachdeva (2016), Approximate Gaussian elimination for Laplacians—fast, sparse, and simple, in *57th Annual IEEE Symposium on Foundations of Computer Science—FOCS 2016*, IEEE Computer Soc., Los Alamitos, CA, pp. 573–582.

R. Kyng, Y. T. Lee, R. Peng, S. Sachdeva and D. A. Spielman (2016), Sparsified Cholesky and multigrid solvers for connection Laplacians, in *STOC'16—Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing*, ACM, New York, pp. 842–850.

Q. Le, T. Sarlós and A. Smola (2013), Fastfood - computing hilbert space expansions in loglinear time, in *Proceedings of the 30th International Conference on Machine Learning* (S. Dasgupta and D. McAllester, eds), Vol. 28 of *Proceedings of Machine Learning Research*, PMLR, Atlanta, Georgia, USA, pp. 244–252.

M. Ledoux and M. Talagrand (1991), *Probability in Banach spaces*, Vol. 23 of *Ergebnisse der Mathematik und ihrer Grenzgebiete (3) [Results in Mathematics and Related Areas (3)]*, Springer-Verlag, Berlin. Isoperimetry and processes.

D. Leventhal and A. S. Lewis (2010), 'Randomized methods for linear constraints: convergence rates and conditioning', *Math. Oper. Res.* **35**(3), 641–654.

Z. Leyk and H. Woźniakowski (1998), 'Estimating a largest eigenvector by Lanczos and polynomial algorithms with a random start', *Numer. Linear Algebra Appl.* **5**(3), 147–164.

H. Li, G. C. Linderman, A. Szlam, K. P. Stanton, Y. Kluger and M. Tygert (2017), 'Algorithm 971: an implementation of a randomized algorithm for principal component analysis', *ACM Trans. Math. Software* **43**(3), Art. 28, 14.

M. Li, W. Bi, J. T. Kwok and B. Lu (2015*a*), 'Large-scale nyström kernel matrix approximation using randomized svd', *IEEE Transactions on Neural Networks and Learning Systems* **26**(1), 152–164.

Y. Li and H. Yang (2017), 'Interpolative butterfly factorization', *SIAM Journal on Scientific Computing* **39**(2), A503–A531.

Y. Li, H. L. Nguyen and D. P. Woodruff (2014*a*), On sketching matrix norms and the top singular vector, in *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, ACM, New York, pp. 1562–1581.

Y. Li, H. L. Nguyen and D. P. Woodruff (2014*b*), Turnstile streaming algorithms might as well be linear sketches, in *STOC'14—Proceedings of the 2014 ACM Symposium on Theory of Computing*, ACM, New York, pp. 174–183.

Y. Li, H. Yang and L. Ying (2018), 'Multidimensional butterfly factorization', *Applied and Computational Harmonic Analysis* **44**(3), 737 – 758.

Y. Li, H. Yang, E. R. Martin, K. L. Ho and L. Ying (2015*b*), 'Butterfly factorization', *Multiscale Modeling & Simulation* **13**(2), 714–732.

E. Liberty (2009), Accelerated Dense Random Projections, PhD thesis, Computer Science, Yale University.

E. Liberty, F. Woolfe, P.-G. Martinsson, V. Rokhlin and M. Tygert (2007), 'Randomized algorithms for the low-rank approximation of matrices', *Proc. Natl. Acad. Sci. USA* **104**(51), 20167–20172.

L.-H. Lim and J. Weare (2017), 'Fast randomized iteration: Diffusion monte carlo through the lens of numerical linear algebra', *SIAM Review* **59**(3), 547–587.

L. Lin, J. Lu and L. Ying (2011), 'Fast construction of hierarchical matrix representation from matrix–vector multiplication', *Journal of Computational Physics* **230**(10), 4071 – 4087.

N. Linial, E. London and Y. Rabinovich (1995), 'The geometry of graphs and some of its algorithmic applications', *Combinatorica* **15**(2), 215–245.

M. E. Lopes (2019), 'Estimating the algorithmic variance of randomized ensembles via the bootstrap', *Ann. Statist.* **47**(2), 1088–1112.

D. Lopez-Paz, S. Sra, A. Smola, Z. Ghahramani and B. Schölkopf (2014), Randomized nonlinear component analysis, in *Proceedings of the 31st International Conference on Machine Learning* (E. P. Xing and T. Jebara, eds), Vol. 32 of *Proceedings of Machine Learning Research*, PMLR, Bejing, China, pp. 1359–1367.

F. Lust-Piquard (1986), 'Inégalités de Khintchine dans C_p $(1 < p < \infty)$', *C. R. Acad. Sci. Paris Sér. I Math.* **303**(7), 289–292.

L. Mackey, M. I. Jordan, R. Y. Chen, B. Farrell and J. A. Tropp (2014), 'Matrix concentration inequalities via the method of exchangeable pairs', *Ann. Probab.* **42**(3), 906–945.

M. W. Mahoney (2011), 'Randomized algorithms for matrices and data', *Foundations and Trends® in Machine Learning* **3**(2), 123–224.

M. W. Mahoney and P. Drineas (2009), 'Cur matrix decompositions for improved data analysis', *Proceedings of the National Academy of Sciences* **106**(3), 697–702.

O. A. Malik and S. Becker (2019), 'Guarantees for the kronecker fast johnson-lindenstrauss transform using a coherence and sampling argument'.

W. B. March, B. Xiao and G. Biros (2015), 'Askit: Approximate skeletonization kernel-independent treecode in high dimensions', *SIAM Journal on Scientific Computing* **37**(2), A1089–A1110.

M. B. Marcus and G. Pisier (1981), *Random Fourier series with applications to harmonic analysis*, Vol. 101 of *Annals of Mathematics Studies*, Princeton University Press, Princeton, N.J.; University of Tokyo Press, Tokyo.

P. Martinsson (2011), 'A fast randomized algorithm for computing a hierarchically semiseparable representation of a matrix', *SIAM Journal on Matrix Analysis and Applications* **32**(4), 1251–1274.

P. Martinsson (2015), 'Blocked rank-revealing qr factorizations: How randomized sampling can be used to avoid single-vector pivoting'. arXiv preprint #1505.08115.

P. Martinsson and V. Rokhlin (2005), 'A fast direct solver for boundary integral equations in two dimensions', *J. Comp. Phys.* **205**(1), 1–23.

P. Martinsson and S. Voronin (2016), 'A randomized blocked algorithm for efficiently computing rank-revealing factorizations of matrices', *SIAM Journal on Scientific Computing* **38**(5), S485–S507.

P.-G. Martinsson (2016), 'Compressing rank-structured matrices via randomized sampling', *SIAM Journal on Scientific Computing* **38**(4), A1959–A1986.

P.-G. Martinsson (2018), Randomized methods for matrix computations, in *The Mathematics of Data* (M. Mahoney, J. Duchi and A. Gilbert, eds), Vol. 25, American Mathematical Society, chapter 4, pp. 187 – 231. IAS/Park City Mathematics Series.

P.-G. Martinsson (2019), *Fast Direct Solvers for Elliptic PDEs*, Vol. CB96 of *CBMS-NSF conference series*, SIAM.

P.-G. Martinsson, V. Rokhlin and M. Tygert (2006*a*), A randomized algorithm for the approximation of matrices, Technical Report Yale CS research report YALEU/DCS/RR-1361, Yale University, Computer Science Department.

P. Martinsson, G. Quintana Orti and N. Heavner (2019), 'randUTV: A blocked randomized algorithm for computing a rank-revealing UTV factorization', *arXiv preprint arXiv:1703.00998*. Accepted for publication by ACM TOMS.

P. Martinsson, G. Quintana-Ortí, N. Heavner and R. van de Geijn (2017), 'Householder qr factorization with randomization for column pivoting (hqrrp)', *SIAM Journal on Scientific Computing* **39**(2), C96–C115.

P. Martinsson, V. Rokhlin and M. Tygert (2006*b*), 'On interpolation and integration in finite-dimensional spaces of bounded functions', *Comm. Appl. Math. Comput. Sci* pp. 133–142.

R. Mathias and G. Stewart (1993), 'A block {QR} algorithm and the singular value decomposition', *Linear Algebra and its Applications* **182**, 91 – 100.

M. B. McCoy and J. A. Tropp (2013), The achievable performance of convex demixing, ACM Technical Report 2017-02, Caltech, Pasadena, CA.

M. B. McCoy and J. A. Tropp (2014), 'From Steiner formulas for cones to concentration of intrinsic volumes', *Discrete Comput. Geom.* **51**(4), 926–963.

C. Melgaard and M. Gu (2015), 'Gaussian elimination with randomized complete pivoting', *arXiv preprint arXiv:1511.08528*.

S. Mendelson, H. Rauhut and R. Ward (2018), 'Improved bounds for sparse recovery from subsampled random convolutions', *Ann. Appl. Probab.* **28**(6), 3491–3527.

X. Meng and M. W. Mahoney (2013), Low-distortion subspace embeddings in input-sparsity time and applications to robust linear regression, in *STOC'13—Proceedings of the 2013 ACM Symposium on Theory of Computing*, ACM, New York, pp. 91–100.

X. Meng, M. A. Saunders and M. W. Mahoney (2014), 'LSRN: a parallel iterative solver for strongly over- or underdetermined systems', *SIAM J. Sci. Comput.* **36**(2), C95–C118.

F. Mezzadri (2007), 'How to generate random matrices from the classical compact groups', *Notices Amer. Math. Soc.* **54**(5), 592–604.

E. Moulines and F. R. Bach (2011), Non-asymptotic analysis of stochastic approximation algorithms for machine learning, in *Advances in Neural Information Processing Systems*, pp. 451–459.

R. J. Muirhead (1982), *Aspects of multivariate statistical theory*, John Wiley & Sons, Inc., New York. Wiley Series in Probability and Mathematical Statistics.

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

C. Musco and C. Musco (2017), Recursive sampling for the nystrom method, in *Advances in Neural Information Processing Systems 30* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan and R. Garnett, eds), Curran Associates, Inc., pp. 3833–3845.

C. Musco, C. Musco and A. Sidford (2018), Stability of the Lanczos method for matrix function approximation, in *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SIAM, Philadelphia, PA, pp. 1605–1624.

S. Muthukrishnan (2005), 'Data streams: Algorithms and applications', *Foundations and Trends® in Theoretical Computer Science* **1**(2), 117–236.

T. N. A. G. (NAG) (2019), 'The nag library mark 27'. https://www.nag.com/content/naglibrary-mark27.

R. M. Neal (1996), *Priors for Infinite Networks*, Springer New York, New York, NY, pp. 29–53.

D. Needell and J. A. Tropp (2014), 'Paved with good intentions: analysis of a randomized block kaczmarz method', *Linear Algebra and its Applications* **441**, 199–221.

D. Needell, R. Ward and N. Srebro (2014), Stochastic gradient descent, weighted sampling, and the randomized kaczmarz algorithm, in *Advances in neural information processing systems*, pp. 1017–1025.

J. Nelson and H. L. Nguyen (2013), OSNAP: faster numerical linear algebra algorithms via sparser subspace embeddings, in *2013 IEEE 54th Annual Symposium on Foundations of Computer Science—FOCS 2013*, IEEE Computer Soc., Los Alamitos, CA, pp. 117–126.

E. J. Nyström (1930), 'Über Die Praktische Auflösung von Integralgleichungen mit Anwendungen auf Randwertaufgaben', *Acta Math.* **54**(1), 185–204.

E. Oja (1982), 'A simplified neuron model as a principal component analyzer', *J. Math. Biol.* **15**(3), 267–273.

R. I. Oliveira (2009*a*), 'Concentration of the adjacency matrix and of the laplacian in random graphs with independent edges'.

R. I. Oliveira (2009*b*), 'On the convergence to equilibrium of Kac's random walk on matrices', *Ann. Appl. Probab.* **19**(3), 1200–1231.

M. O'Neil (2007), A new class of analysis-based fast transforms, PhD thesis, Mathematics, Yale University.

S. Oymak and J. A. Tropp (2018), 'Universality laws for randomized dimension reduction, with applications', *Inf. Inference* **7**(3), 337–446.

R. Pagh (2013), 'Compressed matrix multiplication', *ACM Trans. Comput. Theory*.

V. Y. Pan and L. Zhao (2017), 'Numerically safe gaussian elimination with no pivoting', *Linear Algebra and its Applications* **527**, 349 – 383.

C. H. Papadimitriou, P. Raghavan, H. Tamaki and S. Vempala (2000), Latent semantic indexing: a probabilistic analysis, Vol. 61, pp. 217–235. Special issue on the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (Seattle, WA, 1998).

H. Park and L. Eldén (1995), 'Downdating the rank-revealing urv decomposition', *SIAM Journal on Matrix Analysis and Applications* **16**(1), 138–155.

D. Parker (1995), Random butterfly transformations with applications in computational linear algebra, Technical Report CSD-950023, UCLA.

B. N. Parlett (1998), *The symmetric eigenvalue problem*, Vol. 20 of *Classics in Applied Mathematics*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA. Corrected reprint of the 1980 original.

N. Pham and R. Pagh (2013), Fast and scalable polynomial kernels via explicit feature maps, in *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, ACM, New York, NY, USA, pp. 239–247.

M. Pilanci and M. J. Wainwright (2015), 'Randomized sketches of convex programs with sharp guarantees', *IEEE Trans. Inform. Theory* **61**(9), 5096–5115.

M. Pilanci and M. J. Wainwright (2016), 'Iterative Hessian sketch: fast and accurate solution approximation for constrained least-squares', *J. Mach. Learn. Res.* **17**, Paper No. 53, 38.

N. S. Pillai and A. Smith (2017), 'Kac's walk on n-sphere mixes in $n \log n$ steps', *Ann. Appl. Probab.* **27**(1), 631–650.

U. Porod (1996), 'The cut-off phenomenon for random reflections', *Ann. Probab.* **24**(1), 74–96.

F. Pourkamali-Anaraki and S. Becker (2019), 'Improved fixed-rank nyström approximation via qr decomposition: Practical and theoretical aspects', *Neurocomputing* **363**, 261 – 272.

A. Rahimi and B. Recht (2008), Random features for large-scale kernel machines, in *Advances in Neural Information Processing Systems 20* (J. C. Platt, D. Koller, Y. Singer and S. T. Roweis, eds), Curran Associates, Inc., pp. 1177–1184.

A. Rahimi and B. Recht (2009), Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning, in *Advances in Neural Information Processing Systems 21* (D. Koller, D. Schuurmans, Y. Bengio and L. Bottou, eds), Curran Associates, Inc., pp. 1313–1320.

H. Rauhut and R. Ward (2012), 'Sparse Legendre expansions via ℓ_1-minimization', *J. Approx. Theory* **164**(5), 517–533.

H. Rauhut and R. Ward (2016), 'Interpolation via weighted ℓ_1 minimization', *Appl. Comput. Harmon. Anal.* **40**(2), 321–351.

H. Rauhut, J. Romberg and J. A. Tropp (2012), 'Restricted isometries for partial random circulant matrices', *Appl. Comput. Harmon. Anal.* **32**(2), 242–254.

P. Richtárik and M. Takáč (2017), 'Stochastic reformulations of linear systems: Algorithms and convergence theory'.

V. Rokhlin and M. Tygert (2008), 'A fast randomized algorithm for overdetermined linear least-squares regression', *Proc. Natl. Acad. Sci. USA* **105**(36), 13212–13217.

V. Rokhlin, A. Szlam and M. Tygert (2009), 'A randomized algorithm for principal component analysis', *SIAM Journal on Matrix Analysis and Applications* **31**(3), 1100–1124.

J. S. Rosenthal (1994), 'Random rotations: characters and random walks on $\mathrm{SO}(N)$', *Ann. Probab.* **22**(1), 398–423.

N. Ross (2011), 'Fundamentals of Stein's method', *Probab. Surv.* **8**, 210–293.

M. Rudelson (1999), 'Random vectors in the isotropic position', *J. Funct. Anal.* **164**(1), 60–72.

M. Rudelson (2012), 'Row products of random matrices', *Adv. Math.* **231**(6), 3199–3231.

M. Rudelson and R. Vershynin (2007), 'Sampling from large matrices: an approach through geometric functional analysis', *J. ACM* **54**(4), Art. 21, 19.

M. Rudelson and R. Vershynin (2008), 'On sparse reconstruction from Fourier and Gaussian measurements', *Comm. Pure Appl. Math.* **61**(8), 1025–1045.

A. Rudi and L. Rosasco (2017), Generalization properties of learning with random features, in *Advances in Neural Information Processing Systems 30* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan and R. Garnett, eds), Curran Associates, Inc., pp. 3215–3225.

A. Rudi, D. Calandriello, L. Carratino and L. Rosasco (2018), On fast leverage score sampling and optimal learning, in *Advances in Neural Information Processing Systems 31* (S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi and R. Garnett, eds), Curran Associates, Inc., pp. 5672–5682.

A. Rudi, R. Camoriano and L. Rosasco (2015), Less is more: Nyström computational regularization, in *Advances in Neural Information Processing Systems 28* (C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama and R. Garnett, eds), Curran Associates, Inc., pp. 1657–1665.

A. Rudi, L. Carratino and L. Rosasco (2017), Falkon: An optimal large scale kernel method, in *Advances in Neural Information Processing Systems 30* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan and R. Garnett, eds), Curran Associates, Inc., pp. 3888–3898.

Y.-L. K. Samo and S. Roberts (2015), 'Generalized spectral kernels'.

A. Sankar, D. A. Spielman and S.-H. Teng (2006), 'Smoothed analysis of the condition numbers and growth factors of matrices', *SIAM J. Matrix Anal. Appl.* **28**(2), 446–476.

T. Sarlós (2006), Improved approximation algorithms for large matrices via random projections, in *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pp. 143–152.

I. J. Schoenberg (1942), 'Positive definite functions on spheres', *Duke Math. J.* **9**, 96–108.

B. Schölkopf and A. Smola (2001), *Learning with Kernels*, MIT Press, Cambridge, MA.

B. Schölkopf, A. Smola and K.-R. Müller (1996), Nonlinear component analysis as a kernel eigenvalue problem, Technical report 44, Max-Planck-Institut für biologische Kybernetik.

M. Simchowitz, A. E. Alaoui and B. Recht (2017), 'On the gap between strict-saddles and true convexity: An omega(log d) lower bound for eigenvector approximation'.

N. J. A. Sloane (1983), Encrypting by random rotations, in *Cryptography (Burg Feuerstein, 1982)*, Vol. 149 of *Lecture Notes in Comput. Sci.*, Springer, Berlin, pp. 71–128.

D. C. Sorensen and M. Embree (2016), 'A deim induced cur factorization', *SIAM Journal on Scientific Computing* **38**(3), A1454–A1482.

D. A. Spielman and N. Srivastava (2011), 'Graph sparsification by effective resistances', *SIAM J. Comput.* **40**(6), 1913–1926.

B. Sriperumbudur and Z. Szabo (2015), Optimal rates for random fourier features, in *Advances in Neural Information Processing Systems 28* (C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama and R. Garnett, eds), Curran Associates, Inc., pp. 1144–1152.

G. Stewart (1994), 'UTV decompositions', *PITMAN RESEARCH NOTES IN MATHEMATICS SERIES* pp. 225–225.

G. Stewart (1998), *Matrix Algorithms Volume 1: Basic Decompositions*, SIAM.

G. Stewart (1999), 'The QLP approximation to the singular value decomposition', *SIAM Journal on Scientific Computing* **20**(4), 1336–1348.

G. W. Stewart (2001), *Matrix algorithms volume 2: eigensystems*, Vol. 2, Siam.

M. Stojnic (2010), ℓ_1 optimization and its various thresholds in compressed sensing, in *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 3910–3913.

G. Strang (2019), *Linear algebra and learning from data*, Wellesley-Cambridge Press.

V. Strassen (1969), 'Gaussian elimination is not optimal', *Numerische mathematik* **13**(4), 354–356.

T. Strohmer and R. Vershynin (2009), 'A randomized Kaczmarz algorithm with exponential convergence', *J. Fourier Anal. Appl.* **15**(2), 262–278.

Y. Sun, Y. Guo, J. A. Tropp and M. Udell (2018), Tensor random projection for low memory dimension reduction, in *NeurIPS Workshop on Relational Representation Learning*.

Z. Szabo and B. Sriperumbudur (2019), On kernel derivative approximation with random fourier features, in *Proceedings of Machine Learning Research* (K. Chaudhuri and M. Sugiyama, eds), Vol. 89 of *Proceedings of Machine Learning Research*, PMLR, pp. 827–836.

S.-H. Teng (2010), The Laplacian paradigm: emerging algorithms for massive graphs, in *Theory and applications of models of computation*, Vol. 6108 of *Lecture Notes in Comput. Sci.*, Springer, Berlin, pp. 2–14.

C. Thrampoulidis and B. Hassibi (2015), 'Isotropically random orthogonal matrices: Performance of lasso and minimum conic singular values'.

C. Thrampoulidis, S. Oymak and B. Hassibi (2014), 'The gaussian min-max theorem in the presence of convexity'.

C. Thurau, K. Kersting and C. Bauckhage (2012), Deterministic cur for improved large-scale data analysis: An empirical study, in *Proceedings of the 2012 SIAM International Conference on Data Mining*, SIAM, pp. 684–695.

N. Tomczak-Jaegermann (1974), 'The moduli of smoothness and convexity and the Rademacher averages of trace classes $S_p(1 \leq p < \infty)$', *Studia Math.* **50**, 163–182.

J.-F. Ton, S. Flaxman, D. Sejdinovic and S. Bhatt (2018), 'Spatial mapping with gaussian processes and nonstationary fourier features', *Spatial Statistics* **28**, 59 – 78. One world, one health.

L. N. Trefethen and D. Bau III (1997), *Numerical linear algebra*, Vol. 50, Siam.

T. Trogdon (2017), 'On spectral and numerical properties of random butterfly matrices', *Appl. Math. Lett.* **95**, 48–58.

J. A. Tropp (2011*a*), 'Freedman's inequality for matrix martingales', *Electron. Commun. Probab.* **16**, 262–270.

J. A. Tropp (2011*b*), 'Improved analysis of the subsampled randomized Hadamard transform', *Adv. Adapt. Data Anal.* **3**(1-2), 115–126.

J. A. Tropp (2012*a*), 'A comparison principle for functions of a uniformly random subspace', *Probab. Theory Related Fields* **153**(3-4), 759–769.

J. A. Tropp (2012*b*), 'User-friendly tail bounds for sums of random matrices', *Found. Comput. Math.* **12**(4), 389–434.

J. A. Tropp (2015), 'An introduction to matrix concentration inequalities', *Foundations and Trends in Machine Learning* **8**(1-2), 1–230.

J. A. Tropp (2016), The expected norm of a sum of independent random matrices: an elementary approach, in *High dimensional probability VII*, Vol. 71 of *Progr. Probab.*, Springer, [Cham], pp. 173–202.

J. A. Tropp (2018), Analysis of randomized block krylov methods, Under revision.

J. A. Tropp (2019), Matrix concentration and computational linear algebra, CMS Lecture Notes 2019-01, Caltech, Pasadena, CA.

J. A. Tropp, M. B. Wakin, M. F. Duarte, D. Baron and R. G. Baraniuk (2006), Random filters for compressive sampling and reconstruction, in *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, Vol. 3, pp. III–III.

J. A. Tropp, A. Yurtsever, M. Udell and V. Cevher (2017*a*), Fixed-rank approximation of a positive-semidefinite matrix from streaming data, in *Advances in Neural Information Processing Systems 30* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan and R. Garnett, eds), Curran Associates, Inc., pp. 1225–1234.

J. A. Tropp, A. Yurtsever, M. Udell and V. Cevher (2017*b*), 'Practical sketching algorithms for low-rank matrix approximation', *SIAM Journal on Matrix Analysis and Applications* **38**(4), 1454–1485.

J. A. Tropp, A. Yurtsever, M. Udell and V. Cevher (2017*c*), 'Practical sketching algorithms for low-rank matrix approximation', *SIAM J. Matrix Anal. Appl.* **38**(4), 1454–1485.

J. A. Tropp, A. Yurtsever, M. Udell and V. Cevher (2019), 'Streaming low-rank matrix approximation with an application to scientific simulation', *SIAM J. Sci. Comput.* **41**(4), A2430–A2463.

S. Ubaru, J. Chen and Y. Saad (2017), 'Fast estimation of $\mathtt{tr}(f(A))$ via stochastic Lanczos quadrature', *SIAM J. Matrix Anal. Appl.* **38**(4), 1075–1099.

E. Ullah, P. Mianjy, T. V. Marinov and R. Arora (2018), Streaming kernel pca with $\tilde{\lambda}(\sqrt{n})$ random features, in *Advances in Neural Information Processing Systems 31* (S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi and R. Garnett, eds), Curran Associates, Inc., pp. 7311–7321.

J. Upadhyay (2016), 'Fast and space-optimal low-rank factorization in the streaming model with application in differential privacy', *arXiv preprint arXiv:1604.01429*.

Y. Urano (2013), A fast randomized algorithm for linear least-squares regression via sparse transforms, Masters thesis, New York University.

R. Van Handel (2016), Probability in high dimension, Apc 550 lecture notes, Princeton Univ.

R. Vershynin (2018), *High-dimensional probability*, Vol. 47 of *Cambridge Series in Statistical and Probabilistic Mathematics*, Cambridge University Press, Cambridge. An introduction with applications in data science, With a foreword by Sara van de Geer.

R. Vershynin (2019), 'Concentration inequalities for random tensors'.

S. Voronin and P.-G. Martinsson (2017), 'Efficient algorithms for cur and interpolative matrix decompositions', *Advances in Computational Mathematics* **43**(3), 495–516.

S. F. D. Waldron (2018), *An introduction to finite tight frames*, Applied and Numerical Harmonic Analysis, Birkhäuser/Springer, New York.

S. Wang (2019), 'Simple and almost assumption-free out-of-sample bound for random feature mapping'.

S. Wang, A. Gittens and M. W. Mahoney (2019), 'Scalable kernel k-means clustering with nyström approximation: Relative-error bounds', *J. Mach. Learn. Res.* **20**(1), 431–479.

R. C. Whaley and J. J. Dongarra (1998), Automatically tuned linear algebra software, in *Proceedings of the 1998 ACM/IEEE Conference on Supercomputing*, SC '98, IEEE Computer Society, Washington, DC, USA, pp. 1–27.

C. K. I. Williams and M. Seeger (2001), Using the nyström method to speed up kernel machines, in *Advances in Neural Information Processing Systems 13* (T. K. Leen, T. G. Dietterich and V. Tresp, eds), MIT Press, pp. 682–688.

D. P. Woodruff (2014), 'Sketching as a tool for numerical linear algebra', *Foundations and Trends in Theoretical Computer Science* **10**(1-2), 1 – 157.

F. Woolfe, E. Liberty, V. Rokhlin and M. Tygert (2008), 'A fast randomized algorithm for the approximation of matrices', *Appl. Comput. Harmon. Anal.* **25**(3), 335–366.

W. K. Wootters and B. D. Fields (1989), 'Optimal state-determination by mutually unbiased measurements', *Annals of Physics* **191**(2), 363 – 381.

J. Xia (2013), 'Randomized sparse direct solvers', *SIAM Journal on Matrix Analysis and Applications* **34**(1), 197–227.

J. Xia, S. Chandrasekaran, M. Gu and X. S. Li (2010), 'Fast algorithms for hierarchically semiseparable matrices', *Numerical Linear Algebra with Applications* **17**(6), 953–976.

J. Xiao, M. Gu and J. Langou (2017), Fast parallel randomized qr with column pivoting algorithms for reliable low-rank matrix approximations, in *2017 IEEE 24th International Conference on High Performance Computing (HiPC)*, IEEE, pp. 233–242.

C. D. Yu, J. Levitt, S. Reiz and G. Biros (2017*a*), Geometry-oblivious fmm for compressing dense spd matrices, in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ACM, p. 53.

W. Yu, Y. Gu, J. Li, S. Liu and Y. Li (2017*b*), 'Single-pass pca of large high-dimensional data'.

Q. Yuan, M. Gu and B. Li (2018), Superlinear convergence of randomized block lanczos algorithm, in *2018 IEEE International Conference on Data Mining (ICDM)*, pp. 1404–1409.

F. Zhang, ed. (2005), *The Schur complement and its applications*, Vol. 4 of *Numerical Methods and Algorithms*, Springer-Verlag, New York.

A. Zouzias (2013), Randomized primitives for linear algebra and applications, Phd thesis, University of Toronto, Toronto.