RANDOMIZED MATRIX COMPUTATIONS: THEMES AND VARIATIONS

ANASTASIA KIREEVA AND JOEL A. TROPP

ABSTRACT. This short course offers a new perspective on randomized algorithms for matrix computations. It explores the distinct ways in which probability can be used to design algorithms for numerical linear algebra. Each design template is illustrated by its application to several computational problems. This treatment establishes conceptual foundations for randomized numerical linear algebra, and it forges links between algorithms that may initially seem unrelated.

CONTENTS

1. MOTIVATION

Numerical analysis and probability theory have not always been on the best interpersonal terms. Although Monte Carlo methods date back to the earliest days of numerical computation [Met87], researchers have often been skeptical about their performance. For example, Monte Carlo algorithms for approximating integrals [QSS07, Sect. 9.9.3] can only achieve low precision and their output is highly variable. This fact about this particular procedure inspired a general sentiment that randomized algorithms are unreliable tools for numerical problems:

> "Our experience suggests that many practitioners of scientific computing view randomized algorithms as a desperate and final resort."
>
> —Halko et al. [HMT11, Rem. 1.1].

Over the last 20 years, the numerical analysis community has come to appreciate the value of probabilistic algorithms. This sea change can be attributed to the development and popularization of randomized methods that can solve large-scale problems efficiently, reliably, and robustly. In particular, the randomized SVD [HMT11; TW23] has become a workhorse algorithm for obtaining low-rank matrix approximations in scientific computing and machine learning. By now, the numerical analysis literature contains a diverse collection of randomized algorithms.

This short course offers a new perspective on *randomized algorithms for matrix computations*. Our goal has been to identify distinct conceptual ways that probability can be used for algorithm design in numerical linear algebra. We refer to these design templates as *themes*. For each theme, we describe how it allows us to solve a number of basic computational problems. We refer to these examples as *variations*. It is our hope that this treatment highlights connections between approaches that may seem different in spirit.

1.1. **The role of randomness.** Common problems in numerical linear algebra include linear systems, least-squares problems, eigenvalue problems, matrix approximation, and more. Before we discuss computational approaches to these problems, we would like to clarify what we mean by a *randomized algorithm* and how it differs from *statistical randomness*.

1.1.1. *Algorithmic randomness.* Recall that a *computer algorithm* is a finite sequence of well-defined elementary steps that provably returns a solution to a well-specified computational problem. In numerical analysis, we allow approximate solutions that achieve a specified error tolerance. A *randomized algorithm* solves each fixed problem instance by generating random variables and using them as part of the computational procedure.

Example 1.1 (Checking matrix multiplication)**.** There is a simple randomized algorithm for testing whether we have correctly multiplied two matrices. Given square $n \times n$ matrices (A, B, C) with integer entries, we would like to confirm whether $C = AB$ without performing a matrix multiplication. An inexpensive algorithm is to draw a random vector $x \sim \text{UNIFORM}\{\pm 1\}^n$ and form the matrix–vector products Cx and $A(Bx)$. If the two vectors are unequal, then we can confidently report **false**. If the two vectors are equal, then we report **true** even though it is possible that our judgment is wrong.

Since this procedure can fail with nonzero probability, it is called a *Monte Carlo-style* randomized algorithm. Most of the algorithms we discuss are Monte Carlo-style methods where the outcome is random. Fortunately, it is often inexpensive to check whether they have returned the correct answer.

Example 1.2 (Maximum eigenvalue)**.** The randomized power method is a familiar algorithm for approximating the maximum eigenvalue of a positive-semidefinite matrix A to a specified relative error. It begins with a random vector $x \sim \text{NORMAL}(0, I)$ and repeatedly applies the matrix to the vector to

obtain a sequence of maximum eigenvalue estimates:

$$\xi_t = \boldsymbol{x}^*(\boldsymbol{A}^t\boldsymbol{x})/\boldsymbol{x}^*\boldsymbol{x} \quad \text{for } t = 0, 1, 2, 3, \ldots.$$

After a sufficient number of steps, we hope that the estimate ξ_t approximates the maximum eigenvalue within a specified relative error tolerance. We will see that this approach succeeds with probability one. See Section 3.1 for more details about the randomized power method.

Although this method eventually succeeds, its *runtime* is a random variable. A procedure with a random runtime is called a *Las Vegas-style* randomized algorithm. We will encounter several more algorithms of this type.

1.1.2. *Statistical randomness.* In sharp contrast, *statistical randomness* is the idea of using probability to model a computational problem or to describe the behavior of an algorithm. Aside from a short overview here, this course does not consider the role of statistical randomness in numerical analysis.

Statisticians often model observations as independent samples from an underlying population. This perspective leads to the concept of *average-case analysis*, where we ask how an algorithm performs for a problem instance drawn at random from some distribution. For examples in numerical linear algebra, see Demmel's paper [Dem88].

In many scientific fields, data collection involves both measurement errors and uncertainty. To address these issues, numerical analysts study the *sensitivity* of mathematical problem formulations to perturbations of the data. They also study the *stability* of algorithms under changes in the input. Historically, these analyses dealt with the worst choice of perturbation of a given magnitude. See [Hig02] for discussion.

As another application of statistical randomness to numerical analysis, we can model *perturbations of problem data* using probability. We can try to understand how much the answer to a computational problem is likely to change under a random perturbation. We can also try to assess how a random perturbation of the problem data impacts the performance of a particular algorithm, which is called a *smoothed analysis* of the algorithm [ST02]. We will see that smoothed analysis has computational implications for linear algebra (Section 6.3).

Numerical analysts have also worked hard to understand the effects of *rounding errors* on the output of numerical algorithms. The worst-case analysis is quite pessimistic, which motivates us to consider statistical models for rounding errors. This idea was already considered in 1951 by Goldstine & von Neumann [GN51] in their work on solving linear systems. Higham and coauthors have recently revisited the probabilistic analysis of rounding errors [HM19; CH23b]. Computationally, we can also exploit the beneficial effects of random rounding errors by using stochastic rounding procedures [CHM21].

Analysis based on statistical randomness has been a useful paradigm for understanding numerical computation. Nevertheless, statistical randomness imposes the *modeling assumption* that the problem data, perturbations to the problem data, or errors made during computations can be treated as random quantities. By contrast, algorithmic randomness is injected into the computational procedure in a controlled manner by the algorithm designer, allowing for numerical methods that are provably effective for arbitrary inputs.

1.1.3. *Statistical behavior of randomized algorithms.* Keep in mind that randomized algorithms for matrix computations produce statistical output. Therefore, we can use tools from statistical theory to understand the distribution of the output or even to develop more effective algorithms.

Example 1.3 (Majority vote). Consider a randomized algorithm for a decision problem that gives the correct result with probability $p > 0.5$. Although we may not have a lot of confidence in the output from a single run of the algorithm, we can run the algorithm n times and take a majority vote. (Were the trials more often **true** or **false**?) To obtain failure probability $\delta \in (0, 1)$, it is sufficient to

perform $n \geq (p - 0.5)^{-2} \log(1/\delta)$ independent repetitions. This approach can dramatically increase the reliability of the basic algorithm.

Statistical theory can also be used to equip the output of a randomized algorithm with a confidence interval; see Section 2.1.6 for an application to the problem of trace estimation. We can also use insights from the design of statistical estimators to develop better algorithms, e.g., whose output is less variable; see Section 2.1.7 for some discussion.

We will not emphasize the nexus between statistics and computation, but this link offers potential opportunities for supercharging the methods that we discuss.

1.2. **Prerequisites.** This course is intended for advanced undergraduates, beginning graduate students, and curious researchers in the mathematical sciences. On the mathematical side, we assume a thorough knowledge of linear algebra, matrix analysis, applied probability, and basic statistics. On the computational side, you should understand the basic principles of computer algorithms, and we expect that you have taken a first course on numerical linear algebra. If you have followed everything in the introduction so far, you're probably in good shape.

1.3. **Other resources.** As a classical reference on matrix computations, the book of Golub & Van Loan [GVL13] is a comprehensive and trusted source.

The second author has written several surveys and sets of lecture notes on aspects of randomized matrix computations. These resources are available from the website `https://tropp.caltech.edu/`. The current notes borrow heavily from this body of work.

- **Finding structure with randomness [HMT11].** This survey provides a framework for randomized SVD algorithms for low-rank matrix approximation. The paper includes pseudocode for reliable algorithms, a number of applications in scientific computing and machine learning, and the first theoretical analysis that can predict the detailed behavior of these algorithms.
- **An introduction to matrix concentration inequalities [Tro15].** This monograph provides a comprehensive introduction to matrix concentration inequalities, which are basic tools for studying random matrices. It explains how these results can be used to analyze probabilistic methods for matrix computation, including matrix Monte Carlo methods and randomized dimension reduction schemes. These notes support a tutorial given at NeurIPS 2012.
- **Matrix concentration and computational linear algebra [Tro19].** These notes accompany a short course taught at École Normale Supérieure in Paris in July 2019. They present some foundational results on matrix concentration for independent sums and martingales, and they include detailed applications to quantum state tomography, sparsification of graphs, and the analysis of the near-linear-time graph Laplacian solver of Kyng & Sachdeva [KS16].
- **Randomized algorithms for matrix computations [Tro20].** These notes were written to support a quarter class on randomized matrix computations taught at Caltech in the Winter 2020 term. The material includes trace estimation, eigenvalue computations, low-rank matrix approximation, matrix concentration, matrix Monte Carlo methods, randomized dimension reduction, and more.
- **Randomized numerical linear algebra [MT20].** This survey was invited to the 2020 volume of *Acta Numerica*. It covers a significant proportion of the literature on randomized matrix computations, with a focus on methods that are reliable in practice.
- **Randomized algorithms for low-rank matrix approximation [TW23].** This survey updates the randomized SVD paper [HMT11] to include more modern approaches based on Krylov subspace methods. It also treats Nyström approximations of psd matrices. In addition, the paper showcases scientific applications in genetics and in molecular biophysics.

There are several other treatises on randomized matrix computations. For a theoretical computer science perspective, see the notes [Woo14; KV17a; DM18]. The monograph [Mur+23] discusses the RandBLAS primitives and the RandLAPack library that are currently under development.

1.4. **Notation.** We employ standard notation that, hopefully, is self-explanatory. In most places, the initial appearance of a symbol is accompanied by a reminder about what it means. If you want to gird yourself first, this section will acquaint you with the main pieces of notation.

We work primarily in the real field $\mathbb{R}$, although large parts of the discussion extend to matrices with entries in the complex field $\mathbb{C}$. All logarithms have the natural base e. The Pascal notation $:=$ or $=:$ generates a definition.

We use lowercase italic letters (a, b, c) for scalar variables. Lowercase bold italic letters $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})$ refer to vectors, which are always column vectors. Uppercase bold italic letters $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$ denote matrix variables. The symbol $\mathbf{e}_j$ is the jth standard basis vector. The letter $\mathbf{I}$ or $\mathbf{I}_n$ represents a square identity matrix; the subscript specifies the dimension. The symbol * is the (conjugate) transpose of a vector or matrix. The dagger † refers to the Moore–Penrose pseudoinverse of a matrix.

We equip the Euclidean space $\mathbb{R}^n$ with the standard inner product $\langle \cdot, \ \cdot \rangle$ and the norm $\|\cdot\|_2$. There are several ways to express the coordinates of a vector or a matrix. To denote the (j, k) entry of a matrix $\boldsymbol{A}$, we may write a_{jk} or use the functional notation $\boldsymbol{A}(j, k)$. The symbol $\boldsymbol{A}(i, :)$ refers to the ith row of the matrix, while $\boldsymbol{A}(:, j)$ refers to the jth row of the matrix. As a warning, $\boldsymbol{a}_k$ may refer to either the kth row or the kth column of the matrix, depending on context. In some settings, we must extract components with respect to bases other than the standard basis.

The linear space $\mathbb{M}_n(\mathbb{R})$ comprises all square $n \times n$ matrices. The real-linear subspace $\mathbb{H}_n(\mathbb{R})$ comprises the real, symmetric matrices with dimension n. Each of these matrices has n real eigenvalues, arranged in decreasing order: $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$. The cone $\mathbb{H}_n^+(\mathbb{R})$ contains the positive-semidefinite (psd) matrices, those symmetric matrices whose eigenvalues are all nonnegative. For a pair $(\boldsymbol{A}, \boldsymbol{B})$ of symmetric matrices, the semidefinite relation $\boldsymbol{A} \preccurlyeq \boldsymbol{B}$ means that $\boldsymbol{B} - \boldsymbol{A}$ is a psd matrix.

For a general matrix $\boldsymbol{B} \in \mathbb{R}^{m \times n}$, we list the singular values in decreasing order:

$$\sigma_{\max} := \sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n =: \sigma_{\min}.$$

The spectral condition number is the matrix function $\kappa := \sigma_{\max}/\sigma_{\min}$. We frequently use the Frobenius norm $\|\cdot\|_F$, the spectral norm $\|\cdot\|$, and the trace norm $\|\cdot\|_*$.

Everything takes place in an (unspecified) probability space that is big enough to support all of the random variables we consider. We often use letters at the end of the alphabet to refer to random variables, while maintaining our other conventions. For instance, X is a scalar random variable, $\boldsymbol{x}$ is a vector-valued random variable, and $\boldsymbol{X}$ is a random matrix. The symbol $\sim$ means "has the distribution". Small capitals are reserved for named distributions (NORMAL, UNIFORM). The abbreviation "i.i.d." means independent and identically distributed. Random variables with hats, such as $\widehat{\boldsymbol{x}}$, generally refer to estimators.

The symbol $\mathbb{P}$ reports the probability of an event. The operator $\mathbb{E}$ returns the expectation of a random variable, computed entrywise for vectors or matrices. We use subscripts to denote partial expectations, so $\mathbb{E}_X$ is the expectation with respect to the randomness in X. The partial expectation is only used when there is no possibility for confusion (e.g., X is independent from everything else).

We use the computer science interpretation of the $\mathcal{O}$ symbol. For example, $\mathcal{O}(n)$ is the class of functions that grow no more quickly than n as $n \to \infty$.

2. Monte Carlo approximation

Already in the earliest days of numerical computation, scientists were exploring randomized algorithms for approximating complicated physical models [Met87]. Enrico Fermi is credited with using statistical sampling methods to simulate neutron diffusion in the 1930s. Instead of tracking the

full dynamical process, he randomly constructed the next segment of an individual neutron trajectory. The computations were often performed with a mechanical adding machine. Independently, in 1946 and 1947, Stanislaw Ulam and John von Neumann proposed a similar model for neutron diffusion as a trial problem for the newly constructed ENIAC computer. In late 1947, Nick Metropolis and Klari von Neumann designed an implementation, and they performed the first tests of the Monte Carlo method on the ENIAC.

Today, every student of numerical analysis learns about Monte Carlo methods as a technique for approximating a complicated integral, especially of a high-dimensional function [QSS07, Sect. 9.9.3]. In this application, we draw a random variable that has the same distribution as the integrand. To reduce the variance of this simple estimator, we can draw independent samples and average them together.

Although Monte Carlo methods are an ancient part of numerical analysis, their advent in numerical linear algebra is more recent.

> **Theme** (Monte Carlo approximation). Construct a simple unbiased estimator of a complicated linear-algebraic quantity. Reduce the variance by averaging independent copies of the estimator.

This section describes how Monte Carlo approximation can be applied to two problems in matrix computations. In Section 2.1, we consider how to estimate a specific scalar quantity, the trace of a square matrix. In Section 2.2, we explain how related ideas allow us to estimate a matrix quantity, such as the product of two matrices. We also provide citations to papers that formulate other applications of Monte Carlo approximation in numerical linear algebra.

2.1. Trace estimation.
Let us begin with an approachable example of scalar Monte Carlo approximation in linear algebra. The goal is to estimate the trace of a square matrix using as few matrix–vector products as possible. We will introduce a randomized algorithm for this task and describe some elements of the analysis. This approach, due to Girard [Gir89], stands among the oldest examples of probability in matrix computations. Our treatment follows [MT20; Tro20].

2.1.1. *Implicit trace estimation.*
Suppose that we are given a square matrix $A \in \mathbb{M}_n(\mathbb{R})$. Our goal is to compute the trace:

$$\mathrm{tr}(A) \coloneqq \sum\nolimits_{i=1}^{n} a_{ii}.$$

Of course, the problem is trivial if we can access the diagonal entries of A at low computational cost. In this case, revealing and summing the diagonal entries takes about $\mathcal{O}(n)$ arithmetic operations.

Instead, assume that we have access to the matrix via matrix–vector products (*matvecs*). That is, given a vector $x \in \mathbb{R}^n$, we can form the product $x \mapsto Ax$ at a reasonable cost. We will justify this model in Section 2.1.2. For now, our task is to determine $\mathrm{tr}(A)$ using as few matvecs as possible.

As before, we can complete this job by using matvecs to read off the diagonal entries of the matrix:

$$a_{ii} = \mathbf{e}_i^*(A\mathbf{e}_i) \quad \text{for each } i = 1, 2, 3, \ldots, n.$$

The symbol $\mathbf{e}_i$ denotes the ith standard basis vector in $\mathbb{R}^n$. This procedure provides an exact value for the trace, but it requires fully n matvec operations. The cost may be prohibitive when n is large.

Fortunately, in many settings, it is acceptable to produce an *approximation* for the trace if we can reduce the number of matvecs substantially.

> **Problem** (Implicit trace estimation). Suppose that A is a square matrix, accessible via matvecs: $x \mapsto Ax$. The task is to approximate $\mathrm{tr}(A)$ while limiting the number of matvecs.

2.1.2. *Applications of trace estimation.* Although the implicit trace estimation problem may seem unnatural, it arises in a wide range of applications. These domains include statistical computation, network science, chemical graph theory, and quantum statistical mechanics.

- **Smoothing splines [Gir89; Hut90].** When fitting a smoothing spline to scattered data, we need to estimate $\mathrm{tr}(A^{-1})$, the trace of a matrix inverse, to perform generalized cross-validation for the smoothing parameter. In this context, the matrix A is a tridiagonal matrix that models a second-difference operator. We can apply the inverse to a vector in $\mathcal{O}(n)$ arithmetic operations using a Cholesky factorization and triangular substitution.
- **Social network analysis [AHD18].** We can count triangles in a graph to measure the centrality of nodes in a social network. This task reduces to computing $\mathrm{tr}(A^3)$, where A is the adjacency matrix of the graph. We can apply the third power A^3 to a vector by repeated multiplication.
- **Chemical graph theory [Est22].** The folding degree of long-chain proteins can be measured using the *Estrada index*, which is defined as $\mathrm{tr}(\mathrm{e}^A)$. In this setting, A is a tridiagonal matrix whose diagonal is related to the dihedral angles between planes in the folded protein.
- **Quantum statistical mechanics [CH23a; ETW24].** We can use trace estimation to compute the partition function $\mathrm{tr}(\mathrm{e}^{-\beta H})$ and the (unnormalized) average energy $\mathrm{tr}(H\mathrm{e}^{-\beta H})$ of a quantum mechanical system described by a Hamiltonian matrix H. We can approximate these exponentials efficiently using low-degree polynomials or Krylov subspace methods.

See the survey [US18] for further discussion and examples.

2.1.3. *The role of randomness.* In 1989, Girard [Gir89] proposed an elegant Monte Carlo approximation algorithm for the implicit trace estimation problem. Hutchinson [Hut90] described a variant of Girard's method that has become popular. Both of these authors were concerned with cross-validation for smoothing splines. Here is the idea.

Suppose that $x \in \mathbb{R}^n$ is a random test vector whose distribution is *isotropic*:

$$\mathbb{E}[xx^*] = \mathbf{I}_n.$$

Many familiar distributions are isotropic:

- **Standard normal.** A random vector $x \sim \mathrm{NORMAL}(0, \mathbf{I}_n)$ with independent standard normal entries is isotropic.
- **Uniform on a sphere.** A random vector $x \sim \mathrm{UNIFORM}(\sqrt{n}\mathbb{S}^{n-1})$ distributed uniformly on the Euclidean sphere with radius $\sqrt{n}$ is isotropic.
- **Independent signs.** A random vector $x \sim \mathrm{UNIFORM}\{\pm 1\}^n$ whose entries are independent and take values ± 1 with equal probability is isotropic.

These distributions all have a lot of randomness, which helps us build estimators with low variance.

Let $A \in \mathbb{M}_n(\mathbb{R})$ be a square input matrix. Taking one matvec with the matrix A, we can form the scalar random variable

$$Y := x^*(Ax).$$

Observe that Y is an *unbiased estimator* for the trace of the matrix:

$$\mathbb{E}[Y] = \mathbb{E}[\mathrm{tr}(x^*Ax)] = \mathbb{E}[\mathrm{tr}(Axx^*)] = \mathrm{tr}(A \cdot \mathbb{E}[xx^*]) = \mathrm{tr}(A).$$

We have used the cyclic property of the trace, the linearity of the trace and the expectation, and the assumption that the test vector x has an isotropic distribution.

Although the simple trace estimator Y is correct on average, its fluctuations may be large in comparison with $\mathrm{tr}(A)$. A standard remedy is to *average independent copies* of the simple trace estimator. Taking s matvecs with the matrix A, we can form the Monte Carlo trace estimator:

$$\widehat{\mathrm{tr}}_s := \frac{1}{s}\sum_{i=1}^{s} Y_i \quad \text{where } Y_i \sim Y \text{ i.i.d.}$$

Algorithm 1. *Trace estimation: Simplest Monte Carlo method.*

Input: Input matrix $A \in \mathbb{M}_n(\mathbb{R})$, number s of samples
Output: Trace estimate $\widehat{\mathrm{tr}}_s$, an estimate $\widehat{v}_s$ for the variance of $\widehat{\mathrm{tr}}_s$

---

1 **function** TraceEstimate(A, s)
2   **for** $i = 1, \ldots, s$ **do**
3   Sample isotropic random vector $x_i \in \mathbb{R}^n$ ▷ For example, $x_i \sim \text{UNIFORM}\{\pm 1\}^n$
4          $Y_i = x_i^*(A x_i)$ ▷ One matvec
5      $\widehat{\mathrm{tr}}_s = s^{-1} \sum_{i=1}^s Y_i$ ▷ Trace estimator
6      $\widehat{v}_s = (s(s-1))^{-1} \sum_{i=1}^s (Y_i - \widehat{\mathrm{tr}}_s)^2$ ▷ Variance estimate for trace estimator

---

By linearity, the trace estimator $\widehat{\mathrm{tr}}_s$ remains unbiased. By independence of the summands, we have reduced the variance by a factor of s over the simple estimator. Altogether,

$$\mathbb{E}[\widehat{\mathrm{tr}}_s] = \mathrm{tr}(A) \quad \text{and} \quad \mathrm{Var}[\widehat{\mathrm{tr}}_s] = \frac{1}{s}\mathrm{Var}[Y]. \tag{2.1}$$

Of course, we still need to control $\mathrm{Var}[Y]$, the variance of the simple estimator, to confirm that the trace estimator $\widehat{\mathrm{tr}}_s$ is precise enough to employ in practice. Algorithm 1 presents pseudocode for the basic method.

2.1.4. *Trace estimation: A priori bounds.* It remains to assess how many samples s are sufficient for the Monte Carlo estimator $\widehat{\mathrm{tr}}_s$ to approximate $\mathrm{tr}(A)$ within a specified tolerance. This analysis hinges on the variance of the simple trace estimator.

For ease of exposition, assume that the input matrix $A \in \mathbb{H}_n^+(\mathbb{R})$ is a real symmetric *positive-semidefinite (psd)* matrix that is not zero. We also assume that the test vector follows the *independent sign distribution*: $x \sim \text{UNIFORM}\{\pm 1\}^n$.

With these assumptions, the simple trace estimator $Y = x^*(A x)$ remains unbiased: $\mathbb{E}[Y] = \mathrm{tr}(A)$. We can also obtain an exact expression for the variance:

$$\mathrm{Var}[Y] = 2 \sum_{i \neq j} a_{ij}^2 \leq 2\|A\|_F^2.$$

Here and elsewhere, $\|\cdot\|_F$ is the Frobenius norm. Since A is psd, its eigenvalues $\lambda_1, \ldots, \lambda_n$ are nonnegative. Therefore, we have the upper bound

$$\mathrm{Var}[Y] = 2\|A\|_F^2 = 2 \sum_{i=1}^n \lambda_i^2 \leq 2\big(\max_{j=1,\ldots,n} \lambda_j\big) \sum_{i=1}^n \lambda_i = 2\|A\|\,\mathrm{tr}(A).$$

As usual, $\|\cdot\|$ is the spectral norm. This inequality compares the variance of the estimator with the trace that we are trying to estimate.

The variance is a quadratic functional, so we would like to place it on the scale of the *squared* trace. This step yields the bound

$$\frac{\mathrm{Var}[Y]}{\mathrm{tr}(A)^2} \leq \frac{2\|A\|}{\mathrm{tr}(A)}.$$

To interpret this result, let us make a definition.

Definition 2.1 (Intrinsic dimension)**.** For a nonzero psd matrix $A \in \mathbb{H}_n^+(\mathbb{R})$, the *intrinsic dimension* is the quantity

$$\mathrm{int\,dim}(A) := \frac{\mathrm{tr}(A)}{\|A\|}.$$

We define $\mathrm{int\,dim}(0) := 0$.

If $A \in \mathbb{H}_n^+(\mathbb{R})$ is nonzero, then $1 \leq \operatorname{int dim}(A) \leq n$. The lower bound is attained if and only if A has rank one. The upper bound is attained if and only if A is a scalar matrix: $A = \alpha I_n$ for $\alpha > 0$. More generally, we think about the intrinsic dimension as a continuous proxy for the rank of the matrix.

It follows that the variance of the simple trace estimator Y is inversely proportional to the intrinsic dimension of the target matrix:

$$\frac{\operatorname{Var}[Y]}{\operatorname{tr}(A)^2} \leq \frac{2}{\operatorname{int dim}(A)}.$$

As a consequence of (2.1), the Monte Carlo estimator $\widehat{\operatorname{tr}}_s$ with s independent sign vectors satisfies the variance bound

$$\frac{\operatorname{Var}[\widehat{\operatorname{tr}}_s]}{\operatorname{tr}(A)^2} \leq \frac{2}{s \cdot \operatorname{int dim}(A)}. \tag{2.2}$$

Let us emphasize that this calculation depends on the distribution of the test vector x. We arrive at the following statement.

Proposition 2.2 (Monte Carlo trace estimation). *Assume that $A \in \mathbb{H}_n^+(\mathbb{R})$ is a nonzero psd matrix. Form the Monte Carlo trace estimator $\widehat{\operatorname{tr}}_s$ with s independent sign vectors:*

$$\widehat{\operatorname{tr}}_s := \frac{1}{s} \sum_{i=1}^s x_i^*(A x_i) \quad \text{where } x_i \sim \text{UNIFORM}\{\pm 1\}^n \text{ i.i.d.}$$

For any tolerance $\varepsilon > 0$, the estimator satisfies the probability bound

$$\mathbb{P}\left\{|\widehat{\operatorname{tr}}_s - \operatorname{tr}(A)| \geq \varepsilon \operatorname{tr}(A)\right\} \leq \frac{2}{\varepsilon^2 s \cdot \operatorname{int dim}(A)}. \tag{2.3}$$

In particular, the probability bound is nontrivial as soon as the number s of samples satisfies

$$s \geq 2\varepsilon^{-2}(\operatorname{int dim}(A))^{-1}. \tag{2.4}$$

Proof. This result follows from an easy application of Chebyshev's inequality:

$$\mathbb{P}\left\{|\widehat{\operatorname{tr}}_s - \operatorname{tr}(A)| \geq \varepsilon \operatorname{tr}(A)\right\} \leq \frac{\mathbb{E}|\widehat{\operatorname{tr}}_s - \operatorname{tr}(A)|^2}{\varepsilon^2 \cdot \operatorname{tr}(A)^2} = \frac{\operatorname{Var}[\widehat{\operatorname{tr}}_s]}{\varepsilon^2 \cdot \operatorname{tr}(A)^2} \leq \frac{2}{\varepsilon^2 s \cdot \operatorname{int dim}(A)}.$$

The equality holds because $\mathbb{E}[\widehat{\operatorname{tr}}_s] = \operatorname{tr}(A)$, and the last inequality is (2.2). $\qquad\square$

2.1.5. *Discussion.* Let us unpack the statement of Proposition 2.2. A surprising feature of the bound (2.4) is that we can reliably estimate the trace of a psd matrix within a factor of two using just a $constant$ number of samples, say, $s = 16$. This holds true regardless of the dimension of the matrix! For example, if we can perform a matvec with $\mathcal{O}(n)$ arithmetic operations, then the cost of coarse trace estimation is also $\mathcal{O}(n)$ operations.

Second, observe that the quality of the trace estimator $increases$ with the intrinsic dimension, the number of energetic dimensions in the range of the matrix. Our intuition suggests that a random sample provides more information about a high-rank matrix because it is more likely to be aligned with an energetic direction. Thus, it is easier to approximate the trace of high-rank matrix.

Third, the expression (2.4) for the number s of samples scales with ε^{-2}. Thus, it may require an exorbitant number of samples to achieve a small relative error. For instance, $\varepsilon = 1/10$ already introduces a factor of 100 into the bound. The scaling is an inexorable consequence of the central limit theorem, rather than an artifact of the analysis. This is the curse of Monte Carlo estimation (but see Section 2.1.7!).

We can obtain probability bounds much stronger than (2.3) by using exponential concentration inequalities in place of Chebyshev's inequality; for example, see [AT11; GTP18]. While this analysis can tighten the bound on the failure probability, the basic scaling for the number s of samples in (2.4) remains the same.

Finally, let us mention that the best distribution for the test vector depends on the application. The independent sign distribution and the spherical distribution are both excellent candidates. The Gaussian distribution leads to trace estimates with higher variance than both the independent sign distribution and the spherical distribution, so it should usually be avoided. See [MT20, Sec. 4] for discussion of some alternative distributions.

2.1.6. *Trace estimation: A posteriori bounds.* We have obtained a prior theoretical bound (2.4) for the number of samples s that suffice to achieve relative error ε. Unfortunately, this bound depends on the intrinsic dimension, $\operatorname{int dim}(\boldsymbol{A})$, of the target matrix, which is usually not available to the user of an algorithm. Lacking this information, how can we determine the number s of samples we need?

Luckily, the Monte Carlo trace estimator is a statistical method, so we can bring statistical tools to bear. In particular, the sample variance $\widehat{v}_s$ provides an unbiased estimator for the variance of the trace estimator. Recall the definition:

$$\widehat{v}_s := \frac{1}{s(s-1)} \sum\nolimits_{i=1}^{s} (Y_i - \widehat{\operatorname{tr}}_s)^2.$$

If the distribution of the test vector $\boldsymbol{x}$ has four finite moments ($\mathbb{E}\,\|\boldsymbol{x}\|_2^4 < +\infty$), then we have the relation

$$\mathbb{E}[\widehat{v}_s] = \operatorname{Var}[\widehat{\operatorname{tr}}_s].$$

Therefore, we can employ the sample variance to construct a stopping rule for the trace estimator. For a psd input matrix $\boldsymbol{A}$, to attain a relative error tolerance $\varepsilon > 0$, we can continue drawing samples until

$$\widehat{v}_s \leq (\varepsilon \cdot \widehat{\operatorname{tr}}_s)^2.$$

We can also construct confidence intervals for the trace using classical techniques (e.g., Student's t confidence intervals) or simulation-based methods (e.g., the bootstrap). See the survey [MT20, Sec. 4] or the paper [ET24] for some discussion.

2.1.7. *Monte Carlo trace estimation: Improvements.* We can refine the simplest Monte Carlo methods by exploiting other insights from the theory of statistical estimation. Collectively, these ideas are called ==variance reduction== techniques [Liu04].

At present, the most accurate implicit trace estimators layer several variance reduction strategies [Mey+21; ETW24]. The main idea is to use random matvecs to build a low-rank approximation of the input matrix. The trace of this approximation serves as an initial estimate for the trace of the input matrix (called a "control variate"). We estimate the trace of the residual by forming additional random matvecs. A second key idea is to design an exchangeable estimator that employs the random matvecs in a symmetrical fashion.

2.1.8. *Scalar Monte Carlo: Further examples.* Several other core problems in numerical linear algebra reduce to trace estimation [MT20, Secs. 4, 5]. First, we can estimate the squared Frobenius norm of a rectangular matrix $\boldsymbol{B} \in \mathbb{R}^{m \times n}$ by applying a Monte Carlo trace estimator to the Gram matrix:

$$\|\boldsymbol{B}\|_{\mathrm{F}}^2 = \operatorname{tr}(\boldsymbol{B}^* \boldsymbol{B}).$$

Similarly, we can estimate the Schatten-4 norm $\|\boldsymbol{B}\|_{S_4}^4$ by calculating the sample variance of the trace estimator for $\boldsymbol{B}^* \boldsymbol{B}$ based on standard normal test vectors.

Estimating higher-order Schatten norms is significantly harder, but the literature [LNW14; KV17b] describes some elegant strategies based on Monte Carlo approximation. The paper [KV17b] extends these techniques to estimate the spectral density of a large matrix.

We can also use Monte Carlo methods to approximate the trace of a matrix function $\operatorname{tr}(f(\boldsymbol{A}))$ where $\boldsymbol{A}$ is a self-adjoint matrix and f is a spectral function. In particular, there is a remarkable technique called stochastic Lanczos quadrature [GM10; UCS17] that addresses this problem by combining the

Lanczos tridiagonalization method with stochastic trace esimation. See [MT20, Sec. 6] for an overview. The paper [CH23a] proposes a method that combines the virtues of stochastic Lanczos quadrature with the variance reduction techniques mentioned in Section 2.1.7.

2.2. **Matrix approximation by sampling.** Monte Carlo approximation is often used to estimate scalar quantities, such as the value of an integral or the trace of a matrix. The same strategy is valid in any linear space, including a space of matrices. In this section, we investigate how to use Monte Carlo methods to approximate a "complicated" matrix. This problem can arise in settings where we want to reduce the cost of computing the matrix exactly. It also allows us to construct a matrix approximation that has more regularity than the target matrix.

We begin with an abstract treatment of matrix Monte Carlo before turning to some more concrete examples, including approximate matrix multiplication and matrix sparsification. Our presentation follows [Tro15].

2.2.1. *Matrix approximation by sampling.* Consider a rectangular target matrix $\boldsymbol{B} \in \mathbb{R}^{m \times n}$. Suppose that the target matrix can be represented as a sum:

$$\boldsymbol{B} = \sum_{j=1}^{J} \boldsymbol{B}_j \quad \text{where each } \boldsymbol{B}_j \in \mathbb{R}^{m \times n}. \tag{2.5}$$

We can imagine that each summand $\boldsymbol{B}_j$ has a simpler structure than the target matrix $\boldsymbol{B}$. For example, we might assume that each $\boldsymbol{B}_j$ is sparse or that each $\boldsymbol{B}_j$ has low rank. Of course, we can decompose any matrix into a sum of sparse matrices or into a sum of low-rank matrices.

We can exploit the decomposition (2.5) to design a Monte Carlo estimator for the target matrix $\boldsymbol{B}$. The key idea is to sample one of the simple matrices $\boldsymbol{B}_j$ according to a carefully chosen probability distribution $(p_j : j = 1, \dots, J)$ on the indices. More precisely, we construct a random matrix $\boldsymbol{Y} \in \mathbb{R}^{m \times n}$ with the distribution

$$\mathbb{P}\left\{\boldsymbol{Y} = p_j^{-1}\boldsymbol{B}_j\right\} = p_j \quad \text{for each } j = 1, \dots, J. \tag{2.6}$$

It is easy to check that $\boldsymbol{Y}$ is an unbiased estimator for the target matrix $\boldsymbol{B}$. Indeed,

$$\mathbb{E}[\boldsymbol{Y}] = \sum_{j=1}^{J} \left(p_j^{-1}\boldsymbol{B}_j\right) \cdot p_j = \sum_{j=1}^{J} \boldsymbol{B}_j = \boldsymbol{B}.$$

Furthermore, the random matrix $\boldsymbol{Y}$ inherits properties shared by the summands, such as sparsity or low rank.

As with scalar Monte Carlo methods, the simple estimator $\boldsymbol{Y}$ can be a poor approximation of the target matrix $\boldsymbol{B}$. To address this concern, we apply the same Monte Carlo approximation strategy: sample independent copies of $\boldsymbol{Y}$, and average to reduce the variance. For a given number s of samples, the matrix Monte Carlo estimator takes the form

$$\widehat{\boldsymbol{B}}_s := \frac{1}{s} \sum_{i=1}^{s} \boldsymbol{Y}_i \quad \text{where } \boldsymbol{Y}_i \sim \boldsymbol{Y} \text{ i.i.d.} \tag{2.7}$$

When the number s of samples is small, we anticipate that the Monte Carlo approximation can be computed inexpensively. Moreover, the approximation may enjoy a simpler structure than the target matrix. For example, if each $\boldsymbol{B}_j$ has rank one, then the approximation $\widehat{\boldsymbol{B}}_s$ has rank s.

2.2.2. *Matrix sampling: A priori bounds.* We are interested in determining the number s of samples that suffice to achieve a precise approximation of the target matrix. For a tolerance $\varepsilon > 0$, we would like to ensure that the expected relative error satisfies

$$\mathbb{E}\left[\frac{\|\widehat{\boldsymbol{B}}_s - \boldsymbol{B}\|}{\|\boldsymbol{B}\|}\right] \leq \varepsilon. \tag{2.8}$$

Algorithm 2. *Approximate matrix multiplication: Monte Carlo method.*

Input: Input matrix $\boldsymbol{A} \in \mathbb{R}^{n \times d}$, number s of samples
Output: Approximation $\widehat{\boldsymbol{B}}_s$ for the product $\boldsymbol{B} = \boldsymbol{AA}^*$

---

 1 **function** ApproxMatrixMult($\boldsymbol{A}, s$)
 2      $c_j = \|\boldsymbol{a}_j\|_2^2$ for $j = 1, \ldots, d$ $\triangleright$ $\boldsymbol{a}_j$ is jth column of matrix
 3      $p_j = c_j / \sum_{i=1}^{d} c_i$ for $j = 1, \ldots, d$ $\triangleright$ Sampling probabilities
 4   **for** $i = 1, \ldots, s$ **do**
 5   Sample k_i from distribution $(p_j : j = 1, \ldots, d)$
 6          $\boldsymbol{Y}_i = p_{k_i}^{-1} \boldsymbol{a}_{k_i} \boldsymbol{a}_{k_i}^*$ $\triangleright$ Sample random matrix
 7      $\widehat{\boldsymbol{B}}_s = s^{-1} \sum_{i=1}^{s} \boldsymbol{Y}_i$ $\triangleright$ Estimate for matrix product

---

The desideratum (2.8) is analogous with the variance bound (2.2) for the trace estimator. The spectral-norm error provides very strong control on the quality of the matrix approximation. It allows us to compare linear functionals of the two matrices, their singular values, their singular vectors, and more [TW23, Sec. 4.2].

To analyze the dependence of the relative error (2.8) on the number s of samples, we rely on the matrix Monte Carlo theorem [Tro15, Cor. 6.2.1].

Theorem 2.3 (Matrix Monte Carlo; Tropp 2015). *Define $\boldsymbol{B} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{Y} \in \mathbb{R}^{m \times n}$ as in Section 2.2.1. For the random matrix $\boldsymbol{Y}$, define the per-sample second moment $v(\boldsymbol{Y})$ and the uniform norm bound $L(\boldsymbol{Y})$:*

$$v(\boldsymbol{Y}) := \max\{\|\mathbb{E}[\boldsymbol{YY}^*]\|, \|\mathbb{E}[\boldsymbol{Y}^*\boldsymbol{Y}]\|\} \quad and \quad L(\boldsymbol{Y}) := \sup \|\boldsymbol{Y}\|.$$

Then the randomized matrix approximation $\widehat{\boldsymbol{B}}_s$ based on s samples (2.7) satisfies the error bound

$$\mathbb{E} \|\widehat{\boldsymbol{B}}_s - \boldsymbol{B}\| \leq \sqrt{\frac{2v(\boldsymbol{Y}) \log(m+n)}{s}} + \frac{L(\boldsymbol{Y}) \log(m+n)}{s}.$$

For a tolerance $\varepsilon \in (0, 1]$, the relative error bound (2.8) holds when the sample complexity s satisfies

$$s \geq 4 \max \left\{ \frac{\varepsilon^{-2} v(\boldsymbol{Y})}{\|\boldsymbol{B}\|^2}, \frac{\varepsilon^{-1} L(\boldsymbol{Y})}{\|\boldsymbol{B}\|} \right\} \cdot \log(m+n). \tag{2.9}$$

Theorem 2.3 is a corollary of the matrix Bernstein inequality [Tro15, Thm. 6.1.1], a fundamental tool from nonasymptotic random matrix theory.

To understand this result, observe that the sample complexity s depends on the two statistics $v(\boldsymbol{Y})/\|\boldsymbol{B}\|^2$ and $L(\boldsymbol{Y})/\|\boldsymbol{B}\|$. These scale-free quantities reflect the fluctuations of the simple approximation $\boldsymbol{Y}$ relative to the spectral norm $\|\boldsymbol{B}\|$ of the target matrix. To obtain an accurate approximation, we must tune the probability distribution $(p_j : j = 1, \ldots, J)$ on the indices to control both quantities.

Next, compare the sample complexity (2.9) for matrix approximation with the sample complexity (2.4) for trace estimation. As $\varepsilon \to 0$, the first term in the matrix case exhibits the same quadratic growth ε^{-2} as we see in the scalar case. Once again, this phenomenon reflects the central limit theorem. On the other hand, when $\varepsilon \geq v(\boldsymbol{Y})/L(\boldsymbol{Y})$, the linear term ε^{-1} dominates. This feature of the bound results from the possibility that a few samples with large norm can ruin the approximation.

We also pay a small factor $\log(m+n)$ that is logarithmic in the total dimension of the target matrix. This term arises because we are bounding the spectral-norm error, and it cannot be removed in general. The factor can often be replaced with a quantity that reflects the intrinsic dimension of the target matrix $\boldsymbol{B}$; see [Tro15, Chap. 7].

2.2.3. *Example: Approximate matrix multiplication.* As an illustrative example of matrix Monte Carlo approximation, we study a randomized algorithm for approximate matrix multiplication (Algorithm 2). This approach was initially proposed by Drineas & Kannan [DK01], who were inspired by [FKV98]. The paper [HI15] contains a more detailed investigation of the special case that we present here.

Consider a rectangular matrix $A \in \mathbb{R}^{n \times d}$. Heuristically, we think of the case where the number d of columns is much larger than number n of rows. Suppose we want to approximate the outer product $B := AA^*$. A direct computation expends $\mathcal{O}(n^2 d)$ arithmetic operations. By *approximating* the product, we can try to reduce the cost of the computation.

> **Problem** (Approximate matrix multiplication). Let A be a (rectangular) matrix. The task is to approximate the outer product $B = AA^*$ while limiting the number of arithmetic operations.

We will see that the approximate matrix multiplication problem fits into the framework for matrix approximation by sampling that we introduced in Section 2.2.1.

Observe that we can express the product as a sum of rank-one matrices:

$$B = AA^* = \sum_{j=1}^{d} a_j a_j^* =: \sum_{j=1}^{d} B_j. \qquad (2.10)$$

We have written a_j for the jth column of the matrix A. Explicitly, $B_j = a_j a_j^*$. Using s samples, the Monte Carlo approximation of the product B takes the form

$$\widehat{B}_s := \frac{1}{s} \sum_{i=1}^{s} a_{k_i} a_{k_i}^*$$

where $k_1, \ldots, k_s$ are chosen independently at random from the column indices $\{1, \ldots, d\}$.

To design the Monte Carlo approximation algorithm, we must construct a sampling distribution $(p_j : j = 1, \ldots, d)$ on the indices. It is natural to sample the matrices B_j in proportion to their norms, as bigger summands contribute more to the sum. Thus, we select

$$p_j := \frac{\|B_j\|}{\sum_{j=1}^{d} \|B_j\|} = \frac{\|a_j\|_2^2}{\|A\|_F^2}. \qquad (2.11)$$

These probabilities can be computed with one pass over the matrix A, at a cost of $\mathcal{O}(nd)$ arithmetic operations.

With the sampling probabilities in place, define the random matrix Y as in (2.6). We can easily find a uniform upper bound for the norm of Y. Indeed,

$$L := \sup \|Y\| = \max_j \|p_j^{-1} B_j\| = \|A\|_F^2.$$

To compute the per-sample second moment, we need to work a little harder. Since Y is a symmetric matrix,

$$v(Y) := \|\mathbb{E}[Y^2]\| = \left\| \sum_{j=1}^{d} \left(p_j^{-2} B_j^2\right) \cdot p_j \right\| = \left\| \sum_{j=1}^{d} p_j^{-1} \|a_j\|^2 \cdot a_j a_j^* \right\|$$

$$= \|A\|_F^2 \cdot \left\| \sum_{j=1}^{d} a_j a_j^* \right\| = \|A\|_F^2 \cdot \|B\|.$$

In the first line, we have used the definition (2.10) of the matrices B_j. To reach the second line, we introduced the sampling probabilities (2.11). Note that the two statistics satisfy the identities

$$\frac{v(Y)}{\|B\|^2} = \frac{L(Y)}{\|B\|} = \frac{\|A\|_F^2}{\|B\|} = \frac{\mathrm{tr}(B)}{\|B\|} = \mathrm{int\,dim}(B).$$

In other words, the statistics that control the quality of the Monte Carlo approximation both coincide with the intrinsic dimension of the product. This is a consequence of the thoughtful choice of sampling probabilities.

Applying the matrix Monte Carlo theorem (Theorem 2.3), we obtain a bound for the number s of terms that we need to sample to approximate the matrix product. Let $\varepsilon \in (0, 1)$ be an error tolerance. Suppose that

$$s \geq 4\varepsilon^{-2} \cdot \operatorname{int}\dim(\boldsymbol{B}) \cdot \log(2n).$$

Then the approximation $\widehat{\boldsymbol{B}}_s$ of the product $\boldsymbol{B}$ using s samples satisfies the relative error bound (2.8). In other words, we obtain an accurate relative approximation when the number s of samples is roughly proportional to $\operatorname{int}\dim(\boldsymbol{B})$, the intrinsic dimension of the product. After forming the sampling probabilities with $\mathcal{O}(nd)$ operations, the cost of approximating the product is $\mathcal{O}(n^2 s)$ operations. For contrast, forming the full product costs $\mathcal{O}(n^2 d)$ operations. When $\operatorname{int}\dim(\boldsymbol{B}) \ll d$, we obtain a significant reduction in the computational effort.

It is interesting to investigate the performance of approximate matrix multiplication with alternative sampling probabilities. The adaptive sampling probabilities (2.11) are the most natural, but some applications force us to use uniform sampling probabilities: $p_j = 1/d$ for each j. As an exercise, you should work out sample complexity of approximate matrix multiplication with uniform sampling.

To avoid complications, we only considered multiplication of a matrix with its own transpose. Nevertheless, a similar algorithm is valid for approximating the product of two different matrices. For more details, see [Tro15, Secs. 6.4 and 7.3.3].

2.2.4. *Example: Matrix sparsification.* As another simple example of matrix Monte Carlo approximation, consider the problem of replacing a matrix by a sparse proxy. This is called "matrix sparsification"; its nominal applications include compression or acceleration of downstream computations. Randomized sparsification methods were first studied by Achlioptas & McSherry [AM01].

We can apply the same framework as before. Observe that every matrix $\boldsymbol{B} \in \mathbb{R}^{m \times n}$ admits the entrywise decomposition

$$\boldsymbol{B} = \sum_{j=1}^{m} \sum_{k=1}^{n} b_{jk} \mathbf{E}_{jk},$$

where b_{jk} denotes the (j, k) entry of matrix $\boldsymbol{B}$ and the $\mathbf{E}_{jk}$ are the elements of the standard basis for matrices. It requires some cleverness to identify an effective choice for the sampling probabilities. Kundu & Drineas [KD14] proposed the distribution

$$p_{jk} = \frac{1}{2} \left(\frac{|b_{jk}|^2}{\|\boldsymbol{B}\|_{\mathrm{F}}^2} + \frac{|b_{jk}|}{\|\boldsymbol{B}\|_{\ell_1}} \right) \quad \text{for } j = 1, \ldots, m \text{ and } k = 1, \ldots, n.$$

In this expression, $\|\cdot\|_{\ell_1}$ reports the sum of the absolute values of the entries of a matrix. With these probabilities, we can obtain an approximation with spectral-norm relative error $\varepsilon > 0$ with sample complexity

$$s \geq 8\varepsilon^{-2} \cdot \operatorname{int}\dim(\boldsymbol{B}\boldsymbol{B}^*) \cdot \max\{m, n\} \log(m + n)$$

Thus, the sparsity of the approximation is roughly the rank of $\boldsymbol{B}$ times the maximum number of entries in a row or column. We leave the detailed analysis of this scheme as an exercise for the reader.

2.2.5. *Matrix Monte Carlo: Improvements.* In practice, the sampling complexity of a matrix Monte Carlo approximation often depends on information that is not easily accessible. Fortunately, we can obtain *a posteriori* bounds on the quality of the approximation using simulation methods, such as the jackknife [ET24] or the bootstrap [Lop19].

As with scalar Monte Carlo, the sampling complexity of matrix Monte Carlo approximation scales as $s \approx \varepsilon^{-2}$. To address this problem, we may need to incorporate variance reduction techniques [Tro15, Sec. 6.2.3]. A simple idea is to adjust the sampling strategy so that "significant" terms always appear in the approximation. At the cost of introducing a bias into the approximation, we can also avoid sampling "insignificant" terms that generate a lot of variability. Beyond these first steps, the literature does not really discuss more sophisticated approaches for variance reduction in the matrix setting.

2.2.6. *Matrix Monte Carlo: Further examples.* Our examples of matrix Monte Carlo approximation are intended primarily for illustration, rather than as serious numerical methods. Nevertheless, these techniques can serve as ingredients in more sophisticated algorithms. The matrix Monte Carlo method can also be used to address other challenges in numerical linear algebra.

- **Graph sparsification [SS11].** We can approximate the Laplacian of an undirected graph by sampling edges in proportion to their effective resistances. By this procedure, every graph on n vertices admits a spectrally accurate approximation with $\mathcal{O}(n \log n)$ edges.
- **Sparse Cholesky factorization [KS16].** Inside their algorithm for solving Laplacian linear systems, Kyng & Sachdeva rely on a method for replacing a clique in a combinatorial graph with a sparser graph. This amounts to a matrix Monte Carlo approximation of the subgraph. See Section 6.1.3 for more discussion, or see [Tro19] for a full exposition of this algorithm and its analysis.
- **Random features [RR08].** We can approximate certain types of kernel matrices efficiently using a sum of random rank-one matrices. This method is called a *random features* approximation. The benefit is that the random features for each data point can be computed independently of the other data points. The analysis of random features as a matrix Monte Carlo method appears in [LP+14; Tro15; Tro19].
- **Quantum state tomography [Guţ+20].** By the laws of quantum mechanics, measuring (copies of) a quantum state results produces a sequence of random numbers. We can use these random measurement values and the matrix elements of the measurement ensemble to construct a matrix Monte Carlo approximation of the quantum state.

The literature contains many further examples of matrix approximation by sampling.

3. Random initialization

As you know, the performance of an iterative algorithm often depends on the initialization. By starting an iterative algorithm at a randomly chosen point, we can encourage the algorithm to converge more rapidly to a solution. Alternatively, we can avoid bad starting points that may result in slow convergence or outright failure.

> **Theme** (Random initialization). Initialize an (iterative) algorithm at a random point to make a favorable trajectory likely.

This technique has played a role in numerical linear algebra and optimization for decades. But researchers did not succeed in *quantifying* the benefits of the random initialization until much later. We now recognize that the random start plays a fundamental role in the performance of certain numerical methods. Let us tell part of this story.

This section begins with a classical randomized algorithm for computing the maximum eigenvalue of a psd matrix. Afterward, we study a more recent randomized algorithm for producing a low-rank approximation of a rectangular matrix. Both of these algorithms depend on a random initialization to support their performance.

3.1. **The randomized power method.** A basic problem in spectral computation is to find the maximum eigenvalue of a psd matrix. In many applications, we have access to the matrix via the matvec operation. For example, consider the case where the matrix describes the discretization of an integral operator. The challenge is to determine the maximum eigenvalue of the matrix.

> **Problem** (Maximum eigenvalue). Suppose that A is a psd matrix, accessible via matvecs $x \mapsto Ax$. The task is to compute the maximum eigenvalue $\lambda_1(A)$ while limiting the number of matvecs.

By Abel's theorem, we cannot solve this problem using a finite number of arithmetic operations; we must resort to approximation. Consequently, we study iterative methods that repeatedly refine an estimate for the maximum eigenvalue.

3.1.1. *The power method.* To approximate the maximum eigenvalue of a psd matrix using matvecs, we can employ the power method. This venerable algorithm was developed in the early 20th century by Müntz and by von Mises. See Golub & van der Vorst [Gvoo] for some history and citations.

Let $A \in \mathbb{H}_n^+(\mathbb{R})$ be a real symmetric psd matrix that is nonzero. Our goal is to approximate the maximum eigenvalue $\lambda_1 := \lambda_1(A) > 0$ using matvecs with A.

The power method requires a nonzero starting vector $\omega \in \mathbb{R}^n$. We set the initial iterate $x_0 := \omega$ equal to the starting vector. At each iteration, the power method updates the iterate x_t using one matvec with the matrix A:

$$q_t := x_t / \|x_t\|_2 \quad \text{and} \quad x_{t+1} := Aq_t \quad \text{for } t = 0, 1, 2, \dots. \tag{3.1}$$

Each iteration provides a new estimate ξ_t for the maximum eigenvalue:

$$\xi_t := q_t^* x_{t+1} = q_t^*(Aq_t) \quad \text{for } t = 0, 1, 2, \dots. \tag{3.2}$$

We continue this process indefinitely, or until a stopping criterion is triggered.

To measure how well the estimate ξ_t approximates the maximum eigenvalue λ_1, we use relative error:

$$\text{err}(\xi_t) := \frac{\lambda_1 - \xi_t}{\lambda_1}. \tag{3.3}$$

It is easy to check that $\text{err}(\xi_t) \in [0, 1]$ since $\xi_t \in [0, \lambda_1]$. The bounds on ξ_t hold because it is a Rayleigh quotient of the psd matrix A.

3.1.2. *Power method: Intuition.* To understand why the power method procedure might work, we introduce an eigenvalue decomposition of the input matrix:

$$A = \sum_{i=1}^n \lambda_i v_i v_i^* \quad \text{where} \quad \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq 0.$$

The family $(v_i : i = 1, \dots, n)$ of eigenvectors composes an orthonormal basis for $\mathbb{R}^n$.

Unroll the iteration (3.1) to confirm that

$$q_t = \frac{A^t \omega}{\|A^t \omega\|_2} \quad \text{for } t = 0, 1, 2, \dots. \tag{3.4}$$

In terms of the spectral decomposition of the matrix A, it is clear that the power amplifies the largest eigenvalue:

$$A^t = \sum_{i=1}^n \lambda_i^t v_i v_i^*.$$

Eventually, as t increases, the power damps the eigenvalues strictly smaller than λ_1, and $A^t / \|A^t\|$ converges to an orthogonal projector onto the eigenspace associated with the eigenvalue λ_1. Provided that the starting vector ω has a nonzero component in this subspace, then the iterate q_t will converge toward a unit vector in this subspace. As a consequence, the associated eigenvector estimate ξ_t will converge toward λ_1.

3.1.3. *Power method: Convergence.* The classical analysis of the power method allows us to sharpen this intuition. To simplify the presentation, we rescale the nonzero matrix A so that the maximum eigenvalue $\lambda_1 = 1$. We will express the starting vector ω in the orthonormal basis of eigenvectors:

$$\omega = \sum_{i=1}^n \omega_i v_i \quad \text{where} \quad \omega_i = v_i^* \omega.$$

This notation for the components for the starting vector remains in force throughout the section. If you prefer, you could instead change basis so that A is diagonal.

Let us develop an expression for the relative error (3.3). A short calculation yields

$$
\begin{aligned}
\mathrm{err}(\xi_t) = 1 - \boldsymbol{q}_t^*(\boldsymbol{A}\boldsymbol{q}_t) &= \frac{(\boldsymbol{\omega}^*\boldsymbol{A}^t)(\boldsymbol{A}^t\boldsymbol{\omega})}{\|\boldsymbol{A}^t\boldsymbol{\omega}\|_2^2} - \frac{(\boldsymbol{\omega}^*\boldsymbol{A}^t)(\boldsymbol{A}^{t+1}\boldsymbol{\omega})}{\|\boldsymbol{A}^t\boldsymbol{\omega}\|_2^2} \\
&= \frac{\boldsymbol{\omega}^*\boldsymbol{A}^{2t}(\mathbf{I}_n - \boldsymbol{A})\boldsymbol{\omega}}{\|\boldsymbol{A}^t\boldsymbol{\omega}\|_2^2} = \frac{\sum_{i=1}^n \omega_i^2 \lambda_i^{2t}(1 - \lambda_i)}{\sum_{i=1}^n \omega_i^2 \lambda_i^{2t}}.
\end{aligned}
\tag{3.5}
$$

The first line relies on the definition (3.2) of the eigenvalue estimate ξ_t, along with the representation (3.4) of the iterate $\boldsymbol{q}_t$. In the second line, we calculate in the eigenvector basis.

The expression (3.5) for the relative error exposes the influence of the eigenvalues and the components of the starting vector in the eigenvector basis. From here, we can easily confirm the asymptotic convergence of the power method under minimal conditions.

Proposition 3.1 (Power method: Convergence). *Assume that the psd matrix $\boldsymbol{A}$ is nonzero. If the starting vector $\boldsymbol{\omega}$ satisfies $\omega_1 \neq 0$, then the power method produces maximum eigenvalue estimates ξ_t whose relative errors (3.3) satisfy $\mathrm{err}(\xi_t) \to 0$ as $t \to \infty$. In particular, $\xi_t \to \lambda_1$.*

Proof. Impose the normalization $\lambda_1 = 1$. For the scalar matrix $\boldsymbol{A} = \mathbf{I}$, we can easily check that $\mathrm{err}(\xi_t) = 0$ for all $t = 0, 1, 2, \dots$.

Otherwise, let $m < \dim(\boldsymbol{A})$ be the multiplicity of the largest eigenvalue: $1 = \lambda_1 = \cdots = \lambda_m > \lambda_{m+1}$. Then the identity (3.5) for the error ensures that

$$
\mathrm{err}(\xi_t) = \frac{\sum_i \omega_i^2 \lambda_i^{2t}(1 - \lambda_i)}{\sum_i \omega_i^2 \lambda_i^{2t}} = \frac{\sum_{i>m} \omega_i^2 \lambda_i^{2t}(1 - \lambda_i)}{(\omega_1^2 + \cdots + \omega_m^2) + \sum_{i>m} \omega_i^2 \lambda_i^{2t}}.
$$

Since $\omega_1^2 > 0$, the parenthesis in the denominator is strictly positive. For each $i > m$, we have the limit $\lambda_i^{2t} \to 0$ as $t \to \infty$. We conclude that $\mathrm{err}(\xi_t) \to 0$. $\qquad\square$

The proof of Proposition 3.1 emphasizes that $\xi_t \to \lambda_1$, the estimates for the maximum eigenvalue converge to the correct value, *if and only if* the starting vector has a nonzero component in the eigenspace associated with the maximum eigenvalue.

3.1.4. *Power method: Convergence rate.* As an aside, we can also exploit (3.5) to derive asymptotic convergence rates for the power method. These results enforce a *spectral gap*: $\lambda_1 > \lambda_2$.

Proposition 3.2 (Power method: Asymptotic convergence rate). *Assume that the psd matrix $\boldsymbol{A}$ satisfies $\lambda_1 > \lambda_2 > \lambda_3$. Assume that the first two components of the starting vector $\boldsymbol{\omega}$ are nonzero: $\omega_1, \omega_2 \neq 0$. Then the power method produces eigenvalue estimates ξ_t whose relative errors (3.3) satisfy*

$$
\frac{\mathrm{err}(\xi_{t+1})}{\mathrm{err}(\xi_t)} \to \left(\frac{\lambda_2}{\lambda_1}\right)^2 \quad \text{as } t \to \infty.
$$

This result states that $\xi_t \to \lambda_1$ exponentially fast with a rate that depends on the spectral gap. Let us stress that this is an *asymptotic* rate of convergence that only prevails in the limit. As with Proposition 3.1, it is both necessary and sufficient that $\omega_1, \omega_2 \neq 0$ to achieve this rate.

Proof. As before, we may rescale so that $\lambda_1 = 1$. Using the identity (3.5), we can form the ratio of the relative errors:

$$
\begin{aligned}
\frac{\mathrm{err}(\xi_{t+1})}{\mathrm{err}(\xi_t)} &= \frac{\sum_i \omega_i^2 \lambda_i^{2t+2}(1 - \lambda_i)}{\sum_i \omega_i^2 \lambda_i^{2t}(1 - \lambda_i)} \cdot \frac{\sum_i \omega_i^2 \lambda_i^{2t}}{\sum_i \omega_i^2 \lambda_i^{2t+2}} \\
&= \frac{\lambda_2^2 \cdot \omega_2^2 \lambda_2^{2t}(1 - \lambda_2) + \sum_{i>2} \omega_i^2 \lambda_i^{2t+2}(1 - \lambda_i)}{\omega_2^2 \lambda_2^{2t}(1 - \lambda_2) + \sum_{i>2} \omega_i^2 \lambda_i^{2t}(1 - \lambda_i)} \cdot \frac{\omega_1^2 + \sum_{i>1} \omega_i^2 \lambda_i^{2t}}{\omega_1^2 + \sum_{i>1} \omega_i^2 \lambda_i^{2t+2}} \to \lambda_2^2 \cdot 1.
\end{aligned}
$$

Algorithm 3. *Randomized power method.*

Input: Psd input matrix $A \in \mathbb{H}_n^+(\mathbb{R})$, number T of iterations
Output: Maximum eigenvalue estimate ξ_T

```
1  function RandPowerMethod(A; T)
2      Sample x_0 ~ NORMAL(0, I_n)                    ▷ Standard normal starting vector
3      for t = 0, ..., T do
4          q_t = x_t / ||x_t||_2                                  ▷ Normalize the iterate
5          x_{t+1} = A q_t                                            ▷ One matvec
6          ξ_t = q_t^* x_{t+1}                                  ▷ Eigenvalue estimate
```

In the second line, we have extracted the dominant term from each sum. As $t \to \infty$, each sum converges to zero faster than this dominant term because $\lambda_1 > \lambda_2 > \lambda_3$. We can cancel the effects of ω_1, ω_2 because they are both nonzero. $\square$

3.1.5. *The role of randomness.* Propositions 3.1 and 3.2 demand that the starting vector ω for the power method have nonzero components in the direction of the leading eigenspaces of the matrix A. How can we secure this property if we do not know anything about the eigenvectors? The classical prescription is to select the starting vector *at random*.

Using the structure of the maximum eigenvalue problem, we can design a distribution for the random starting vector in the power method. Since we do not have access to the eigenvector basis of the input matrix, we should select a distribution for the starting vector that is indifferent to the eigenvector basis. The natural choice is a *rotationally invariant* distribution. The power method is invariant to the scale of the starting vector, so the distribution of the radial component is at our option.

Recognizing these principles, we will draw the starting vector from the *standard normal* distribution: $\omega \sim \text{NORMAL}(0, I_n)$. In particular, the components $(\omega_1, \ldots, \omega_n)$ of the starting vector in the eigenvector basis compose an *independent* family of real standard normal variables: $\omega_i \sim \text{NORMAL}(0, 1)$. Algorithm 3 lists pseudocode for this version of the *power method with a random initialization*.

This random starting vector has a valuable feature that has long been appreciated by numerical analysts [Dem97, p. 155]. Almost surely, each component of the standard normal starting vector is nonzero: $\mathbb{P}\{\omega_i \neq 0\} = 1$. Thus, we can activate Propositions 3.1 and 3.2 to deduce that the randomized power method converges to the maximum eigenvalue *with probability one*!

In fact, the random starting vector provides additional benefits that were first recognized [KW92] in the early 1990s. With a careful probabilistic analysis, we can derive several beautiful results about the randomized power method. These theorems appear in the upcoming subsections.

For intuition, a closer examination of the relative error (3.5) suggests that the power method converges most quickly when the first component ω_1 of the starting vector is large, while the remaining components $\omega_2, \ldots, \omega_n$ are small. Since the components are independent random variables with $\mathbb{E}[\omega_i^2] = 1$, the typical size of each coordinate ω_i is about one. In other words, the standard normal distribution does a reasonable job at achieving our goals for the components of the starting vector. This is the best we can hope for, absent knowledge about the leading eigenspace.

3.1.6. *Randomized power method: Spectral gap.* The next step is to quantify how quickly the randomized power method converges to the maximum eigenvalue. This theory adapted from a remarkable paper of Kuczyński & Woźniakowski [KW92]. As observed in [Tro20, Lec. 2], we can dramatically simplify their calculations using the properties of the standard normal distribution.

Echoing the classical analysis, we begin with a result that imposes an assumption on the spectral gap of the input matrix. The next section will show how to remove this condition.

Theorem 3.3 (Randomized power method: Spectral gap). *Assume that $A \in \mathbb{H}_n^+(\mathbb{R})$ is a psd matrix with a spectral gap: $\lambda_1 > \lambda_2$. Then the randomized power method (Algorithm 3) produces eigenvalue estimates ξ_t whose relative errors (3.3) satisfy*

$$\mathbb{E} \operatorname{err}(\xi_t) \leq \sqrt{2n} \cdot \left(\frac{\lambda_2}{\lambda_1} \right)^t \quad \textit{for } t = 0, 1, 2, \ldots. \tag{3.6}$$

In contrast with the classical convergence result Proposition 3.2, Theorem 3.3 offers a *nonasymptotic* error bound that is valid in each iteration t. It may be disappointing that the expected error (3.6) only improves by a factor of (λ_2/λ_1) at each iteration, which is substantially worse than the asymptotic rate $(\lambda_2/\lambda_1)^2$. The bound (3.6) reflects the possibility that the first component ω_1 of the starting vector is unusually small. For the randomized power method, Theorem 3.3 is essentially sharp [KW92, Thm. 3.1(b)], but see [Tro22] for more texture.

What else does the bound (3.6) tell us about the performance of the randomized power method? Let us rewrite the error in terms of the *relative spectral gap*, defined as

$$\gamma := \frac{\lambda_1 - \lambda_2}{\lambda_1} \in [0, 1].$$

Theorem 3.3 yields the exponential convergence rate

$$\mathbb{E} \operatorname{err}(\xi_t) \leq \sqrt{2n} \cdot e^{-\gamma t} \quad \text{for } t = 0, 1, 2, \ldots.$$

For any tolerance $\varepsilon \in (0, 1)$, we can achieve a relative error ε after a predictable number t of iterations:

$$t \geq \frac{\log \sqrt{2n} + \log (1/\varepsilon)}{\gamma} \quad \text{implies} \quad \mathbb{E} \operatorname{err}(\xi_t) \leq \varepsilon.$$

This formula reveals that the randomized power method exhibits a *burn-in period*. It may not make any progress for the first $\log(2n)/(2\gamma)$ iterations. Afterward, it begins to converge at an exponential rate that depends on the relative spectral gap γ. The logarithmic dependence $\log(1/\varepsilon)$ on the tolerance ε indicates that we can achieve very small relative errors after a modest number of iterations, relative to the spectral gap γ. For comparison, recall that a simple Monte Carlo approximation suffers an ε^{-2} dependence on the tolerance.

Proof of Theorem 3.3. As always, we may normalize the matrix so $\lambda_1 = 1$. Start with the expectation of the bound (3.5) on the relative error:

$$\mathbb{E} \operatorname{err}(\xi_t) = \mathbb{E} \left[\frac{\sum_i \omega_i^2 \lambda_i^{2t}(1 - \lambda_i)}{\sum_i \omega_i^2 \lambda_i^{2t}} \right] \leq \mathbb{E}_{\omega_2, \ldots, \omega_n} \mathbb{E}_{\omega_1} \left[\frac{\sum_{i>1} \omega_i^2 \lambda_i^{2t}}{\omega_1^2 + \sum_{i>1} \omega_i^2 \lambda_i^{2t}} \right]. \tag{3.7}$$

To bound the numerator, note that $0 \leq 1 - \lambda_i \leq 1$. Since the components $\omega_1, \ldots, \omega_n$ are *independent* standard normal variables, we can take the expectation $\mathbb{E}_{\omega_1}$ with respect to the first component before averaging over the remaining components.

To continue, we rely on an expectation bound for a standard normal variable [Tro20, Fact 2.8].

Fact 3.4 (A Gaussian integral). *Fix $c \geq 0$. For a real standard normal variable $g \sim \text{NORMAL}(0, 1)$,*

$$\mathbb{E}_g \left[\frac{c}{g^2 + c} \right] \leq \sqrt{\frac{\pi}{2} \cdot c}.$$

Employing Fact 3.4 with $c = \sum_{i>1} \omega_i^2 \lambda_i^{2t}$ and $g = \omega_1$, we can bound the error as

$$\mathbb{E} \operatorname{err}(\xi_t) \leq \mathbb{E}_{\omega_2, \ldots, \omega_n} \sqrt{\frac{\pi}{2} \sum_{i>1} \omega_i^2 \lambda_i^{2t}} \leq \sqrt{\frac{\pi}{2} \sum_{i>1} \mathbb{E}[\omega_i^2] \cdot \lambda_i^{2t}} \leq \sqrt{\frac{\pi}{2} (n-1)} \cdot \lambda_2^t$$

The second inequality is Jensen's. Third, we recall that $\mathbb{E}[\omega_i^2] = 1$ and introduce the bound $\lambda_i \leq \lambda_2$ for $i > 1$. Finally, invoke the numerical inequality $\pi/2 < 2$ and remove the normalization.   □

3.1.7. *Randomized power method: No spectral gap.* The probabilistic analysis of the randomized power method also points to new phenomena that are invisible to the classical analysis. Without assuming a spectral gap, we can still obtain a convergence rate for the randomized power method. This result is also due to Kuczyński & Woźniakowski [KW92], with roots in the work of Dixon [Dix83]. The proof here is adapted from [Tro20, Lec. 2].

Theorem 3.5 (Randomized power method: No spectral gap). *Assume that $A \in \mathbb{H}_n^+(\mathbb{R})$ is a nonzero psd matrix. Then the randomized power method (Algorithm 3) produces eigenvalue estimates ξ_t whose relative errors* (3.3) *satisfy*

$$\mathbb{E}\,\mathrm{err}(\xi_t) \leq \frac{1 + \log \sqrt{2n} + \log t}{t} \quad \text{for } t = 1, 2, 3, \ldots.$$

In particular, for any tolerance $\varepsilon \in (0, 1)$,

$$t \geq \frac{1 + \log \sqrt{2n} + \log t}{\varepsilon} \quad \text{implies} \quad \mathbb{E}\,\mathrm{err}(\xi_t) \leq \varepsilon.$$

Before turning to the proof, a few remarks. Theorem 3.5 is a *nonasymptotic* result that holds in each iteration. It reveals a *burn-in period* of about $\log \sqrt{2n}$ steps before the randomized power method starts to make progress. Afterward, the error decays at the *polynomial rate* of $1/t$. This is far worse than the exponential rate attained when we have a spectral gap, but it requires *no assumptions*. Keep in mind that Theorems 3.3 and 3.5 are both valid, so we can employ whichever bound is strongest.

Proof of Theorem 3.5. Normalize the matrix so that $\lambda_1 = 1$. Nominate a parameter $\beta \in [0, 1]$, which will be determined later. As compared with the proof of Theorem 3.3, the key new insight is to split the sum in the numerator into terms where $\lambda_i \geq \beta$ and where $\lambda_i < \beta$.

Proceeding in this way, the expected relative error (3.5) satisfies the bound

$$\mathbb{E}\,\mathrm{err}(\xi_t) = \mathbb{E}\left[\frac{\sum_i \omega_i^2 \lambda_i^{2t}(1 - \lambda_i)}{\sum_i \omega_i^2 \lambda_i^{2t}} \right] \leq \mathbb{E}\left[\frac{(1 - \beta)\sum_{\lambda_i \geq \beta} \omega_i^2 \lambda_i^{2t} + \sum_{\lambda_i < \beta} \omega_i^2 \lambda_i^{2t}}{\sum_i \omega_i^2 \lambda_i^{2t}} \right]$$

$$\leq (1 - \beta) + \mathbb{E}_{\omega_2,\ldots,\omega_n} \mathbb{E}_{\omega_1}\left[\frac{\sum_{\lambda_i < \beta} \omega_i^2 \lambda_i^{2t}}{\omega_1^2 + \sum_{\lambda_i < \beta} \omega_i^2 \lambda_i^{2t}} \right]$$

Invoke Fact 3.4 with $c = \sum_{\lambda_i < \beta} \omega_i^2 \lambda_i^{2t}$ and $g = \omega_1$ to obtain

$$\mathbb{E}\,\mathrm{err}(\xi_t) \leq (1 - \beta) + \mathbb{E}_{\omega_2,\ldots,\omega_n} \sqrt{\frac{\pi}{2} \sum_{\lambda_i < \beta} \omega_i^2 \lambda_i^{2t}} \leq (1 - \beta) + \sqrt{\frac{\pi}{2} \sum_{\lambda_i < \beta} \lambda_i^{2t}}$$

$$\leq (1 - \beta) + \sqrt{\frac{\pi}{2}(n - 1)} \cdot \beta^t \leq (1 - \beta) + \sqrt{2n} \cdot \mathrm{e}^{-(1-\beta)t}.$$

The second inequality is Jensen's. Then we made the coarse bound $\lambda_i \leq \beta$ for each eigenvalue that participates in the sum. The last inequality is numerical. Finally, minimize the right-hand side over $\beta \in [0, 1]$ to arrive at advertised result.   □

3.1.8. *Discussion.* By exploiting the properties of the standard normal starting vector, we can systematically establish a body of results on the behavior of the randomized power method. We can reproduce classical facts [Par98, Thm. 4.2.1] about the performance in the presence of a spectral gap, including the burn-in period and nonasymptotic exponential convergence rates. We can also identify the surprising

phenomenon that the randomized power method makes progress *even without a spectral gap*. To the best of our knowledge, this result first emerged from the probabilistic analysis of the algorithm.

This section focuses on the randomized power method because it is both familiar and simple. In practice, however, we typically rely on more powerful algorithms for estimating maximum eigenvalues. Kuczyński & Woźniakowski [KW92] also proposed a probabilistic analysis of the Lanczos method with a random starting vector. Their work demonstrates that the randomized Lanczos method has a burn-in period, nonasymptotic exponential convergence in the presence of a spectral gap, and nonasymptotic polynomial convergence even without a spectral gap. Related results [Tro22] are also available for the randomized block power method and the randomized block Lanczos method; these variants use multiple starting vectors to enhance convergence. In all of these cases, the probabilistic analysis reveals new insights about the performance of these fundamental numerical algorithms.

3.2. **The randomized SVD.** We have seen that random initialization can play a role in the computation of scalar quantities, such as the maximum eigenvalue of a psd matrix. In this section, we develop a second example where random initialization supports an algorithm for a matrix computation: the randomized SVD for low-rank matrix approximation.

Consider a rectangular matrix that we can access with via matvec operations. A basic task in numerical linear algebra is to produce a low-rank approximation of the matrix. For example, principal component analysis reduces to low-rank approximation. We frame the following problem:

> **Problem** (Low-rank matrix approximation). Suppose B is a rectangular matrix, accessible via matvecs $x \mapsto Bx$ and $y \mapsto B^*y$. The task is to produce a low-rank approximation of B that is competitive with a best approximation of similar rank.

3.2.1. *Singular-value decompositions and low-rank approximations.* We begin with some deep background on low-rank approximation of matrices. Consider a rectangular matrix $B \in \mathbb{R}^{m \times n}$, and recall that $m \wedge n := \min\{m, n\}$. We introduce a singular-value decomposition (SVD) of the matrix:

$$B = \sum_{i=1}^{m \wedge n} \sigma_i u_i v_i^* \quad \text{where} \quad \sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{m \wedge n} \geq 0. \tag{3.8}$$

The right singular vectors $(v_i : i = 1, \ldots, m \wedge n)$ compose an orthonormal system in $\mathbb{R}^n$, while the left singular vectors $(u_i : i = 1, \ldots, m \wedge n)$ compose an orthonormal system in $\mathbb{R}^m$. As usual, the singular values σ_i are listed in decreasing order. Geometrically, the SVD tells us that the matrix B maps the unit sphere in $\mathbb{R}^n$ to an ellipsoid in $\mathbb{R}^m$; see Figure 3.1a.

What can we say about the optimal low-rank approximation of the matrix B? Given a rank parameter $r \leq \min\{m, n\}$, define the orthogonal projector P_r onto the span of the leading r left singular vectors:

$$P_r := \sum_{i=1}^{r} u_i u_i^*.$$

The Eckart–Young theorem [EY36] identifies the best rank-r approximation with respect to the Frobenius norm:

$$\min_{\text{rank}(M) \leq r} \|B - M\|_F^2 = \|B - P_r B\|_F^2 = \sum_{i>r} \sigma_i^2. \tag{3.9}$$

In words, the best approximation is achieved by projecting the target matrix onto the r-dimensional subspace that captures most of the action of the matrix. Throughout this discussion, we focus on the Frobenius-norm error to obtain more transparent results.

In some applications, it is unnecessary to compute the best approximation to high accuracy. Instead, we can relax our requirements. Let $s = r + \ell$ for a small natural number ℓ. For a tolerance $\varepsilon > 0$, we seek a *rank-s* approximation $\widehat{B}_s$ that competes with the best *rank-r* approximation (3.9):

$$\|B - \widehat{B}_s\|_F^2 \leq (1 + \varepsilon) \cdot \|B - P_r B\|_F^2 = (1 + \varepsilon) \cdot \sum_{i>r} \sigma_i^2. \tag{3.10}$$
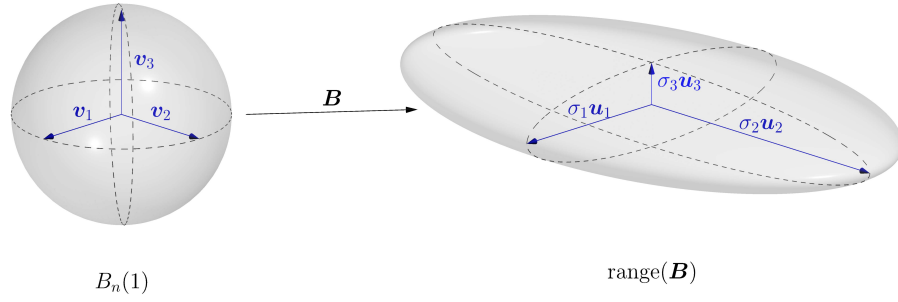
(a) Geometry of the SVD. The linear map $\boldsymbol{B} \in \mathbb{R}^{n \times m}$ transforms the unit sphere in $\mathbb{R}^n$ to an ellipsoid in $\mathbb{R}^m$ with semiaxes determined by the singular values and the singular vectors of $\boldsymbol{B}$.
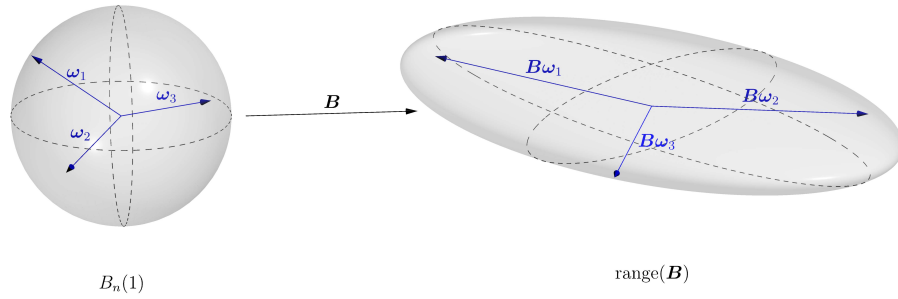
(b) Images of random vectors. A normalized Gaussian vector $\boldsymbol{\omega}/\|\boldsymbol{\omega}\|_2$ is distributed uniformly on the unit sphere. The image $\boldsymbol{B}\boldsymbol{\omega}$ tends to align with the major semiaxes of the ellipsoid defined by $\boldsymbol{B}$. To cover the leading r directions, we need to use slightly more than r random test vectors.

Figure 3.1. Randomized SVD: Intuition. These schematics illustrate the geometric ideas behind the SVD and the randomized SVD.

When the best rank-r approximation error is tiny, it is not a big deal to pay a modest factor $(1 + \varepsilon)$ more. This situation occurs in scientific computing applications where the singular values of the matrix decay exponentially fast [HMT11, Sec. 7.1].

3.2.2. *Randomized SVD: Intuition.* Our goal is to produce an estimate for the best rank-r approximation $\boldsymbol{P}_r \boldsymbol{B}$ of the input matrix. Instead of computing the projector $\boldsymbol{P}_r$ onto the r leading left singular vectors, we will use randomness to estimate these directions. We can trace this insight to work of Frieze et al. [FKV98]. Martinsson et al. [MRT06] developed this idea into a practical algorithm, which was crystallized in the paper [HMT11].

The method is to multiply random test vectors into the matrix $\boldsymbol{B}$ to identify salient directions. We have the intuition that the image of a random vector tends to be aligned with the leading left singular vectors. By repetition, we can cover most of the significant directions in the range of the projector $\boldsymbol{P}_r$. Since we do not know the orthonormal basis of right singular vectors, it is natural to draw the random vectors from a rotationally invariant distribution. See Figure 3.1b for an illustration.

We can describe this approach mathematically [TW23, Sec. 2.1]. Draw a standard normal test vector $\boldsymbol{\omega} \in \mathbb{R}^n$. The image of the random vector satisfies

$$\boldsymbol{B}\boldsymbol{\omega} = \sum_{i=1}^{m \wedge n} \sigma_i \, \boldsymbol{u}_i (\boldsymbol{v}_i^* \boldsymbol{\omega}) =: \sum_{i=1}^{m \wedge n} \omega_i \sigma_i \, \boldsymbol{u}_i$$

Algorithm 4. *Randomized SVD.*

Input: Input matrix $\boldsymbol{B} \in \mathbb{R}^{m \times n}$, number s of samples
Output: Factors $\boldsymbol{Q} \in \mathbb{R}^{m \times s}$ and $\boldsymbol{C} \in \mathbb{R}^{s \times n}$ of the approximation $\widehat{\boldsymbol{B}}_s = \boldsymbol{QC}$

---

1 **function** RandSVD($\boldsymbol{B}$)
2   Sample $\boldsymbol{\Omega} \sim \text{NORMAL}(\boldsymbol{0}, \mathbf{I}_m \otimes \mathbf{I}_n) \in \mathbb{R}^{m \times n}$ ▷ Standard normal test matrix
3      $\boldsymbol{Y} = \boldsymbol{B\Omega}$ ▷ s matvecs with $\boldsymbol{B}$
4      $\boldsymbol{Q} = \text{orth}(\boldsymbol{Y})$ ▷ Orthonormal basis for range of $\boldsymbol{Y}$
5      $\boldsymbol{C} = \boldsymbol{Q}^*\boldsymbol{B}$ ▷ s matvecs with $\boldsymbol{B}^*$
6      $(\widehat{\boldsymbol{U}}_0, \widehat{\boldsymbol{\Sigma}}, \widehat{\boldsymbol{V}}) = \text{svd}(\boldsymbol{C})$ ▷ **Optional:** Report SVD of approximation
7      $\widehat{\boldsymbol{U}} = \boldsymbol{Q}\widehat{\boldsymbol{U}}_0$ ▷ $\widehat{\boldsymbol{B}}_s = \widehat{\boldsymbol{U}}\widehat{\boldsymbol{\Sigma}}\widehat{\boldsymbol{V}}^*$

As before, the component $\omega_i := \boldsymbol{v}_i^* \boldsymbol{\omega}$ of the random vector along the ith right singular vector follows a standard normal distribution, and the components $(\omega_i : i = 1, \dots, m \wedge n)$ compose an independent family. On average, $\mathbb{E}[\omega_i^2] = 1$. Therefore, the image $\boldsymbol{B\omega}$ tends to align with the left singular vectors associated with large singular values.

By repeating this process with a statistically independent family $(\boldsymbol{\omega}^{(j)} : j = 1, \dots, s)$ of random test vectors, we can obtain a family $(\boldsymbol{B\omega}^{(j)} : j = 1, \dots, s)$ of vectors whose span contains most of range($\boldsymbol{P}_r$). The number $s = r + \ell$ of test vectors needs to be a bit larger than the target rank r to obtain coverage of the subspace with high probability.

3.2.3. *Randomized SVD: Algorithm.* Let us forge this intuition into an algorithm [HMT11, p. 227], called the ==randomized SVD==. For a rank parameter s, we draw a random test matrix:

$$\boldsymbol{\Omega} = \begin{bmatrix} \boldsymbol{\omega}^{(1)} & \dots & \boldsymbol{\omega}^{(s)} \end{bmatrix} \in \mathbb{R}^{n \times s} \quad \text{where } \boldsymbol{\omega}^{(j)} \sim \text{NORMAL}(\boldsymbol{0}, \mathbf{I}_n) \text{ i.i.d.}$$

We obtain the images of the test vectors using a matrix–matrix product with the input matrix:

$$\boldsymbol{Y} := \boldsymbol{B\Omega}.$$

This step requires s matvecs with $\boldsymbol{B}$. The orthogonal projector $\boldsymbol{P}_Y$ onto the range of $\boldsymbol{Y}$ serves as a proxy for the ideal projector $\boldsymbol{P}_r$. Computationally,

$$\boldsymbol{P}_Y := \boldsymbol{QQ}^* \quad \text{where} \quad \boldsymbol{Q} := \text{orth}(\boldsymbol{Y}).$$

The function orth returns an orthonormal basis for the range of a matrix, and it costs $\mathcal{O}(s^2 m)$ arithmetic operations. Finally, we report the approximation $\widehat{\boldsymbol{B}}_s$ in factored form:

$$\widehat{\boldsymbol{B}}_s := \boldsymbol{P}_Y \boldsymbol{B} = \boldsymbol{Q}(\boldsymbol{Q}^*\boldsymbol{B})$$

This step requires s matvecs with the transpose $\boldsymbol{B}^*$. If desired, we can report the SVD of the approximation after a small amount of additional work:

$$\widehat{\boldsymbol{B}}_s = (\boldsymbol{Q}\widehat{\boldsymbol{U}}_0)\widehat{\boldsymbol{\Sigma}}\widehat{\boldsymbol{V}}^* \quad \text{where} \quad (\widehat{\boldsymbol{U}}_0, \widehat{\boldsymbol{\Sigma}}, \widehat{\boldsymbol{V}}) = \text{svd}(\boldsymbol{Q}^*\boldsymbol{B}).$$

This step costs another $\mathcal{O}(s^2 n)$ arithmetic operations.

See Algorithm 4 for pseudocode for the randomized SVD algorithm. For a dense unstructured matrix $\boldsymbol{B}$, the total cost is $\mathcal{O}(smn)$. To compare, recall that a dense, economy-size SVD costs $\mathcal{O}(mn(m \wedge n))$. Classic Krylov subspace methods are competitive in cost with the randomized SVD, but they often fail for challenging problem instances [Li+17]. In contrast, the randomized SVD and its relatives are bulletproof.

3.2.4. *Randomized SVD: Analysis.* Concerning the randomized SVD, the main question is how many test vectors s are needed to obtain a rank-r approximation that satisfies (3.10). We present a result

from Halko et al. [HMT11, Thm. 10.5] that accurately predicts the performance of the randomized SVD algorithm.

Theorem 3.6 (Randomized SVD; Halko et al. 2011). *Consider a matrix $\boldsymbol{B} \in \mathbb{R}^{m \times n}$, and fix the target rank $r \leq m \wedge n$. When $s \geq r + 2$, the randomized SVD method (Algorithm 4) produces a random rank-s approximation $\widehat{\boldsymbol{B}}_s$ that satisfies*

$$\mathbb{E} \|\boldsymbol{B} - \widehat{\boldsymbol{B}}_s\|_{\mathrm{F}}^2 \leq \left(1 + \frac{r}{s - r - 1}\right) \cdot \sum_{i>r} \sigma_i^2(\boldsymbol{B}). \tag{3.11}$$

On average, to obtain a tolerance $\varepsilon > 0$ in the error bound (3.10), it suffices that $s = 1 + r + r/\varepsilon$.

This theorem places no restrictions on the input matrix (such as a spectral gap). To unpack the result, recall that the sum $\sum_{i>r} \sigma_i^2$ on the right-hand side of (3.11) is the Eckart–Young error (3.9) in the best rank-r approximation. The factor $\varepsilon = r/(s - r - 1)$ reflects the loss that we suffer from constructing the approximation $\widehat{\boldsymbol{B}}_s$ using the random projector $\boldsymbol{P}_Y$ instead of the ideal projector $\boldsymbol{P}_r$. By increasing the number s of samples, we can drive the error tolerance ε down as far as we like. In particular, $s = 2r + 1$ yields a tolerance $\varepsilon = 1$. This type of approximation might seem too weak to be useful. But keep in mind that the main use case for the randomized SVD algorithm is when the optimal error is tiny because the singular values decay exponentially [HMT11; TW23].

We will sketch the proof of Theorem 3.6 to show how the properties of the random test matrix $\boldsymbol{\Omega}$ enter into the bound. As with the randomized power method, the key insight is that the random test matrix $\boldsymbol{\Omega}$ aligns reasonably well with the leading right singular vectors of the input matrix $\boldsymbol{B}$. At the same time, the test matrix does not align too strongly with the trailing right singular vectors. To quantify these properties, we depend on some basic facts from random matrix theory.

As an aside, results like Theorem 3.6 also hold for the spectral-norm error [HMT11, Thm. 10.6], but the proof requires some technical arguments that do not contribute new insight about the mechanism that drives the randomized SVD algorithm.

Proof of Theorem 3.6. Fix the target rank $r \leq m \wedge n$ and the number s of random test vectors. We can express the SVD (3.8) of the input matrix $\boldsymbol{B}$ in block matrix form to isolate the leading r singular directions:

$$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{U}_r & \boldsymbol{U}_\perp \end{bmatrix} \begin{bmatrix} \boldsymbol{\Sigma}_r & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{\Sigma}_\perp \end{bmatrix} \begin{bmatrix} \boldsymbol{V}_r & \boldsymbol{V}_\perp \end{bmatrix}^*.$$

Explicitly, $\boldsymbol{U}_r \in \mathbb{R}^{m \times r}$, and $\boldsymbol{\Sigma}_r = \mathrm{diag}(\sigma_1, \dots, \sigma_r)$, and $\boldsymbol{V}_r \in \mathbb{R}^{n \times r}$.

Next, extract the components of the test matrix $\boldsymbol{\Omega}$ in the basis of right singular vectors:

$$\boldsymbol{\Omega}_r := \boldsymbol{V}_r^* \boldsymbol{\Omega} \quad \text{and} \quad \boldsymbol{\Omega}_\perp := \boldsymbol{V}_\perp^* \boldsymbol{\Omega}.$$

This decomposition is similar to the one we used to analyze the power method. The matrix $\boldsymbol{\Omega}_r$ describes the alignment of the test matrix with the leading right singular vectors, while $\boldsymbol{\Omega}_\perp$ describes the alignment with the trailing right singular vectors.

After some linear algebra (omitted), we can develop the following (deterministic) bound for the approximation error [HMT11, Thm. 9.1]. For a streamlined proof, see [TW23, Prop. 8.5].

Proposition 3.7 (Randomized SVD: Approximation error). *With notation as above, assume that $\mathrm{rank}(\boldsymbol{\Omega}_r) = r$. Then the rank-s approximation $\widehat{\boldsymbol{B}}_s$ produced by Algorithm 4 satisfies*

$$\|\boldsymbol{B} - \widehat{\boldsymbol{B}}_s\|_{\mathrm{F}}^2 \leq \|\boldsymbol{\Sigma}_\perp\|_{\mathrm{F}}^2 + \|\boldsymbol{\Sigma}_\perp \boldsymbol{\Omega}_\perp \boldsymbol{\Omega}_r^\dagger\|_{\mathrm{F}}^2 \tag{3.12}$$

The first term on the right-hand side of (3.12) reflects the trailing singular values of the matrix. Every rank-r approximation (3.9) suffers this loss. The second term is the deficit from using the test matrix $\boldsymbol{\Omega}$. We want the component $\boldsymbol{\Omega}_r$ in the leading directions to be well conditioned (ideally, an identity matrix). We want the component $\boldsymbol{\Omega}_\perp$ in the trailing directions to be as small as possible.

It is instructive to compare Proposition 3.7 with the error bound (3.7) for the power method when $t = 1$. Indeed, the quantity $\|\mathbf{\Omega}_r^{\dagger}\|_F^2$ is analogous to the term ω_1^2 in the *denominator* of (3.7). Meanwhile, the quantity $\|\mathbf{\Sigma}_{\perp}\mathbf{\Omega}_{\perp}\|_F^2$ is analogous with the sum $\sum_{i>1} \omega_i^2 \lambda_i^2$ in the numerator of (3.7).

Now, we can exploit the rotational invariance of the standard normal test matrix $\mathbf{\Omega}$ to evaluate the expectation of the error bound (3.12). Note that the columns of the leading singular vectors V_r and the trailing singular vectors $V_{\perp}$ are mutually orthogonal. Therefore, the associated components $\mathbf{\Omega}_r \in \mathbb{R}^{r \times s}$ and $\mathbf{\Omega}_{\perp} \in \mathbb{R}^{(n-r) \times s}$ of the test matrix are *standard normal matrices* that are statistically *independent*. In particular, if $s \geq r$, then $\mathrm{rank}(\mathbf{\Omega}_r) = r$ with probability one. This holds true regardless of the right singular vectors of B.

To evaluate the expected deficit in the error bound (3.12), we use independence to introduce conditional expectations. Thus,

$$\mathbb{E}\,\|\mathbf{\Sigma}_{\perp}\mathbf{\Omega}_{\perp}\mathbf{\Omega}_r^{\dagger}\|_F^2 = \mathbb{E}_{\mathbf{\Omega}_r}\left[\mathbb{E}_{\mathbf{\Omega}_{\perp}}\|\mathbf{\Sigma}_{\perp}\mathbf{\Omega}_{\perp}\mathbf{\Omega}_r^{\dagger}\|_F^2\right] = \mathbb{E}_{\mathbf{\Omega}_r}\left[\|\mathbf{\Sigma}_{\perp}\|_F^2 \cdot \|\mathbf{\Omega}_r^{\dagger}\|_F^2\right].$$

The second identity follows when we write out the Frobenius norm in coordinates and find the expectation over the entries of $\mathbf{\Omega}_{\perp}$ by direct calculation. The Frobenius norm of the remaining random matrix can be rewritten as a trace:

$$\mathbb{E}_{\mathbf{\Omega}_r}\|\mathbf{\Omega}_r^{\dagger}\|_F^2 = \mathbb{E}_{\mathbf{\Omega}_r}\,\mathrm{tr}\left[(\mathbf{\Omega}_r\mathbf{\Omega}_r^*)^{-1}\right] = \frac{r}{s-r-1}.$$

Indeed, since $\mathbf{\Omega}_r \in \mathbb{R}^{r \times s}$ is standard normal, the product $W = \mathbf{\Omega}_r\mathbf{\Omega}_r^*$ follows the standard WISHART$(s, \mathbf{I}_r)$ distribution with s degrees of freedom. The calculation of $\mathbb{E}\,\mathrm{tr}[W^{-1}]$ is a classical result from statistics [HMT11, Prop. A.6].

To conclude, take the expectation of (3.12), and introduce the last two displays:

$$\mathbb{E}\,\|B - \widehat{B}_s\|_F^2 \leq \|\mathbf{\Sigma}_{\perp}\|_F^2 + \mathbb{E}\,\|\mathbf{\Sigma}_{\perp}\mathbf{\Omega}_{\perp}\mathbf{\Omega}_r^{\dagger}\|_F^2 \leq \left(1 + \frac{r}{s-r-1}\right) \cdot \|\mathbf{\Sigma}_{\perp}\|_F^2.$$

Last, we recognize $\|\mathbf{\Sigma}_{\perp}\|_F^2$ as the sum of the squares of the trailing singular values of B. $\qquad\square$

3.2.5. *Discussion.* We have shown that the random choice of test matrix plays an integral role in the randomized SVD algorithm for low-rank matrix approximation. This example may seem different in spirit from the randomized initialization for the power method, but the algorithms are actually close relatives.

Indeed, we can extend the randomized SVD to an algorithm called *randomized subspace iteration* by repeated multiplication with the test matrix:

---

1. Sample $\mathbf{\Omega} = \begin{bmatrix} \boldsymbol{\omega}^{(1)} & \dots & \boldsymbol{\omega}^{(s)} \end{bmatrix} \in \mathbb{R}^{n \times s}$ with i.i.d. standard normal columns.
2. Set the initial iterate $X_0 = \mathbf{\Omega}$.
3. For $t = 1, 2, 3, \dots, T$, compute

$$Q_t := \mathrm{orth}(BX_{t-1}) \quad \text{and} \quad X_t := B^*Q_t$$

4. Return the approximation $\widehat{B}_s = Q_T X_T^*$.

The randomized SVD (Algorithm 4) is the special case of this procedure with $T = 1$.

Unrolling the recurrence, we find that randomized subspace iteration produces approximations

$$\widehat{B}_s = Q_t(Q_t^*B) \quad \text{where} \quad Q_t = \mathrm{orth}((BB^*)^{t-1}B\mathbf{\Omega}) \quad \text{for } t = 1, 2, 3, \dots.$$

Much as the power method drives its iterates toward the leading eigenspace, subspace iteration drives $\mathrm{range}(Q_t)$ so that it aligns with $\mathrm{range}(U_r)$, the leading left singular subspace of B. The random starting matrix $\mathbf{\Omega}$ has sufficient alignment with the leading right singular vectors V_r to begin this process. Meanwhile, the starting matrix $\mathbf{\Omega}$ is oblique with the trailing right singular vectors $V_{\perp}$, so they

do not interfere too much with the convergence. See [HMT11; TW23] for the analysis of randomized subspace iteration.

We can also study randomized block Lanczos methods for low-rank matrix approximation [RST09; Hal+11; MM15]. These algorithms exhibit much faster convergence than randomized subspace iteration, so they are especially valuable for matrices whose singular values decay very slowly. For these methods too, the random starting matrix is a critical ingredient. See [TW23] for a recent survey.

4. Progress on average

In the last section, we considered algorithms that start from a random initialization and drive this point toward a solution to the computational problem. Of course, there is no reason to limit the role of probability to the initialization. This section introduces another template where we make random choices *at each step* of an iterative algorithm. The methods we study share the feature that each step reduces the error (or some other measure of progress) *on average*.

> **Theme** (Progress on average). At each step of an (iterative) algorithm, make a random choice so that the expected error decreases.

In optimization, a familiar example of this template is stochastic gradient iteration [Bot10]. This is a randomized variant of gradient descent in which the gradient is replaced by an unbiased random estimate that is cheaper to compute. Just as a small step in the direction of the negative gradient reduces the value of the objective function, a step in the direction of the random gradient estimate reduces the objective value on average. Even if the individual gradient estimates are very poor, we can still guarantee convergence of the randomized algorithm.

We will study two problems in numerical linear algebra that we can tackle using randomized algorithms that make progress on average. First, we study the randomized Kaczmarz iteration for solving an overdetermined least-squares problem. Second, we describe the randomly pivoted partial Cholesky method for computing a low-rank approximation of a psd matrix.

4.1. **Randomized Kaczmarz.**
Another classic linear algebra problem, dating to the time of Gauss, is the linear least-squares problem, often used for fitting linear models to data. We will focus on the overdetermined setting where the number of observations (dramatically) exceeds the number of variables in the model. This problem arises in a range of applications, such as statistical estimation, signal processing, machine learning, and even as a subroutine in other linear algebra and optimization computations.

> **Problem** (Overdetermined least-squares). Given a tall matrix $A \in \mathbb{R}^{n \times d}$ with $n \gg d$ and a response vector $b \in \mathbb{R}^n$, find a solution to the overdetermined least-squares problem:
> $$\text{minimize}_{x \in \mathbb{R}^d} \quad \tfrac{1}{2} \|Ax - b\|_2^2.$$

For large-scale instances, it may not be desirable or even feasible to tackle the least-squares problem using direct methods, which cost $\mathcal{O}(d^2 n)$ arithmetic operations. Instead, we can seek iterative algorithms that approximate the solution at a lower computational cost. In particular, we will study methods that enforce individual equations. We will see that enforcing a *randomly chosen* equation leads to a simple, elegant algorithm with nice convergence guarantees.

4.1.1. *The Kaczmarz iteration.*
There are many iterative methods for solving least-squares problems. Let us introduce a classic algorithm, attributed to Kaczmarz [Kac37]. This method has historically been popular for computed tomography and digital signal processing applications [SV09].

Consider a tall matrix $\boldsymbol{A} \in \mathbb{R}^{n \times d}$ with $n \gg d$, and write $\boldsymbol{a}_i^*$ for the ith *row* of the matrix. Choose a response vector $\boldsymbol{b} = (b_1, \ldots, b_n)^* \in \mathbb{R}^n$. We can frame the overdetermined least-squares problem as

$$\text{minimize}_{\boldsymbol{x} \in \mathbb{R}^d} \quad \frac{1}{2} \sum_{i=1}^{n} \left(\langle \boldsymbol{a}_i, \, \boldsymbol{x} \rangle - b_i \right)^2. \tag{4.1}$$

As usual, $\langle \cdot, \, \cdot \rangle$ denotes the standard inner product on $\mathbb{R}^d$. Each term $i = 1, \ldots, n$ in the sum corresponds to the squared residual in a single equation, and we can develop algorithms that work with the individual equations.

The Kaczmarz iteration begins with an initial iterate $\boldsymbol{x}_0 \in \mathbb{R}^d$, often taken to be the zero vector. At each iteration $t = 1, 2, 3, \ldots$, we select an equation $j(t)$ and update the previous iterate $\boldsymbol{x}_{t-1}$ so this equation holds exactly:

$$\boldsymbol{x}_t = \boldsymbol{x}_{t-1} - \frac{\langle \boldsymbol{a}_{j(t)}, \, \boldsymbol{x}_{t-1} \rangle - b_{j(t)}}{\|\boldsymbol{a}_{j(t)}\|_2^2} \cdot \boldsymbol{a}_{j(t)} \quad \text{for } t = 1, 2, 3, \ldots. \tag{4.2}$$

A short calculation confirms that $\langle \boldsymbol{a}_{j(t)}, \, \boldsymbol{x}_t \rangle = b_{j(t)}$. Each iteration of the Kaczmarz method costs just $\mathcal{O}(d)$ operations, irrespective of the number n of equations. As an aside, note that the update amounts to a step in the direction of the gradient of the ith squared residual in (4.1), so we can interpret the Kaczmarz iteration as a type of gradient descent method.

To design a Kaczmarz algorithm, we need to specify a control rule $j(t)$ for picking the equation at iteration t. In the classical literature, it was most common to repeatedly cycle through the equations in the order given. Another alternative is to arrange the equations in random order and cycle through them. A third possibility is to enforce the equation that has the largest residual, although this rule can be expensive to implement. These strategies all lead to convergent algorithms, but it is challenging to understand the *rate of convergence* because it depends in a complicated way on the problem data and the control rule.

4.1.2. *The role of randomness.* To address the limitations of the classic Kaczmarz method, Strohmer & Vershynin [SV09] proposed a variant that uses a *randomized control rule*. Their idea is that we can make *progress on average* toward the solution if we select an equation at random at each step. To control the variability of the randomized algorithm, the sampling probabilities should also reflect the "importance" of each equation.

With this preface in mind, we describe a random control rule for the Kaczmarz method. At each iteration t, the random row $J(t)$ is drawn *independently* from all previous random choices $(J(1), \ldots, J(t-1))$. The distribution of the random row $J(t)$ takes the form

$$\mathbb{P}\{J(t) = i\} = \frac{\|\boldsymbol{a}_i\|_2^2}{\|\boldsymbol{A}\|_{\mathrm{F}}^2} \quad \text{for } i = 1, \ldots, n. \tag{4.3}$$

In other words, the "importance" of an equation to the least-squares problem (4.1) is proportional to its squared norm, for reasons that will become clear in the analysis. We already encountered a similar distribution in our discussion of approximate matrix multiplication (Section 2.2.3). This is an instance of *importance sampling,* a variance reduction technique that was developed in the Monte Carlo approximation literature.

Algorithm 5 lists the pseudocode for this *randomized Kaczmarz* method. Note that it takes $\mathcal{O}(dn)$ operations to compute the sampling probabilities exactly. Each iteration requires $\mathcal{O}(d)$ operations.

4.1.3. *Randomized Kaczmarz: Analysis.* In this section, we give an elegant analysis of the randomized Kaczmarz iteration due to Strohmer & Vershynin [SV09]. Following their presentation, we assume that $\boldsymbol{A}$ is injective and that the least-squares problem is consistent: $\boldsymbol{b} \in \text{range}(\boldsymbol{A})$. Under these extra hypotheses, the least-squares problem admits a unique solution $\boldsymbol{x}_\star \in \mathbb{R}^d$ that is exact: $\boldsymbol{A}\boldsymbol{x}_\star = \boldsymbol{b}$. Thus,

Algorithm 5. *Randomized Kaczmarz.*

Input: Matrix $A \in \mathbb{R}^{n \times d}$, response $b \in \mathbb{R}^n$, initial iterate $x_0 \in \mathbb{R}^d$, number T of iterations
Output: Solution estimate $x_T \in \mathbb{R}^d$

```
1  function RandKaczmarz(A, b; x_0)
2      r_i = ||a_i||_2^2 for i = 1, ..., n                    ▷ a_i* is ith row of matrix
3      p_i = r_i / ∑_{j=1}^{n} r_j                            ▷ Sampling probabilities
4      for t = 1, 2, 3, ..., T do
5          Sample row index j(t) from distribution (p_i : i = 1, ..., n)
6          x_t = x_{t-1} − (⟨a_{j(t)}, x_{t-1}⟩ − b_i) a_{j(t)} / ||a_{j(t)}||_2^2      ▷ Update iterate
```

we can measure progress of the iterates toward the solution:

$$\|x_t - x_\star\|_2^2 \quad \text{for } t = 0, 1, 2, \dots.$$

We will prove that, on average, the randomized control rule (4.3) reduces the squared norm of the error by a constant factor at each iteration.

The analysis shows that the convergence factor of the randomized Kaczmarz method depends on the *Demmel condition number* [Dem88] of the matrix:

$$\kappa_{\text{dem}} := \kappa_{\text{dem}}(A) := \frac{\|A\|_{\text{F}}}{\sigma_{\min}(A)}. \tag{4.4}$$

Here, $\sigma_{\min}(A)$ is the dth largest singular value of the matrix $A \in \mathbb{R}^{n \times d}$. For comparison, recall the standard condition number $\kappa := \sigma_{\max}/\sigma_{\min}$ for linear systems. We have the inequalities $\kappa \leq \kappa_{\text{dem}} \leq \sqrt{d}\kappa$ for every matrix with d columns. Much as the intrinsic dimension refines the notion of the rank of a psd matrix, the Demmel condition number κ_{dem} accounts for the number of energetic directions in the range of the matrix, whereas the standard condition number κ does not.

Theorem 4.1 (Randomized Kaczmarz; Strohmer & Vershynin 2009)**.** *Consider a consistent system of linear equations $Ax = b$ where A is injective, and let $x_\star$ be the (unique) solution. For each $t = 1, 2, 3, \dots$, the randomized Kaczmarz iteration (Algorithm 5) yields the error bound*

$$\mathbb{E} \|x_t - x_\star\|_2^2 \leq \left(1 - \kappa_{\text{dem}}^{-2}\right)^t \cdot \|x_0 - x_\star\|_2^2.$$

Before turning to the proof, let us discuss what this result means [SV09, Sec. 2.1]. For a tolerance $\varepsilon > 0$, we can bound the expected number of iterations that suffice to drive the error below the tolerance.

$$T \geq 2\kappa_{\text{dem}}^2 \log(1/\varepsilon) \quad \text{implies} \quad \mathbb{E} \|x_T - x_\star\|_2^2 \leq \varepsilon^2 \|x_0 - x_\star\|_2.$$

Suppose that the standard condition number κ is a constant that does not depend on the matrix dimensions. Then $\kappa_{\text{dem}}^2 = \mathcal{O}(d)$. We determine that $T = \mathcal{O}(d \log(1/\varepsilon))$ iterations suffice to obtain ε-error, relative to the initial guess. If the sampling probabilities are given, then the cost of the whole algorithm is just $\mathcal{O}(d^2 \log(1/\varepsilon))$ operations, in contrast to the $\mathcal{O}(nd^2)$ cost of a direct method. Indeed, we may not even need to enforce all of the equations!

Proof of Theorem 4.1. We begin with a short linear algebra calculation that describes the evolution of the error in the deterministic Kaczmarz iteration. Since $x_\star$ satisfies the linear equations, we have $\langle a_i, x_\star \rangle = b_i$ for each row i. For each iteration $t = 0, 1, 2, \dots$, define the error vector $e_t := x_t - x_\star$.

The Kaczmarz update (4.2) ensures that the error vector satisfies

$$
\boldsymbol{e}_t := \boldsymbol{x}_t - \boldsymbol{x}_\star = (\boldsymbol{x}_{t-1} - \boldsymbol{x}_\star) - \frac{\langle \boldsymbol{a}_{j(t)}, \, \boldsymbol{x}_{t-1} - \boldsymbol{x}_\star \rangle}{\|\boldsymbol{a}_{j(t)}\|_2^2} \cdot \boldsymbol{a}_{j(t)}
$$

$$
= \boldsymbol{e}_{t-1} - \frac{\boldsymbol{a}_{j(t)} \boldsymbol{a}_{j(t)}^*}{\|\boldsymbol{a}_j\|_2^2} \cdot \boldsymbol{e}_{t-1} =: \big(\mathbf{I} - \boldsymbol{P}_{j(t)}\big) \boldsymbol{e}_{t-1}.
$$

For each row index i, we define the orthogonal projector $\boldsymbol{P}_i := \boldsymbol{a}_i \boldsymbol{a}_i^* / \|\boldsymbol{a}_i\|_2^2$ onto the span of $\boldsymbol{a}_i$. Taking the squared norm of the error evolution equation,

$$
\|\boldsymbol{e}_t\|_2^2 = \|(\mathbf{I} - \boldsymbol{P}_{j(t)}) \boldsymbol{e}_{t-1}\|_2^2 = \big\langle \boldsymbol{e}_{t-1}, \, (\mathbf{I} - \boldsymbol{P}_{j(t)}) \boldsymbol{e}_{t-1} \big\rangle. \tag{4.5}
$$

We can bound the reduction in the error by understanding how much the projection onto the row $j(t)$ cuts down the residual. When $j(t)$ is deterministic, this argument is challenging.

Instead, we implement the randomized control rule (4.3). In this case, the projector $\boldsymbol{P}_{J(t)}$ is a random matrix. Averaging with respect to the randomness in $J(t)$,

$$
\mathbb{E}_{J(t)}[\boldsymbol{P}_{J(t)}] = \sum_{i=1}^n \boldsymbol{P}_i \cdot \mathbb{P}\{J(t) = i\} = \sum_{i=1}^n \frac{\boldsymbol{a}_i \boldsymbol{a}_i^*}{\|\boldsymbol{a}_i\|_2^2} \cdot \frac{\|\boldsymbol{a}_i\|_2^2}{\|\boldsymbol{A}\|_F^2} = \frac{1}{\|\boldsymbol{A}\|_F^2} \boldsymbol{A}^* \boldsymbol{A}.
$$

The importance sampling distribution for the row index is designed to cancel out the squared norms. As a consequence, the expected projector is proportional to the Gram matrix $\boldsymbol{A}^* \boldsymbol{A}$, so the convergence rate of the randomized Kaczmarz method will reflect the geometric properties of the input matrix $\boldsymbol{A}$.

Taking the expectation of (4.5) and exploiting linearity,

$$
\mathbb{E}_{J(t)} \|\boldsymbol{e}_t\|_2^2 = \big\langle \boldsymbol{e}_{t-1}, \, (\mathbf{I} - \mathbb{E}_{J(t)}[\boldsymbol{P}_{J(t)}]) \boldsymbol{e}_{t-1} \big\rangle = \big\langle \boldsymbol{e}_{t-1}, \, (\mathbf{I} - \|\boldsymbol{A}\|_F^{-2} \boldsymbol{A}^* \boldsymbol{A}) \boldsymbol{e}_{t-1} \big\rangle
$$

$$
\leq \lambda_1 \big(\mathbf{I} - \|\boldsymbol{A}\|_F^{-2} \boldsymbol{A}^* \boldsymbol{A}\big) \cdot \|\boldsymbol{e}_{t-1}\|_2^2 = \left(1 - \frac{\sigma_{\min}(\boldsymbol{A})^2}{\|\boldsymbol{A}\|_F^2}\right) \cdot \|\boldsymbol{e}_{t-1}\|_2^2.
$$

Recalling the definition (4.4) of the Demmel condition number, we reach

$$
\mathbb{E}_{J(t)} \|\boldsymbol{e}_t\|_2^2 \leq \big(1 - \kappa_{\mathrm{dem}}^{-2}\big) \cdot \|\boldsymbol{e}_{t-1}\|_2^2. \tag{4.6}
$$

At each iteration, the randomized Kaczmarz method makes progress on average. Finally, we take the total expectation using the tower rule and the independence of control sequence:

$$
\mathbb{E} \|\boldsymbol{e}_t\|_2^2 \leq \big(1 - \kappa_{\mathrm{dem}}^{-2}\big) \cdot \mathbb{E} \|\boldsymbol{e}_{t-1}\|_2^2.
$$

Unrolling this recurrence, we obtain the stated result. □

The proof hinges on the formula (4.6), which states that the error sequence $(\|\boldsymbol{e}_t\|_2^2 : t = 0, 1, 2, \dots)$ composes a positive supermartingale. This observation has some striking implications for the long-term behavior of the randomized Kaczmarz method [CP12]. In particular, a standard argument using the Borel–Cantelli lemma provides that

$$
(1 - \varrho)^{-t} \cdot \|\boldsymbol{e}_t\|_2^2 \to 0 \quad \text{almost surely for each } \varrho < \kappa_{\mathrm{dem}}^{-2}.
$$

We can also obtain uniform concentration inequalities for the error sequence by an application of Ville's inequality [Tro23, Thm. 26.12].

The argument also shows that the precise distribution (4.3) of the randomly chosen equation plays an important role in the performance of the algorithm. Instead, suppose that we sampled an equation uniformly at random at each step. Then we obtain an error recurrence of the form

$$
\mathbb{E}_{J(t)} \|\boldsymbol{e}_t\|_2^2 \leq \left(1 - \frac{\sigma_{\min}(\boldsymbol{A})^2}{n \cdot \max_i \|\boldsymbol{a}_i\|_2^2}\right) \cdot \|\boldsymbol{e}_{t-1}\|_2^2.
$$

Since $\|A\|_{\mathrm{F}}^2 \leq n \cdot \max_i \|a_i\|_2^2$, the weighted sampling probabilities always result in faster convergence unless the rows all share the same norm.

4.1.4. *Randomized Kaczmarz: Discussion.* The paper [SV09] of Strohmer & Vershynin generated a substantial body of follow-up work. Their result, Theorem 4.1, states that the randomized Kaczmarz method (Algorithm 5) converges for a consistent linear system with an injective matrix. Zouzias & Freris [ZF13] developed a variant of the randomized Kaczmarz algorithm that can handle inconsistent systems where the matrix is not injective. Needell & Tropp [NT14] studied a generalization of the randomized block Kaczmarz method that enforces several equations at each iteration, similar to the mini-batching strategy used in stochastic gradient methods for machine learning. Needell et al. [NSW16] also demonstrated that the randomized Kaczmarz method is a particular instance of stochastic gradient iteration.

The randomized Kaczmarz algorithm is closely related to other types of randomized algorithms for least-squares problems. Leventhal & Lewis [LL10] observed that we can design and analyze *randomized coordinate descent* algorithms using the same principles; these methods work with columns of the input matrix rather than the rows. Gower & Richtarik [GR15] studied a general class of iterative algorithms for solving linear systems, called *sketch-and-project methods*. This class includes both randomized Kaczmarz and randomized coordinate descent as special cases.

In most applications, the randomized Kaczmarz method is not a very competitive algorithm for solving least-squares problems. Nevertheless, it provides a nice illustration of the progress-on-average paradigm. The papers cited above indicate that related ideas arise in other types of randomized iterative algorithms. In the next section, we consider a second example in numerical linear algebra.

4.2. **Randomly pivoted Cholesky.** The randomized SVD algorithm is designed to produce a low-rank approximation of a (rectangular) matrix, accessed via matvecs. In some applications, however, the cost of forming the input matrix and applying it to a vector is prohibitive. Here, we consider the problem of approximating a psd matrix where we pay to access each entry.

> **Problem** (Low-rank psd approximation from entries)**.** Consider a psd matrix $A \in \mathbb{H}_n^+(\mathbb{R})$ that we access via entry evaluations: $(j, k) \mapsto a_{jk}$. The task is to produce a low-rank psd approximation of A, while limiting the number of entry evaluations.

The key example is a *kernel matrix*, which tabulates the pairwise similarities among a family of n data points. To compute an entry of a kernel matrix, we must compare two data points, so the cost of generating and storing the matrix is $\mathcal{O}(n^2)$, which is *quadratic* in the number of data points. Thus, we prefer to avoid generating the entire kernel matrix.

Kernel methods reduce data analysis problems to linear algebra problems with the kernel matrix. One common strategy for scaling kernel methods to large data sets is to replace the kernel matrix by a low-rank approximation. The goal is to produce the best approximation of the kernel matrix that we can while evaluating as few entries of the kernel matrix as possible.

In this section, we will describe a simple but powerful randomized algorithm for solving the low-rank psd approximation problem. At each step, this algorithm randomly updates the approximation so that the approximation error decreases on average. Our treatment closely follows the paper [Che+22].

4.2.1. *Column Nyström approximation.* Let $A \in \mathbb{H}_n^+(\mathbb{R})$ be a psd matrix that we access by evaluating individual entries. One way to produce an approximation after querying a small number of entries is to form the approximation from a small number of (adaptively chosen) columns.

As it happens, there is a canonical way to approximate a psd matrix from a specified set of columns. Given a list $\mathsf{S} \subseteq \{1, 2, 3, \ldots, n\}$ of column indices, we can form the *column Nyström approximation*:

$$A\langle\mathsf{S}\rangle := A(:, \mathsf{S}) A(\mathsf{S}, \mathsf{S})^{\dagger} A(\mathsf{S}, :).$$

Since A is psd, the rows $A(\mathsf{S}, :)$ are the conjugate transpose of the columns $A(:, \mathsf{S})$. We use the colon operator to indicate a row or column submatrix.

The column Nyström approximation has several remarkable properties:

1. The range of the approximation coincides with the span of the distinguished columns of the target matrix: $\mathrm{range}(A\langle\mathsf{S}\rangle) = \mathrm{range}(A(:, \mathsf{S}))$.
2. In the semidefinite order, the approximation is dominated by the target matrix: $0 \preccurlyeq A\langle\mathsf{S}\rangle \preccurlyeq A$.

Moreover, among all approximations that satisfy (1)–(2), the column Nyström approximation is maximal with respect to the semidefinite order [Ando5, Thm. 5.3].

We measure the error in the column Nyström approximation with respect to the trace norm $\|\cdot\|_*$:

$$\|A - A\langle\mathsf{S}\rangle\|_* = \mathrm{tr}(A - A\langle\mathsf{S}\rangle). \tag{4.7}$$

Owing to property (2), the residual is always psd, so the trace norm agrees with the trace. Our goal is find a set S of k columns that make the trace-norm error (4.7) as small as possible.

4.2.2. *Pivoted partial Cholesky.* We can compute a column Nyström approximation using a classic technique from numerical linear algebra: *the pivoted partial Cholesky algorithm*. This procedure incrementally forms a psd approximation of a psd matrix by eliminating one column at a time. For example, if we choose to eliminate the first column of the psd matrix A_0, the Cholesky update takes the form

$$A_0 = \left[\begin{array}{c|c} a_{11} & a_{21}^* \\ \hline a_{21} & A_{22} \end{array}\right] \quad \xrightarrow{\text{update}} \quad A_1 = \left[\begin{array}{c|c} 0 & \mathbf{0} \\ \hline \mathbf{0} & A_{22} - \frac{a_{21} a_{21}^*}{a_{11}} \end{array}\right]$$

By direct calculation, you can check that the updated matrix A_1 remains psd.

Here is a conceptual description of the pivoted Cholesky algorithm for a psd input matrix $A \in \mathbb{H}_n^+(\mathbb{R})$. Set the initial residual equal to the input matrix: $A_0 = A$. Set the initial approximation equal to the zero matrix: $\widehat{A}_0 = 0$. At each step $t = 1, 2, 3, \ldots, k$, we select a column index $s_t \in \{1, 2, 3, \ldots, n\}$, called a *pivot*. We update the approximation using the s_t column of the residual, and we eliminate the s_t column from the residual:

$$\widehat{A}_t := \widehat{A}_{t-1} + \frac{A_{t-1}(:, s_t) A_{t-1}(:, s_t)^*}{A_{t-1}(s_t, s_t)}; \tag{4.8}$$

$$A_t := A_{t-1} - \frac{A_{t-1}(:, s_t) A_{t-1}(:, s_t)^*}{A_{t-1}(s_t, s_t)}. \tag{4.9}$$

As we will see, the diagonal of the residual plays a key role. We can track it with the formula

$$\mathrm{diag}(A_t) = \mathrm{diag}(A_{t-1}) - \frac{1}{A_{t-1}(s_t, s_t)} \cdot |A_{t-1}(:, s_t)|^2. \tag{4.10}$$

The function $|\cdot|^2 : \mathbb{R}^n \to \mathbb{R}_+^n$ returns the squared magnitudes of the entries of a vector.

Fact 4.2 (Cholesky meets Nyström). *Suppose that we apply the pivoted partial Cholesky algorithm* (4.8)–(4.9) *to a psd matrix A, and we select the pivot set $\mathsf{S} = \{s_1, \ldots, s_k\}$ in any order. The resulting matrix approximation $\widehat{A}_k$ equals the Nyström approximation $A\langle\mathsf{S}\rangle$.*

4.2.3. *Cholesky: Evaluating fewer entries.* To address our computational problem, we must use an alternative formulation of the pivoted partial Cholesky algorithm that avoids unnecessary entry

evaluations. To do so, we maintain the approximation in factored form:

$$\widehat{A}_t = F_t F_t^* \quad \text{for } t = 0, 1, 2, \ldots, r.$$

The initial factor $F_0 = 0$. At each step t, we select a new column index s_t. We extract the s_t column from the input matrix and we subtract the s_t column of the previous approximation to obtain the s_t column of the residual:

$$c_t \coloneqq A(:, s_t) - \widehat{A}_{t-1}(:, s_t) = A(:, s_t) - F_{t-1} \cdot (F_{t-1}(s_t, :))^*.$$

We rescale this vector by the square-root of the diagonal entry $A_{t-1}(s_t, s_t) = c_t(s_t)$, and append the resulting vector to the factor matrix:

$$F_t \coloneqq \left[\; F_{t-1} \;\middle|\; c(s_t)^{-1/2} c_t \;\right].$$

You can check that this procedure yields the same output as the standard version of the Cholesky algorithm (4.8)–(4.9).

In each iteration, we only interact with the input matrix A by reading off the entries of the s_t column. After k iterations, we have expended $\mathcal{O}(k^2 n)$ arithmetic operations and $\mathcal{O}(kn)$ storage.

4.2.4. *Pivot selection rules.* How can we select pivots in the partial Cholesky method to achieve the best approximation? This problem is tricky because we only have access to the current approximation, and we need to pay every time we evaluate an entry of the input matrix.

One simple strategy is *uniform random pivoting*:

$$s_t \sim \text{UNIFORM}\{1, 2, 3, \ldots, n\} \quad \text{for each } t = 1, \ldots, k.$$

This approach is popular in the machine learning literature [WS01]. It is justified by the intuition that data points might represent an i.i.d. sample from a population, so one is just as good as another.

Another classic strategy is *greedy pivoting* [Hig90]:

$$s_t \in \arg\min\{A_{t-1}(s, s) : s = 1, \ldots, n\} \quad \text{for each } t = 1, \ldots, k.$$

In other words, we find the largest diagonal of the residual, and we select the associated column as the pivot. To motivate this approach, recall that the entries of a psd matrix A satisfy the inequality $|a_{ij}|^2 \le a_{ii} a_{jj}$. Therefore, the large diagonal entries of the residual point toward the columns that possibly contain large entries (but may not).

The uniform pivoting strategy and the greedy pivoting strategy are both somewhat effective, but they also have serious failure modes. The uniform random strategy is pure *exploration* without any adaptivity to the matrix. The greedy strategy is pure *exploitation*, governed entirely by the incomplete information in the largest diagonal entry.

4.2.5. *The role of randomness.* Instead, we consider a random pivoting strategy that balances exploitation and exploration. We proceed from the intuition that the diagonal entries of the residual provide hints about which columns might be significant. Since this information is imperfect, we do not want to place too much reliance on it. Therefore, at each step of the iteration, we sample a random column index in proportion to the size of the diagonal entries.

More precisely, the *randomly pivoted Cholesky* algorithm uses the following selection rule. At iteration $t = 1, 2, 3, \ldots, k$, we draw the column index s_t from the probability distribution induced by the diagonal of the residual:

$$\mathbb{P}\{s_t = j\} = \frac{A_{t-1}(j, j)}{\operatorname{tr}(A_{t-1})} \quad \text{for } j = 1, 2, 3, \ldots, n. \tag{4.11}$$

This is another example of an importance sampling procedure. We will see that this random pivot rule reduces the trace of the residual *on average*. The analysis also highlights why this sampling distribution is a natural choice.

Algorithm 6. *Randomly pivoted partial Cholesky.*

Input: Psd matrix $A \in \mathbb{H}_n^+(\mathbb{R})$; approximation rank k

Output: Columns $\{s_1, \ldots, s_k\}$; factor $F \in \mathbb{R}^{n \times k}$ of rank-k approximation $\widehat{A}_k = F F^*$

---

1 **function** RPCholesky(A; k)
2      $F = 0_{n \times k}$ ▷ Factor of initial approximation
3      $d = \mathrm{diag}(A)$ ▷ n entry evals
4   **for** $t = 1, 2, 3, \ldots, k$ **do**
5   Sample $s_t \sim d / \sum_{j=1}^n d(j)$ ▷ Sample w.r.t. diagonal of current residual
6          $c = A(:, s_t) - F \cdot (F(s_t, :))^*$ ▷ Column of residual via n entry evals
7          $F(:, t) = c / (c(s_t)^{1/2})$ ▷ Update factor of approximation
8          $d = d - |F(:, t)|^2$ ▷ Update diagonal of residual
9          $d = \max\{d, 0\}$ ▷ Improve numerical stability
10   Stop when $\sum_{j=1}^n d(j) < \eta \cdot \mathrm{tr}(A)$.   ▷ **Optional:** Halt when trace is small

---

Algorithm 6 provides pseudocode for this approach. Recall that we can track the diagonal entries of the residual inexpensively using the formula (4.10). To produce a rank-r approximation, the algorithm only reveals $(r + 1)n$ entries of the input matrix: its diagonal and the r pivot columns. We spend $\mathcal{O}(r^2 n)$ arithmetic to compute the factors of the approximation, which require $\mathcal{O}(rn)$ storage.

4.2.6. *Randomly pivoted Cholesky: Elements of the analysis.* The analysis of randomly pivoted Cholesky (Algorithm 6) is significantly more challenging than the other algorithms we have discussed. Nevertheless, it is easy to understand why it makes progress. See [Che+22, Thm. 5.1] for full details.

Suppose that we initialize randomly pivoted Cholesky with the psd matrix $A_0 = A$, and we perform one iteration to obtain a matrix A_1. Define the *expected residual map*:

$$\Phi(A) := \mathbb{E}\left[A_1 \mid A_0 = A\right].$$

This function measures the progress that we make after one step of the algorithm. A quick calculation yields a formula for the expected residual map:

$$\Phi(A) = \sum_{j=1}^n \left[A - \frac{A(:, j)A(j, :)}{A(j, j)}\right] \cdot \frac{A(j, j)}{\mathrm{tr}(A)}$$

$$= A - \frac{1}{\mathrm{tr}(A)} \sum_{j=1}^n A(:, j)A(j, :) = A - \frac{1}{\mathrm{tr}(A)} \cdot A^2.$$

We have used the definition (4.9) of the residual and the sampling distribution (4.11).

As a particular consequence, we can calculate the expected trace of the residual after one step of the algorithm:

$$\mathbb{E}\,\mathrm{tr}(A_1) = \left[1 - \frac{\mathrm{tr}(A^2)}{(\mathrm{tr}\,A)^2}\right] \cdot \mathrm{tr}(A) \leq \frac{n-1}{n} \cdot \mathrm{tr}(A).$$

In each iteration, we decrease the expected trace of the residual on average. The reduction is greatest when the eigenvalues of the residual are spiky, and the reduction is least when the eigenvalues are flat. In the latter case, the residual does not admit a good low-rank approximation, so we should not expect to make further progress. Since the residual matrix is psd, we can only reach a flat spectrum if we have already approximated the leading eigenspace of A. The analysis is difficult because it depends on how the spectrum of the matrix evolves.

The expected residual map Φ satisfies several remarkable features. With respect to the semidefinite order, the function is both matrix monotone and matrix concave. These properties allow us to obtain

bounds for the trace of the residual after k iterations:

$$\mathbb{E}\left[\operatorname{tr}(A_k) \mid A_0 = A\right] \leq \operatorname{tr}\left[\Phi^{\circ k}(A)\right].$$

Here, $\circ k$ denotes the k-fold composition. The right-hand side of this formula is unitarily invariant, so we can assume that the input matrix A is diagonal. After some further argument, using the concavity of Φ, we recognize that the worst-case matrix only has two distinct eigenvalues. At this point, we can write down a dynamical system for the eigenvalues of the residuals to understand how they evolve. See [Che+22, Sec. 5] for details.

4.2.7. *Randomly pivoted Cholesky: Error bound.* Chen et al. [Che+22] obtained a strong error bound randomly pivoted Cholesky (Algorithm 6). In this section, we state their result and offer some discussion.

In the trace norm, the best rank-r approximation of a psd matrix satisfies the relation

$$\min_{\operatorname{rank}(M) \leq r} \|A - M\|_* = \sum\nolimits_{j>r} \sigma_j(A).$$

This point follows from a generalization of the Eckart–Young theorem (3.9), obtained by Mirsky [Mir60]. After drawing k columns, randomly pivoted Cholesky produces a rank-k approximation $\widehat{A}_k$. For a tolerance $\varepsilon > 0$, we would like to know how many columns k are sufficient to compete with the error in a best rank-r approximation:

$$\mathbb{E}\|A - \widehat{A}_k\|_* \leq (1 + \varepsilon) \cdot \sum\nolimits_{j>r} \sigma_j(A). \tag{4.12}$$

We have the following statement.

Theorem 4.3 (Randomly pivoted Cholesky; Chen et al. 2022). *Consider a psd matrix $A \in \mathbb{H}_n^+(\mathbb{R})$. Fix a comparison rank r and a tolerance $\varepsilon > 0$. Randomly pivoted Cholesky (Algorithm 6) produces an approximation $\widehat{A}_k$ that attains the error bound (4.12) after selecting k columns where*

$$k \geq \frac{r}{\varepsilon} + r \cdot \log\left(\frac{1}{\varepsilon\eta}\right) \quad \text{and} \quad \eta \coloneqq \frac{1}{\operatorname{tr}(A)} \sum\nolimits_{j>r} \sigma_j(A). \tag{4.13}$$

The quantity η is the error in a best rank-r approximation of the input matrix, relative to its trace.

For the worst-case matrix, the optimal Nyström approximation requires $k \geq r/\varepsilon$ columns in order to achieve error

$$\|A - \widehat{A}\|_* \leq (1 + \varepsilon) \cdot \sum\nolimits_{j>r} \sigma_j(A).$$

Thus, the first term in (4.13) is necessary; see [GS12] or [Che+22, Sect. B.1]. The second term is proportional to the comparison rank r, up to a logarithmic factor in the tolerance ε and the relative approximation error η. Practically speaking, we cannot hope to drive the relative error below the machine precision (about 10^{-16} in double-precision arithmetic), so the excess factor is at most $\log(10^{16}) \approx 37$. Thus, randomly pivoted Cholesky not only has a favorable computational profile, but it produces approximations that are competitive with the best possible Nyström approximation.

5. Randomized dimension reduction

One approach for tackling a high-dimensional problem is to embed it into a space of lower dimension, while approximately preserving the geometry. We may hope that the solution to the lower-dimensional problem is close to the solution to the original problem. At the same time, we aspire to solve the lower-dimensional problem with fewer computational resources. In matrix computations, we can often construct an embedding that works for a fixed problem instance by using randomness.

> **Theme** (Randomized linear dimension reduction). Use a random linear map to embed a high-dimensional linear algebra problem into a lower-dimensional space, while approximately preserving the geometry.

Randomized linear dimension reduction is a particular example of the sketching paradigm for computation [Mut05]. Many treatments [Woo14; DM18] of randomized matrix computation use this idea as an organizing theme. In this section, we will introduce the method, and we will give several applications where it plays a role. Nevertheless, one of the purposes of this short course is to demonstrate that there are many ways to use probability for linear algebra computations, and it is facile to suggest that the entire subject amounts to "random sketching".

5.1. Embedding a linear subspace.
We will focus on dimension reduction for linear subspaces. This primitive has a number of distinct computational applications, and it is the most widely used sketching method in numerical linear algebra. Our presentation is inspired by the paper [NT24].

5.1.1. *The subspace embedding property.*
We begin with a definition, due to Sarlós [Sar06].

Definition 5.1 (Subspace embedding; Sarlós 2006). Let $\mathsf{L} \subseteq \mathbb{R}^n$ be a *linear* subspace with dimension d. Consider a *linear* map $\boldsymbol{\Phi} : \mathbb{R}^n \to \mathbb{R}^s$ with the property that

$$(1 - \varepsilon) \cdot \|\boldsymbol{x}\|_2 \leq \|\boldsymbol{\Phi x}\|_2 \leq (1 + \varepsilon) \cdot \|\boldsymbol{x}\|_2 \quad \text{for all } \boldsymbol{x} \in \mathsf{L}. \tag{5.1}$$

The map $\boldsymbol{\Phi}$ is called a *subspace embedding* for L with embedding dimension $s \in \mathbb{N}$ and distortion $\varepsilon > 0$.

A subspace embedding reduces the dimension of the ambient space $\mathbb{R}^n$, but it preserves the norms of all vectors in the distinguished subspace L up to a small distortion factor $(1 \pm \varepsilon)$. Typically, we assume that the subspace dimension is much smaller than the ambient dimension: $d \ll n$. Since the image $\boldsymbol{\Phi}(\mathsf{L})$ is a subspace in $\mathbb{R}^s$, the embedding dimension must exceed the subspace dimension: $s \geq d$. Still, we may hope that the embedding dimension is close to the subspace dimension and much smaller than the ambient dimension: $s \approx d \ll n$.

Before turning to constructions of subspace embeddings, we will explore some of the consequences of the embedding property (5.1). After presenting the constructions, we will describe a number of linear algebra problems that we can solve efficiently using a subspace embedding.

As an aside, you should be aware that there are other kinds of dimension reduction primitives used in linear algebra, such as affine embeddings [Woo22] and projection-preserving sketches [MM20].

5.1.2. *Metric geometry.*
You may be familiar with the concept of an embedding of metric spaces, a map from one metric space to another that approximately preserves the distances between each pair of points. We can view a subspace embedding as a metric embedding of (a subspace of) a Euclidean space into another Euclidean space.

Indeed, the subspace embedding property (5.1) guarantees that we can simultaneously preserve the distance between each pair of points in a d-dimensional subspace L:

$$(1 - \varepsilon) \cdot \|\boldsymbol{x} - \boldsymbol{y}\|_2 \leq \|\boldsymbol{\Phi x} - \boldsymbol{\Phi y}\|_2 \leq (1 + \varepsilon) \cdot \|\boldsymbol{x} - \boldsymbol{y}\|_2 \quad \text{for all } \boldsymbol{x}, \boldsymbol{y} \in \mathsf{L}.$$

This consequence follows immediately from the linearity of the embedding $\boldsymbol{\Phi}$. It is also clear that we can achieve this goal with embedding dimension $s = d$, simply by choosing a partial unitary matrix $\boldsymbol{\Phi}$ with $\text{null}(\boldsymbol{\Phi}) = \mathsf{L}^\perp$. By increasing the embedding dimension s and allowing some distortion ε, we acquire flexibility that can facilitate the construction of subspace embeddings.

Subspace embeddings also preserve the inner produces between vectors in the subspace, but this entails an additive error with a larger distortion. Indeed, a messy calculation shows that

$$|\langle \boldsymbol{x}, \ \boldsymbol{y} \rangle - \langle \boldsymbol{\Phi x}, \ \boldsymbol{\Phi y} \rangle| \leq \varepsilon' \cdot \|\boldsymbol{x}\|_2 \|\boldsymbol{y}\|_2 \quad \text{for all } \boldsymbol{x}, \boldsymbol{y} \in \mathsf{L}. \tag{5.2}$$

When $0 < \varepsilon \leq 1$, we can take the adjusted distortion $\varepsilon' \leq 10\varepsilon$. This is not necessarily optimal.

5.1.3. *Spectral interpretation.* We can always represent the distinguished subspace as the range of a matrix. This change of perspective allows us to develop a spectral interpretation of the subspace embedding property. This connection will help us construct subspace embeddings and find applications to linear algebra problems.

Proposition 5.2 (Subspace embedding: Spectral interpretation). *Suppose that* $\mathsf{L} = \mathrm{range}(\boldsymbol{U})$ *where* $\boldsymbol{U} \in \mathbb{R}^{n \times d}$ *is a matrix with orthonormal columns. For a matrix* $\boldsymbol{\Phi} \in \mathbb{R}^{s \times n}$, *the subspace embedding property* (5.1) *is equivalent with the condition*

$$1 - \varepsilon \leq \sigma_{\min}(\boldsymbol{\Phi U}) \leq \sigma_{\max}(\boldsymbol{\Phi U}) \leq 1 + \varepsilon. \tag{5.3}$$

Proof. By hypothesis, we can express the subspace in terms of the matrix:

$$\mathsf{L} = \{\boldsymbol{U y} : \boldsymbol{y} \in \mathbb{R}^d\}.$$

Since the matrix $\boldsymbol{U}$ is orthonormal, $\|\boldsymbol{U y}\|_2 = \|\boldsymbol{y}\|_2$ for each vector $\boldsymbol{y} \in \mathbb{R}^d$. Using this parameterization, we can rewrite the embedding condition (5.1) as

$$(1 - \varepsilon) \cdot \|\boldsymbol{y}\|_2 \leq \|\boldsymbol{\Phi U y}\|_2 \leq (1 + \varepsilon) \cdot \|\boldsymbol{U}\|_2 \quad \text{for all } \boldsymbol{y} \in \mathbb{R}^d. \tag{5.4}$$

Dividing through by $\|\boldsymbol{y}\|$, we obtain the equivalent condition

$$1 - \varepsilon \leq \|\boldsymbol{\Phi U z}\|_2 \leq 1 + \varepsilon \quad \text{for each } \text{unit vector } \boldsymbol{z} \in \mathbb{R}^d.$$

The variational definition of the maximum and minimum singular values implies that this system of inequalities is the same as the condition (5.3). $\square$

5.2. Random subspace embeddings.

In many applications, it is imperative to construct a subspace embedding $\boldsymbol{\Phi} : \mathbb{R}^n \to \mathbb{R}^s$ without using prior knowledge about the subspace $\mathsf{L} \subseteq \mathbb{R}^n$. These are called *oblivious* subspace embeddings. We might want to construct an oblivious embedding because it is faster to build the embedding without performing any adaptive computations on the subspace. Another reason is that the subspace may not be available when the computation is initiated. For example, in streaming linear algebra problems, we may need to maintain a summary of a subspace that evolves with time.

It requires some intelligence to construct an oblivious subspace embedding that works for an arbitrary subspace. Indeed, for any deterministically chosen matrix $\boldsymbol{\Phi} \in \mathbb{R}^{s \times n}$ with $s < n$, we can find a pernicious subspace L that defeats the subspace embedding property (5.1) by including a vector from the null space of $\boldsymbol{\Phi}$.

By drawing a subspace embedding *at random*, we can ensure that the embedding property (5.1) holds *with high probability*. This is equivalent to using a *random matrix* for the subspace embedding. In view of Proposition 5.2, we can verify the subspace embedding property by studying the singular values of the random matrix that models the interaction between the embedding and the subspace.

5.2.1. *Gaussian embeddings.* We begin with a demonstration that it is possible to construct an oblivious subspace embedding for an arbitrary subspace. To that end, fix a matrix $\boldsymbol{U} \in \mathbb{R}^{n \times d}$ with orthonormal columns. We will describe a random matrix $\boldsymbol{\Phi} \in \mathbb{R}^{s \times n}$ that serves as a subspace embedding for the range of $\boldsymbol{U}$ with exceedingly high probability.

Let us consider the most basic random matrix model:

$$\boldsymbol{\Phi} \in \mathbb{R}^{s \times n} \quad \text{with i.i.d. } \mathrm{NORMAL}(0, 1/s) \text{ entries.}$$

The variance $1/s$ of the entries is just a normalization to ensure that the matrix preserves the squared norm of a vector on average:

$$\mathbb{E} \|\boldsymbol{\Phi x}\|_2^2 = \|\boldsymbol{x}\|_2^2 \quad \text{for each fixed } \boldsymbol{x} \in \mathbb{R}^n. \tag{5.5}$$

We must show that the random matrix simultaneously preserves the norms of all vectors in range($\boldsymbol{U}$), up to a distortion factor. For this argument, we take advantage of the orthogonal invariance of a standard normal matrix. Indeed,

$$\boldsymbol{\Phi U} \in \mathbb{R}^{s \times d} \quad \text{has i.i.d. NORMAL}(0, 1/s) \text{ entries.}$$

This observation gives us access to tools for analyzing Gaussian matrices.

The subspace embedding property now follows from a classic result on the singular values of a standard normal matrix [DS01, Thm. II.13]:

$$\mathbb{P}\left\{1 - \sqrt{d/s} - t \leq \sigma_{\min}(\boldsymbol{\Phi U}) \leq \sigma_{\max}(\boldsymbol{\Phi U}) \leq 1 + \sqrt{d/s} + t\right\} \geq 1 - \mathrm{e}^{-st^2/2}.$$

Moreover, according to the Bai–Yin law [BY88], the extreme singular values converge to $1 \pm \sqrt{d/s}$ almost surely as $s, d \to \infty$ with the proportion d/s fixed.

These facts allow us to obtain a nice scaling rule for the embedding dimension. By the Bai–Yin law, we have the heuristic that

$$s \geq d/\varepsilon^2 \quad \text{yields distortion } \varepsilon. \tag{5.6}$$

Alternatively, we may take $s \geq 4d/\varepsilon^2$ to guarantee failure probability below $\mathrm{e}^{-d/2}$. In other words, the embedding dimension s only needs to be slightly larger than the subspace dimension d to achieve a modest distortion, say, $\varepsilon = 1/\sqrt{2}$. On the other hand, to obtain a small distortion ε, we must pay an intolerable factor of ε^{-2} in the embedding dimension. Thus, *you should only deploy randomized subspace embeddings in contexts where you can accept a substantial distortion.*

The analysis for Gaussian subspace embeddings is simple and elegant. Nevertheless, Gaussian matrices are often impractical because we pay $\mathcal{O}(sn)$ operations to apply the matrix to a single vector. The cost of generating and storing the matrix is also unappealing. To that end, we must consider subspace embeddings that are more computationally friendly.

5.2.2. *Randomized trigonometric transforms.* We would like to design a subspace embedding that we can apply to a vector quickly. This desideratum suggests that we should work with a linear map that admits a fast matvec operation. One familiar example is an orthogonal discrete cosine transform (DCT2) matrix, which we can apply with FFT-like algorithms. We need to inject some additional randomness to obtain the subspace embedding property for an arbitrary subspace.

Ailon & Chazelle [AC09] proposed the following type of random matrix, now called a *subsampled randomized trigonometric transform* (SRTT):

$$\boldsymbol{\Phi} := \sqrt{\frac{n}{s}}\boldsymbol{RDF} \in \mathbb{R}^{s \times n} \quad \text{where} \quad \begin{array}{l} \boldsymbol{R} \in \mathbb{R}^{s \times n} \text{ subsamples rows;} \\ \boldsymbol{D} \in \mathbb{R}^{n \times n} \text{ is random diagonal;} \\ \boldsymbol{F} \in \mathbb{R}^{n \times n} \text{ is a DCT2 matrix.} \end{array} \tag{5.7}$$

More precisely, $\boldsymbol{R}$ is a uniformly random set of s rows drawn from the identity matrix $\mathbf{I}_n$, and the random diagonal matrix $\boldsymbol{D}$ has i.i.d. UNIFORM$\{\pm 1\}$ entries. It is not hard to check that the SRTT preserves the squared norm of a vector on average, as in (5.5).

The cost of applying the SRTT to a vector is $\mathcal{O}(n \log n)$ operations using a standard fast DCT2 algorithm. Moreover, we can represent the SRTT matrix using only $s + n$ parameters. When $\log n \ll s$, we obtain a dramatic improvement over the $\mathcal{O}(sn)$ cost of storing and applying a Gaussian dimension reduction map.

What embedding dimension s does the SRTT require? The analysis [Tro11] employs tools from matrix concentration to address this question. To obtain a subspace embedding with distortion $\varepsilon = 0.6$, it suffices to take the embedding dimension

$$s \geq 4\left[\sqrt{d} + \sqrt{16 \log(dn)}\right]^2 \log d.$$

The failure probability is $\mathcal{O}(1/d)$. Note that the embedding dimension for the SRTT scales as $s \approx d \log d$, rather than the $s \approx d$ scaling for the Gaussian map. For the worst-case subspace, the extra logarithmic factor cannot be avoided if we use the construction (5.7) for the embedding map. In practice, we often follow the prescription (5.6), in spite of the possible failure modes. Regardless, the computational benefits of the SRTT are worth the increase in the size of the embedding.

5.2.3. *Sparse random matrices.* Another way to obtain a computationally efficient matvec operation is to employ a *sparse* random matrix. These matrices are simple, effective, and fast. They are widely used in practice. On the other hand, the analysis remains somewhat more qualitative than the other examples.

The following construction was proposed by Kane & Nelson [KN14]. Consider a sparse random matrix of the form

$$\mathbf{\Phi} = \begin{bmatrix} \boldsymbol{\varphi}_1 & \cdots & \boldsymbol{\varphi}_n \end{bmatrix} \in \mathbb{R}^{s \times n} \quad \text{where } \boldsymbol{\varphi}_i \in \mathbb{R}^s \text{ are i.i.d. sparse vectors.} \tag{5.8}$$

More precisely, each column $\boldsymbol{\varphi}_i$ contains exactly ζ nonzero entries, equally likely to be $\pm 1/\sqrt{\zeta}$, in uniformly random positions. It is not hard to check that the sparse map (5.8) preserves the squared norm of a vector on average, as in (5.5). We can apply this matrix to a vector in $\mathcal{O}(\zeta n)$ operations. The storage cost is at most ζn parameters. If $\zeta \ll s$, then we obtain a significant computational benefit.

Cohen [Coh16] showed that the sparse map serves as a subspace embedding with constant distortion, $\varepsilon = \text{const}$, when the embedding dimension s and the sparsity ζ satisfy

$$s \geq \text{Const} \cdot d \log d \quad \text{and} \quad \zeta \geq \text{Const} \cdot \log d.$$

The failure probability is constant. Very recently, Chenakkod et al. [Che+23] showed that a variant of the distribution (5.8) serves as a subspace embedding with constant distortion, $\varepsilon = \text{const}$, when

$$s \geq \text{Const} \cdot d \quad \text{and} \quad \zeta \geq \text{Const} \cdot \log^4 d.$$

The failure probability is again constant. In both cases, we can adjust the distortion and the failure probability, but it complicates the statement of the bounds.

The existing theoretical results are not sufficiently precise that we can use them to set algorithm parameters *a priori*. In practice [Tro+19], we may follow the prescription (5.6) and take $\zeta = 8$.

5.2.4. *Summary.* So far, we have not justified our interest in subspace embeddings. We will remedy this shortcoming in the next several sections, where we outline applications of these embeddings in numerical linear algebra.

Before we continue, let us consolidate what we have learned. For a fixed d-dimensional subspace $\mathsf{L} \subseteq \mathbb{R}^n$, we have seen that there are several ways to construct a *random* matrix $\mathbf{\Phi} \in \mathbb{R}^{s \times n}$ that serves as a subspace embedding for L. These constructions are successful with high probability. Thus, we can draw a random matrix $\mathbf{\Phi}$ and condition on the event that it is a subspace embedding for L. Then the *deterministic* subspace embedding property (5.1) is in force, and we can reason about its consequences. *Probability only arises in the construction of the embedding.*

To embed a subspace with dimension d and distortion ε, we typically choose the embedding dimension $s = d/\varepsilon^2$. For the SRTT map (5.7) and the sparse map (5.8), the cost of applying the dimension reduction matrix to a vector in $\mathbb{R}^n$ is typically about $\mathcal{O}(n \log d)$ operations. We will base operation counts on these heuristics.

5.3. **Approximate least-squares.** Let us return to the overdetermined least-squares problem first posed in Section 4.1. We will argue that subspace embeddings allow us to obtain a coarse solution to the least-squares problem efficiently. This approach was developed by Sarlós [Sar06].

Algorithm 7. *Overdetermined least-squares: Simplest sketching method.*

Input: Matrix $A \in \mathbb{R}^{n \times d}$, response $b \in \mathbb{R}^n$
Output: Solution estimate $x_{\mathrm{sk}} \in \mathbb{R}^d$

---

1 **function** SketchLeastSquares(A, b)
2   Sample subspace embedding $\Phi \in \mathbb{R}^{s \times n}$ for range($[A, b]$)   ▷ See Section 5.2
3     $C = \Phi A$ and $d = \Phi b$ ▷ Embed problem data
4     $(Q, R) = \mathrm{qr}(C)$ ▷ Thin (pivoted) QR factorization
5     $x_{\mathrm{sk}} = R^{\dagger}(Q^* d)$ ▷ Solve reduced least-squares problem

---

5.3.1. *Sketch-and-solve for least-squares.* Consider the quadratic optimization problem

$$\text{minimize}_{x \in \mathbb{R}^d} \quad \tfrac{1}{2} \|Ax - b\|_2^2 \quad \text{with } A \in \mathbb{R}^{n \times d}. \tag{5.9}$$

We focus on the case where $d \ll n$, while the matrix A is dense and unstructured. The cost of solving the problem with a direct method, such as QR factorization, is $\mathcal{O}(d^2 n)$ operations. We can obtain faster algorithms via dimension reduction. Here is the procedure:

> 1. Construct a (random, fast) subspace embedding $\Phi \in \mathbb{R}^{s \times n}$ for range($[A, b]$).
> 2. Reduce the dimension of the problem data:
>
> $$\Phi A \in \mathbb{R}^{s \times d} \quad \text{and} \quad \Phi b \in \mathbb{R}^s.$$
>
> This step is commonly referred to as *sketching*.
> 3. Find a solution $x_{\mathrm{sk}} \in \mathbb{R}^d$ to the sketched least-squares problem:
>
> $$\text{minimize}_{x \in \mathbb{R}^d} \tfrac{1}{2} \|\Phi(Ax - b)\|_2^2. \tag{5.10}$$

See Algorithm 7 for pseudocode that implements this procedure. To achieve distortion ε for the $(d + 1)$-dimensional subspace range($[A, b]$), it suffices that $s = \mathcal{O}(d/\varepsilon^2)$. The typical cost of the sketching step is $\mathcal{O}(nd \log d)$ operations. With a direct method, we can solve the sketched least-squares problem (5.10) using $\mathcal{O}(d^2 s)$ operations.

We can try to use a solution x_{sk} to the sketched least-squares problem (5.10) as a *proxy* for a solution $x_{\star}$ to the original least-squares problem (5.9). This is called the *sketch-and-solve paradigm*. The advantage is that we have reduced the operation count to $\mathcal{O}(nd \log d + d^3/\varepsilon^2)$ operations, as compared with the $\mathcal{O}(d^2 n)$ cost of a direct method for the original problem. When ε is modest and $\log d \ll d \ll n$, we obtain a significant reduction in the arithmetic cost.

5.3.2. *Sketch-and-solve: Analysis.* The analysis of Algorithm 7 is straightforward. We will compare the residual in the solution x_{sk} to the sketched problem (5.10) with the optimal residual for the original problem (5.9) attained by $x_{\star}$.

Proposition 5.3 (Approximate least-squares; Sarlós 2006). *Suppose that $A \in \mathbb{R}^{n \times d}$ is a tall matrix and $b \in \mathbb{R}^n$. Construct a subspace embedding $\Phi \in \mathbb{R}^{s \times n}$ for* range($[A, b]$) *with distortion ε. Let $x_{\star} \in \mathbb{R}^d$ be a solution to the original least-squares problem (5.9), and let $x_{\mathrm{sk}} \in \mathbb{R}^d$ be a solution to the sketched problem (5.10). Then*

$$\|Ax_{\mathrm{sk}} - b\|_2 \leq \frac{1 + \varepsilon}{1 - \varepsilon} \cdot \|Ax_{\star} - b\|_2.$$

Proof. We invoke the embedding property (5.1) twice:

$$\|\boldsymbol{A}\boldsymbol{x}_{\mathrm{sk}} - \boldsymbol{b}\|_2 \leq (1 - \varepsilon)^{-1} \cdot \|\boldsymbol{\Phi}(\boldsymbol{A}\boldsymbol{x}_{\mathrm{sk}} - \boldsymbol{b})\|_2$$

$$\leq (1 - \varepsilon)^{-1} \cdot \|\boldsymbol{\Phi}(\boldsymbol{A}\boldsymbol{x}_{\star} - \boldsymbol{b})\|_2 \leq \frac{1 + \varepsilon}{1 - \varepsilon} \cdot \|\boldsymbol{A}\boldsymbol{x}_{\star} - \boldsymbol{b}\|_2.$$

The first inequality is the lower bound in the embedding property (5.1). The second inequality holds because $\boldsymbol{x}_{\mathrm{sk}}$ is the optimal solution to the sketched least-squares problem (5.10), so $\boldsymbol{x}_{\star}$ has a larger residual. Last, we invoke the upper bound in the embedding property (5.1). $\qquad\square$

We have shown that the sketched solution $\boldsymbol{x}_{\mathrm{sk}}$ produces a residual *for the original least-squares problem* that is competitive with the optimal residual. In other words, reducing the dimension of the problem only inflates the residual by a constant factor that depends on the distortion ε of the embedding. For instance, when $\varepsilon = 1/2$, we may increase the residual by a factor of 3. As noted, it is not practical to make the distortion ε small because the computational costs increase rapidly.

When is it acceptable to increase the residual by a factor of 3 or 5 or 10? Suppose that the optimal residual is tiny, say, $\|\boldsymbol{A}\boldsymbol{x}_{\star} - \boldsymbol{b}\|_2 \leq 10^{-8}$. Then the sketch-and-solve method yields a residual $\|\boldsymbol{A}\boldsymbol{x}_{\mathrm{sk}} - \boldsymbol{b}\|_2$ that is also tiny, so the solution $\boldsymbol{x}_{\mathrm{sk}}$ to the sketched problem may be valuable to us. In the next section, we will outline an application of this idea. Nevertheless, it is important to keep in mind that a small residual *does not imply* that $\boldsymbol{x}_{\mathrm{sk}}$ approximates an optimal point $\boldsymbol{x}_{\star}$. See [GVL13, Sec. 5.3] for a discussion of the sensitivity of least-squares problems to perturbations of the data.

For contrast, *sketch-and-solve is not an appropriate strategy for data fitting problems*. In these situations, the residual in the original least-squares problem is usually large because of noise in the observations. Since the sketched solution increases the residual by a constant factor, the sketched solution may not provide any useful information about the original least-squares problem.

5.3.3. *Application: Subspace projection methods.* One attractive application of the sketch-and-solve paradigm is to accelerate subspace projection methods for solving linear systems and eigenvalue problems [NT24].

Suppose that we wish to solve a nonsingular linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ with $\boldsymbol{A} \in \mathbb{R}^{n \times n}$. We first construct a tall matrix $\boldsymbol{K} \in \mathbb{R}^{n \times d}$ with $d \ll n$ whose range contains a good approximation to the solution of the linear system. For example, range($\boldsymbol{K}$) might be the Krylov subspace spanned by vectors $(\boldsymbol{A}^t \boldsymbol{v} : t = 0, 1, 2, \ldots, T)$. The subspace projection method passes from the original linear system to the overdetermined least-squares problem

$$\mathsf{minimize}_{\boldsymbol{y} \in \mathbb{R}^d} \quad \tfrac{1}{2}\|\boldsymbol{A}\boldsymbol{K}\boldsymbol{y} - \boldsymbol{b}\|_2^2. \tag{5.11}$$

The solution $\boldsymbol{y}_{\star}$ of the least-squares problem (5.11) induces an approximate solution $\boldsymbol{x}_K = \boldsymbol{K}\boldsymbol{y}_{\star}$ to the original linear system.

Since range($\boldsymbol{K}$) captures approximate solutions to the linear system, the least-squares problem (5.11) has a tiny residual. Therefore, the subspace projection method is a good candidate for the sketch-and-solve approach. For some problems, this technique can speed up the subspace projection method by a factor of 100× or more.

5.3.4. *Iterative sketching.* We can improve the sketch-and-solve method by deploying it as part of an iterative algorithm [GR15; PW15]. Epperly has shown that this approach is forward stable [Epp23].

Suppose that we wish to solve the overdetermined least-squares problem (5.9). First, construct a dimension reduction map $\boldsymbol{\Phi} \in \mathbb{R}^{s \times n}$ for range($[\boldsymbol{A}, \boldsymbol{b}]$) with constant distortion, say, $\varepsilon = 1/\sqrt{2}$. Given an initial iterate $\boldsymbol{x}_0$, we repeatedly solve the quadratic optimization problem

$$\boldsymbol{x}_{t+1} \in \arg\min \left\{ \|(\boldsymbol{\Phi}\boldsymbol{A})\boldsymbol{y}\|_2^2 + \boldsymbol{y}^* \boldsymbol{A}^* (\boldsymbol{A}\boldsymbol{x}_t - \boldsymbol{b}) : \boldsymbol{y} \in \mathbb{R}^d \right\}.$$

Algorithm 8. *Approximate orthogonalization.*

Input: Matrix $A \in \mathbb{R}^{n \times d}$ with full column rank
Output: Well-conditioned matrix $B \in \mathbb{R}^{n \times d}$ with range(B) = range(A)

---

1 **function** RandomGramSchmidt(A, b)
2   Sample subspace embedding $\Phi \in \mathbb{R}^{s \times n}$ for range(A) ▷ See Section 5.2
3      $C = \Phi A$ ▷ Embed problem data
4      $(Q, R) = \mathsf{qr}(C)$ ▷ Thin (pivoted) QR factorization
5      $B = AR^{-1}$ ▷ Implicit representation for solution

This approach yields a sequence of iterates $(x_t : t = 0, 1, 2, 3, \dots)$ whose residuals converge exponentially fast to the optimal value. Using the sketch-and-solve solution as an initial iterate $(x_0 = x_{\mathrm{sk}})$, with about $\log(1/\alpha)$ iterations, we can achieve the error bound

$$\|Ax_t - b\|_2 \leq (1 + \alpha) \cdot \|Ax_\star - b\|_2.$$

The total cost is $\mathcal{O}(nd \log(d/\alpha) + d^3 \log d)$ arithmetic.

5.4. **Approximate orthogonalization.** As a second application of subspace embeddings, we consider the problem of approximately orthogonalizing a system of high-dimensional vectors.

> **Problem** (Approximate orthogonalization). Consider a matrix $A \in \mathbb{R}^{n \times d}$ with full column rank. The task is to find a well-conditioned matrix $B \in \mathbb{R}^{n \times d}$ with range(B) = range(A).

A direct method for orthogonalizing the columns of the matrix A requires $\mathcal{O}(nd^2)$ arithmetic. In fact, a more serious problem is that these methods require random access to the columns of the matrix, which can be prohibitive in a parallel or distributed computing environment.

5.4.1. *Randomized Gram–Schmidt.* Balabanov & Grigori [BG22] observed that we can address the approximate orthogonalization problem using a subspace embedding. They were inspired by the fact (5.2) that subspace embeddings preserve angles. Their approach can be used in conjunction with GMRES to help solve linear systems or with Rayleigh–Ritz to help solve eigenvalue problems.
    Here is the procedure, which is called *randomized Gram–Schmidt*.

> 1. Construct a (random, fast) subspace embedding $\Phi \in \mathbb{R}^{s \times n}$ for range(A).
> 2. Sketch the problem data: $\Phi A \in \mathbb{R}^{s \times d}$.
> 3. Compute a (thin, pivoted) QR factorization of the sketched data: $\Phi A = QR$.
> 4. (Implicitly) define well-conditioned $B = AR^{-1}$ with range(B) = range(A).

Algorithm 8 provides pseudocode for this method. To achieve distortion ε, it suffices to take $s = \mathcal{O}(d/\varepsilon^2)$. The sketching step typically requires $\mathcal{O}(nd \log d)$ operations. To compute the factorization, we need $\mathcal{O}(sd^2)$ arithmetic. For example, we can perform Gram–Schmidt on the columns; more stable alternatives include Householder QR [GVL13]. If we wish to form the matrix B explicitly, we must spend $\mathcal{O}(nd^2)$ operations.
    The advantage of this method is that the sketched matrix ΦA can be computed one column at a time, and it may be small enough to store in working memory. In this case, we can also orthogonalize the columns of the sketched matrix in working memory. The coefficient matrix R obtained from the sketched data still serves to *approximately* orthogonalize the columns of the original data. In contrast, if we orthogonalize the columns of the original matrix directly, then we may incur high communication costs when rearranging and combining the columns.

5.4.2. *Approximate orthogonalization: Analysis.* The analysis of the approximate orthogonalization method depends on a result of Rokhlin & Tygert [RT08]. See Section 6.1.1 for the original context.

Proposition 5.4 (Whitening; Rokhlin & Tygert 2008)**.** *Let $A \in \mathbb{R}^{n \times d}$ be a tall matrix with full column rank. Construct a subspace embedding $\Phi \in \mathbb{R}^{s \times d}$ for* range(A) *with distortion ε. Form a* QR *factorization of the sketched matrix:*

$$\Phi A = QR \quad \text{with } R \in \mathbb{R}^{d \times d}.$$

Then R has full rank, and the whitened matrix $B = AR^{-1}$ satisfies

$$\frac{1}{1 + \varepsilon} \le \sigma_{\min}(B) \le \sigma_{\max}(B) \le \frac{1}{1 - \varepsilon}.$$

Proposition 5.4 states that the whitened matrix B has the same range as the input matrix A, but the spectral condition number $\kappa(B) \le (1 + \varepsilon)/(1 - \varepsilon)$. In other words, the columns of B are roughly orthogonal.

Proof. Since Φ is a subspace embedding for the d-dimensional subspace range(A), the range of the sketched matrix ΦA also has dimension d. Thus, the $d \times d$ triangular factor R must have full rank.

By the Pythagorean theorem, we have the identities

$$\|Ry\|_2 = \|QRy\|_2 = \|\Phi Ay\|_2 \quad \text{for all } y \in \mathbb{R}^d.$$

By the subspace embedding property (5.1),

$$(1 - \varepsilon) \cdot \|Ay\| \le \|Ry\|_2 \le (1 + \varepsilon) \cdot \|Ay\|_2 \quad \text{for all } y \in \mathbb{R}^d.$$

Make the change of variables $y = R^{-1}x$ to obtain

$$(1 - \varepsilon) \cdot \|AR^{-1}x\|_2 \le \|x\|_2 \le (1 + \varepsilon) \cdot \|AR^{-1}x\|_2 \quad \text{for all } x \in \mathbb{R}^d.$$

Divide by $\|x\|_2$ and use the variational definition of singular values to reach the stated bounds. □

5.5. **Approximate null space.** As a third application, we show that subspace embeddings allow us to compute a trailing right singular subspace of a tall matrix quickly. Our presentation is inspired by the papers [GPW12; PN23].

> **Problem** (Approximate null space)**.** Consider a tall matrix $A \in \mathbb{R}^{n \times d}$. The task is to find an orthonormal matrix $W \in \mathbb{R}^{d \times k}$ whose range aligns with the k trailing right singular vectors of A.

The approximate null space problem supports a number of important applications. It is the core computation required for total least-squares [GL80], which is also known as errors-in-variables regression in statistics. In signal processing, some spectrum estimation algorithms, such as MUSIC [Sch86], require an approximate null space. In approximation theory, the AAA algorithm [NST18] for rational approximation also requires minimum singular vectors of tall matrices.

5.5.1. *Null space from approximate right singular subspaces.* Following a well-oiled track, we apply a subspace embedding to the input matrix and compute its SVD. Then we extract the trailing right singular vectors to solve the approximate null space problem. The reason this approach works is that the SVD of the sketched matrix retains a lot of geometric information from the input matrix, much like the QR factorization of the sketched matrix. This idea is due to Gilbert et al. [GPW12].

> 1. Construct a (random, fast) subspace embedding $\Phi \in \mathbb{R}^{s \times n}$ for range(A).
> 2. Sketch the problem data: $\Phi A \in \mathbb{R}^{s \times d}$.
> 3. Compute a full SVD of the sketched matrix: $\Phi A = U\Sigma V^*$.
> 4. Report the trailing right singular vectors $W = V(:, (d - k + 1) : d) \in \mathbb{R}^{d \times k}$.

Algorithm 9. *Approximate null space.*

Input: Matrix $A \in \mathbb{R}^{n \times d}$, dimension k for approximate null space
Output: Orthonormal matrix $W \in \mathbb{R}^{d \times k}$ whose range spans approximate null space

1 **function** ApproxNullSpace(A, k)
2   Sample subspace embedding $\Phi \in \mathbb{R}^{s \times n}$ for range(A)   ▷ See Section 5.2
3     $C = \Phi A$ ▷ Embed problem data
4     $(U, \Sigma, V^*) = \mathsf{svd}(C)$ ▷ Economy SVD
5     $W = V(:, (d - k + 1) : d)$ ▷ Approximate k-dimensional null space

---

Pseudocode for this procedure appears in Algorithm 9. As usual, we choose the embedding dimension $s = \mathcal{O}(d/\varepsilon^2)$ to achieve distortion ε. The sketching step typically takes $\mathcal{O}(nd \log d)$ operations, and the SVD step takes $\mathcal{O}(sd^2)$ operations. The total is $\mathcal{O}(nd \log d + d^3/\varepsilon^2)$. For comparison, a full SVD of the input matrix requires $\mathcal{O}(d^2 n)$ arithmetic.

5.5.2. *Approximate null space: Analysis.* As usual, let $A \in \mathbb{R}^{n \times d}$ be a tall matrix, perhaps rank deficient. Here is a variational formulation of the null space problem:

$$\mathsf{minimize}_{V \in \mathbb{R}^{d \times k}} \quad \|AV\|_{\mathrm{F}}^2 \quad \text{subject to} \quad V^*V = I_k. \tag{5.12}$$

The solution to this problem is a subspace spanned by k trailing right singular vectors, and the optimal value is the energy in the trailing k singular values:

$$\min_{V^*V = I_k} \|AV\|_{\mathrm{F}}^2 = \sum_{j > d - k} \sigma_j^2(A).$$

If the k smallest singular values are very small (or zero), then a subspace that carries a similar amount of energy may serve as an approximate null space.

Using the subspace embedding property, we can easily verify that the sketching method leads to a good solution to the approximate null space problem [PN23].

Proposition 5.5 (Approximate null space; Park & Nakatsukasa 2023). *Let $A \in \mathbb{R}^{n \times d}$ be a tall matrix, and let $\Phi \in \mathbb{R}^{s \times d}$ be a subspace embedding for range(A) with distortion ε. Algorithm 9 produces an approximate null space $W \in \mathbb{R}^{d \times k}$ that is competitive with the best k-dimensional null space:*

$$\|AW\|_{\mathrm{F}}^2 \leq \frac{1 + \varepsilon}{1 - \varepsilon} \cdot \min_{V^*V = I_k} \|AV\|_{\mathrm{F}}^2.$$

In particular, if $AV = 0$ for some k-dimensional subspace V, then $AW = 0$.

Proof. Fix an orthonormal matrix $V_\star \in \mathbb{R}^{d \times k}$ that solves the null space problem (5.12). Since Φ is a subspace embedding for range(A),

$$\|AW\|_{\mathrm{F}}^2 \leq \frac{1}{1 - \varepsilon} \cdot \|\Phi A W\|_{\mathrm{F}}^2 \leq \frac{1}{1 - \varepsilon} \cdot \|\Phi A V_\star\|_{\mathrm{F}}^2 \leq \frac{1 + \varepsilon}{1 - \varepsilon} \cdot \|A V_\star\|_{\mathrm{F}}^2.$$

The first inequality is the lower subspace embedding property (5.1), applied to each column of the matrix W. The second inequality holds because W solves the null space problem (5.12) for the sketched matrix ΦA, so the matrix $V_\star$ must increase the objective. Last, we apply the upper bound from the subspace embedding property. $\square$

5.5.3. *Approximate SVD: Analysis.* In fact, we can say quite a lot more about the SVD of the sketched matrix that arises in the approximate null space algorithm. This result is essentially due to Gilbert et al. [GPW12]. The proof is adapted from Park & Nakatsukasa [PN23].

Proposition 5.6 (Approximate singular values; Gilbert et al. 2012). *Let $A \in \mathbb{R}^{n \times d}$ be a tall matrix, and let $\Phi \in \mathbb{R}^{s \times d}$ be a subspace embedding for range(A) with distortion ε. The singular values of the sketched*

matrix $\boldsymbol{\Phi A}$ *satisfy*

$$(1 - \varepsilon) \cdot \sigma_i(\boldsymbol{A}) \leq \sigma_i(\boldsymbol{\Phi A}) \leq (1 + \varepsilon) \cdot \sigma_i(\boldsymbol{A}) \quad \textit{for } i = 1, \ldots, d.$$

Proof. Let $\boldsymbol{A} = \boldsymbol{U\Sigma V}^*$ be an SVD of the input matrix. Then $\boldsymbol{\Phi}$ is a subspace embedding for range$(\boldsymbol{U})$. For each index $i = 1, \ldots, d$, by rotational invariance of singular values,

$$\sigma_i(\boldsymbol{\Phi A}) = \sigma_i(\boldsymbol{\Phi}(\boldsymbol{U\Sigma V}^*)) = \sigma_i((\boldsymbol{\Phi U})\boldsymbol{\Sigma})$$

By Ostrowski's relative perturbation theorem [Ost59],

$$\sigma_d(\boldsymbol{\Phi U}) \cdot \sigma_i(\boldsymbol{\Sigma}) \leq \sigma_i(\boldsymbol{\Phi A}) \leq \sigma_1(\boldsymbol{\Phi U}) \cdot \sigma_i(\boldsymbol{\Sigma}).$$

By the subspace embedding property (5.1),

$$(1 - \varepsilon) \cdot \sigma_i(\boldsymbol{A}) \leq \sigma_i(\boldsymbol{\Phi A}) \leq (1 + \varepsilon) \cdot \sigma_i(\boldsymbol{A}).$$

This is the advertised result.   □

It is also possible to argue that the right singular subspaces of the sketched matrix $\boldsymbol{\Phi A}$ align with the right singular subspace of the original matrix $\boldsymbol{A}$, provided that there are gaps between the singular values. These results are more technical, so we refer to [GPW12; PN23] for the details.

6. More themes

We have studied several templates for designing randomized algorithms for matrix computations. Reviewing the literature in this field, it is striking how many other strategies fall outside these categories. In this section, we give a brief introduction to some other algorithmic approaches that have a probabilistic component: randomized approximation for preconditioning, placing problem instances in general position, and smoothed analysis of problem perturbations.

6.1. Randomized preconditioning.
Many linear algebra problems have associated condition numbers that reflect the sensitivity of problem solutions and may impact the difficulty of solving the problem numerically. For example, the linear system condition number describes the stability of the solution to a linear system [GVL13, Sec. 2.6.2]. For a positive-definite linear system, the cost of the conjugate gradient algorithm can be bounded in terms of the linear system condition number [She94, Sec. 10].

The preconditioning strategy replaces a badly conditioned computational problem with an equivalent problem that has a smaller condition number. For linear systems and least-squares, preconditioning schemes typically multiply the input matrix with another matrix that is both easy to invert and that also reduces the condition number.

We have already encountered many algorithms for inexpensively computing a structured approximation of a matrix. Some of these approximations can be used as preconditioners for solving linear algebra problems involving the matrix.

> **Theme** (Randomized preconditioning). Given an input matrix, use a randomized algorithm to form a structured approximation that serves as a preconditioner.

In this section, we summarize a few examples where randomized methods lead to preconditioners for least-squares problems and regularized linear systems. The key benefit of randomized algorithms for preconditioning is that they can provide excellent guarantees while operating more efficiently than deterministic techniques.

6.1.1. Example: Overdetermined least-squares.
Let $\boldsymbol{A} \in \mathbb{R}^{n \times d}$ be a tall matrix with full rank, and let $\boldsymbol{b} \in \mathbb{R}^n$ be a response vector. Recall the overdetermined least-squares problem

$$\text{minimize}_{\boldsymbol{x} \in \mathbb{R}^d} \quad \tfrac{1}{2}\|\boldsymbol{Ax} - \boldsymbol{b}\|_2^2.$$

Rokhlin & Tygert [RT08] observed that we can use sketching to design a preconditioner.

As usual, let $\mathbf{\Phi} \in \mathbb{R}^{s \times n}$ be a (random, fast) subspace embedding for range($\boldsymbol{A}$). We can form a QR factorization of the sketched matrix:

$$\mathbf{\Phi} \boldsymbol{A} = \boldsymbol{Q} \boldsymbol{R}.$$

The cost of these steps can be much smaller than solving the original least-squares problem. Proposition 5.4 states that $\boldsymbol{A} \boldsymbol{R}^{-1}$ is a very well-conditioned matrix with the same range as $\boldsymbol{A}$. We can solve a triangular linear systems with $\boldsymbol{R}$ efficiently, so it serves as an excellent preconditioner.

Thus, we pass to the equivalent least-squares problem

$$\mathsf{minimize}_{\boldsymbol{y} \in \mathbb{R}^d} \; \tfrac{1}{2} \|(\boldsymbol{A} \boldsymbol{R}^{-1}) \boldsymbol{y} - \boldsymbol{b}\|_2^2 \quad \text{where} \quad \boldsymbol{x} = \boldsymbol{R}^{-1} \boldsymbol{y}.$$

We can solve this problem efficiently using the LSQR algorithm [GVL13, Sec. 11.4.2]. Assuming exact arithmetic, the Rokhlin & Tygert algorithm can provably solve a least-squares problem to high accuracy faster than any previously known algorithm. On the other hand, there remain unresolved questions about the numerical stability of this approach [Mei+23].

6.1.2. *Example: Randomly pivoted Cholesky.* In a similar vein, we can use randomized Nyström approximations to design a preconditioner for a linear system with a psd matrix [FTU23]. This section outlines a related method that uses randomly pivoted Cholesky to accelerate kernel computations [Día+23].

Let $\boldsymbol{A} \in \mathbb{H}_n^+(\mathbb{R})$ be a psd matrix where we pay for entry evaluations, such as a kernel matrix. Suppose that we wish to solve the regularized linear system

$$(\boldsymbol{A} + \mu \mathbf{I}) \boldsymbol{x} = \boldsymbol{b} \quad \text{for } \mu > 0. \tag{6.1}$$

This problem often arises in kernel methods, where it is known as *kernel ridge regression*.

In Section 4.2, we saw that the randomly pivoted Cholesky algorithm can be used to compute a near-optimal psd approximation of the matrix $\boldsymbol{A}$. More precisely, we can obtain a rank-k psd matrix $\widehat{\boldsymbol{A}}_k$, represented as an eigenvalue decomposition. The cost of producing the approximation is $\mathcal{O}(kn)$ entry evaluations and $\mathcal{O}(k^2 n)$ arithmetic.

This approximation yields a preconditioner [Día+23] for solving the regularized system:

$$(\widehat{\boldsymbol{A}}_k + \mu \mathbf{I})^{-1} (\boldsymbol{A} + \mu \mathbf{I}) \boldsymbol{x} = (\widehat{\boldsymbol{A}}_k + \mu \mathbf{I})^{-1} \boldsymbol{b}.$$

We can solve linear systems with the preconditioner efficiently, after just $\mathcal{O}(kn)$ operations. Therefore, we can apply preconditioned conjugate gradient [GVL13, Sec. 11.5.2] to quickly solve the regularized linear system (6.1). See [Día+23, Thm. 2.2] for an analysis and details on the runtime.

Other authors [ACW17] have proposed preconditioners based on random features approximations. The concept is similar but this approach is less competitive in practice.

6.1.3. *Example: Sparse Cholesky for graph Laplacians.* We conclude with a spectacular example of the randomized preconditioning paradigm, due to Kyng & Sachdeva [KS16]. See [Tro19] for another treatment of this result. The recent paper [GKS23] describes an implementation and empirical evaluation of a closely related algorithm.

Suppose that $\boldsymbol{A} \in \mathbb{H}_n^+(\mathbb{R})$ is the graph Laplacian of a combinatorial graph with n vertices and m edges. Many applications in data analysis and computer algorithms require the solution to linear systems in this Laplacian matrix:

$$\boldsymbol{A} \boldsymbol{x} = \boldsymbol{b} \quad \text{where} \quad \mathbf{1}^* \boldsymbol{b} = 0 \quad \text{and} \quad \mathbf{1}^* \boldsymbol{x} = 0.$$

We write $\mathbf{1} \in \mathbb{R}^n$ for the all-ones vector. Nominally, this problem costs $\mathcal{O}(n^3)$ operations to solve with a direct method. Classical iterative methods, such as algebraic multigrid, may not be reliable for complicated graphs.

Kyng & Sachdeva devised a randomized algorithm that produces an approximate Cholesky factorization of the Laplacian matrix:

$$0.5\,\boldsymbol{A} \preccurlyeq \boldsymbol{L}\boldsymbol{L}^* \preccurlyeq 1.5\,\boldsymbol{A},$$

where $\preccurlyeq$ denotes the semidefinite order. Their algorithm is related to the classic full Cholesky decomposition algorithm, but it repeatedly uses matrix Monte Carlo to sparsify the rank-one correction. Their method produces a lower triangular factor $\boldsymbol{L}$ with just $\mathcal{O}(m \log n)$ nonzero entries after $\mathcal{O}(m \log^3 n)$ arithmetic operations.

The approximate Cholesky factor can be used as a preconditioner for conjugate gradient to solve the linear system:

$$(\boldsymbol{L}^\dagger \boldsymbol{A} \boldsymbol{L}^{*\dagger})\boldsymbol{y} = \boldsymbol{L}^\dagger \boldsymbol{b} \quad \text{where} \quad \boldsymbol{x} = \boldsymbol{L}^{*\dagger}\boldsymbol{y}.$$

The dagger † denotes the pseudoinverse. Indeed, we can solve the triangular system in $\boldsymbol{L}$ with just $\mathcal{O}(m \log n)$ operations, and it reduces the condition number of the problem to a constant. Therefore, we can find a solution with ε relative error after about $\log(1/\varepsilon)$ iterations.

The total operation count for the approximate factorization and solving the linear system is

$$\mathcal{O}(m \log^2(n)(\log(n) + \log(1/\varepsilon))).$$

Recall that the direct method may cost fully $\mathcal{O}(n^3)$ operations. For a sparse graph, $m \approx n$, and we achieve a factor n^2 improvement over the direct method. For a dense graph, $m \approx n^2$, and we still achieve a factor n improvement over the direct method. This is a remarkable outcome.

6.2. **General position.** Many deterministic algorithms exhibit failure modes that can arise from unfortunate choices of the problem instance. For example, Gaussian elimination can encounter zero pivots or very small pivots that render it numerically unstable, but experience suggests that "most" input matrices do not cause problems.

One way to deal with the possibility of a challenging problem instance is to randomly transform the instance into an equivalent instance that avoids the failure modes with high probability. In other words, we can try to place the instance in *general position*. We are using this term loosely, as the meaning depends on the computational problem and the algorithm.

> **Theme** (General position). Randomly transform a problem instance to an equivalent instance where the computational method is likely to succeed.

We will summarize two situations where this is possible. First, we discuss how random rotations of a linear system can ensure that Gaussian elimination succeeds. Second, we show how to simultaneously diagonalize a pair of matrices by combining them randomly.

6.2.1. *Example: Random rotations for Gaussian elimination.* In this section, we present one of the earliest randomized linear algebra methods, following the original report [Par95] by D. Stott Parker.

Suppose we wish to solve the (square, nonsingular) linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ where $\boldsymbol{A} \in \mathbb{R}^{n \times n}$. To address this problem, the classic algorithm starts by applying pivoted Gaussian elimination to the matrix $\boldsymbol{A}$ to obtain a pivoted LU factorization. Pivoting is the process of selecting which row to eliminate at each step. For a general matrix, pivoting may be necessary because we cannot eliminate a row that contains a zero in the diagonal position. Unfortunately, the need to pivot conflicts with other goals, such as blocking computations or minimizing communication costs.

Parker observed that the *degenerate* matrices where zero pivots arise compose a set with Lebesgue measure zero. He proposed that we might avoid these degenerate examples by transforming the input matrix. Let $\boldsymbol{U}, \boldsymbol{V} \in \mathbb{M}_n(\mathbb{R})$ be orthogonal matrices, drawn uniformly at random. We can pass from the original linear system to the equivalent problem

$$(\boldsymbol{U}\boldsymbol{A}\boldsymbol{V})\boldsymbol{y} = \boldsymbol{U}\boldsymbol{b} \quad \text{and} \quad \boldsymbol{x} = \boldsymbol{V}\boldsymbol{y}.$$

Parker proved that the transformed matrix $\boldsymbol{A}_{\mathrm{rot}} = \boldsymbol{U}\boldsymbol{A}\boldsymbol{V}$ is more favorable for Gaussian elimination. *With probability one, we can perform Gaussian elimination without pivoting on $\boldsymbol{A}_{\mathrm{rot}}$ and never encounter a zero diagonal entry.* We have placed the problem in general position.

What is the cost of this approach? With random orthogonal matrices, the transformation requires $\mathcal{O}(n^3)$ operations for two matrix–matrix multiplications. Recall that Gaussian elimination (with or without pivoting) demands $\mathcal{O}(n^3)$ operations, so the preprocessing does not increase the asymptotic cost of the algorithm. Meanwhile, by avoiding pivoting in the Gaussian elimination step, we can design implementations that exploit blocking and parallelism. Variants of Parker's proposal [Bab+13] have proven effective in practice.

Related ideas appear in work of Demmel et al. [DDH07]. They propose the "randomized URV algorithm" for rank-revealing factorization, and they establish more precise bounds on the stability of the resulting numerical algorithms.

6.2.2. *Example: Randomized joint diagonalization.* Consider two symmetric matrices $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{H}_n(\mathbb{R})$ that commute with each other: $\boldsymbol{AB} = \boldsymbol{BA}$. As you know, commuting symmetric matrices are simultaneously diagonalizable. In other words, there always exists an orthogonal matrix $\boldsymbol{Q} \in \mathbb{M}_n(\mathbb{R})$ with the property that $\boldsymbol{Q}^*\boldsymbol{A}\boldsymbol{Q}$ and $\boldsymbol{Q}^*\boldsymbol{B}\boldsymbol{Q}$ are both diagonal matrices. Our goal is to compute such an orthogonal matrix.

This task is delicate. It is *not true* that an orthogonal matrix that diagonalizes $\boldsymbol{A}$ will also diagonalize $\boldsymbol{B}$, so we have to understand something about how the matrices interact. Moreover, in applications, the matrices may not commute exactly because they contain numerical errors or derive from noisy data.

He & Kressner [HK23] have studied an elegant strategy that can address both these challenges. Form a *random linear combination* of the two input matrices:

$$\boldsymbol{C} = \gamma_1 \boldsymbol{A} + \gamma_2 \boldsymbol{B} \quad \text{where } \gamma_1, \gamma_2 \sim \text{NORMAL}(0, 1) \text{ i.i.d.}$$

In other words, $\boldsymbol{C}$ is a *generic point* in the plane spanned by the two matrices $\boldsymbol{A}$ and $\boldsymbol{B}$. If the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ commute with each other, then $\boldsymbol{C}$ clearly commutes with both $\boldsymbol{A}$ and $\boldsymbol{B}$. To solve the simultaneous diagonalization problem, we can simply extract the orthogonal matrix $\boldsymbol{Q}$ from an eigenvalue decomposition $\boldsymbol{C} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^*$.

He & Kressner analyzed the case where the matrices commute exactly, as well as the situation where they only commute approximately. *If $\boldsymbol{A}$ and $\boldsymbol{B}$ commute, then $\boldsymbol{Q}$ simultaneously diagonalizes the pair $(\boldsymbol{A}, \boldsymbol{B})$ with probability one.* Moreover, if $\boldsymbol{A}$ and $\boldsymbol{B}$ nearly commute, then $\boldsymbol{C}$ still produces an orthogonal matrix $\boldsymbol{Q}$ that approximately diagonalizes the pair $(\boldsymbol{A}, \boldsymbol{B})$.

6.3. **Smoothed analysis.** As we have discussed, deterministic algorithms can exhibit failure modes for malign problem instances. For some algorithms, the challenging examples are unusual in the sense that most nearby problem instances are easier to solve. Spielman & Teng [ST02] developed this perspective, called *smoothed analysis*, and they showed that it sheds light on the performance of several basic algorithms in linear algebra and optimization. Their insight immediately suggests a computational strategy. By randomly perturbing a hard problem instance, we can find a nearby instance where the algorithm is likely to succeed.

Theme (Smoothed analysis). Apply a random perturbation to a problem instance to obtain a nearby instance that is more computationally favorable.

Conceptually, this method is distinct from the preconditioning strategy or the general position strategy because it results in an inequivalent problem instance that has a different solution. While algorithms based on smoothed analysis often produce a high-quality solution to a nearby problem instance, the sleight of hand will discomfit many numerical analysts.

In this section, we treat two dramatically different applications of this idea: Gaussian elimination and nonsymmetric eigenvalue problems.

6.3.1. *Example: Gaussian elimination.* As before, we wish to apply Gaussian elimination to the matrix $A \in \mathbb{M}_n(\mathbb{R})$. Parker's method of random rotation (Section 6.2.1) avoids zero diagonal entries, but it does not necessarily control the *growth* of the diagonal entries—which is the main source of instability in the algorithm.

Sankar et al. [SST06] observed that Gaussian elimination admits a smoothed anlaysis. For a parameter $\sigma > 0$, suppose that we *perturb* the input matrix:

$$A_\sigma := A + \sigma G \quad \text{where } G \in \mathbb{M}_n(\mathbb{R}) \text{ has i.i.d. } \textsc{normal}(0, 1) \text{ entries.}$$

With high probability, Gaussian elimination on the matrix A_σ never encounters a pivot that is larger than a fixed polynomial in the dimension n and the reciprocal σ^{-1} of the perturbation. This property ensures that it is safe to apply Gaussian elimination to the instance A_σ.

As a consequence, we could simply run Gaussian elimination on the perturbed matrix A_σ to obtain an approximate LU factorization of the original matrix A. This result demonstrates that we can sometimes avoid numerical problems if we are willing to make a modest change in the problem data.

6.3.2. *Example: Pseudospectral shattering.* Consider a *complex* matrix $A \in \mathbb{M}_n(\mathbb{C})$ that is diagonalizable, but not necessarily normal. The eigenvalue problem asks us to compute a full eigenvalue decomposition:

$$A = VDV^{-1} \quad \text{where } D \in \mathbb{M}_n(\mathbb{C}) \text{ is diagonal and } V \in \mathbb{M}_n(\mathbb{C}) \text{ is nonsingular}$$

This problem is challenging when the eigenvalues of the matrix are close together or when the matrix V of eigenvectors is badly conditioned, because the computation is very sensitive to small perturbations of the data. The matrix computation literature describes reliable algorithms (based on QR iteration with shifts), but there was no rigorous analysis to justify the use of these methods—until recently.

Banks et al. [Ban+22] showed that smoothed analysis offers a strategy for addressing these difficulties. They considered a *complex* Gaussian perturbation of the matrix on the scale $\sigma > 0$:

$$A_\sigma := A + \sigma G \quad \text{where } G \in \mathbb{M}_n(\mathbb{R}) \text{ has i.i.d. } \textsc{normal}_\mathbb{C}(0, 1) \text{ entries.}$$

They proved that the matrix A_σ remains diagonalizable, the eigenvalues are well-separated, and the eigenvectors become well-conditioned. More precisely, they demonstrate that the pseudospectrum of A_σ has n isolated components on a scale that is polynomial in n^{-1} and σ. See Figure 6.1 for an illustration.

As a consequence, Banks et al. [Ban+22] were able to prove that the nonsymmetric eigenvalue problem submits to algorithms based on spectral divide-and-conquer techniques, which operate by recursively dividing the set of eigenvalues into two comparably sized groups. Gaps between eigenvalues, introduced by the perturbation, are essential for the success of the procedure. Later, also using smoothed analysis, Banks et al. [BGVS23] proved that a variant of shifted QR iteration also succeeds. This work finally confirms that there are backward-stable algorithms that can solve the nonsymmetric eigenvalue problem in $\mathcal{O}(n^3)$ time, modulo polynomial factors in $\log(n)$. We can quickly and reliably produce a very accurate solution to an eigenvalue problem near the target instance.

6.4. **Conclusions.** Our goal has been to isolate the probabilistic insights that play a role in the design of randomized algorithms for matrix computations. We have organized the methods thematically to find similarities among algorithms that might seem different in concept:

1. Monte Carlo approximation
2. Random initialization
3. Progress on average
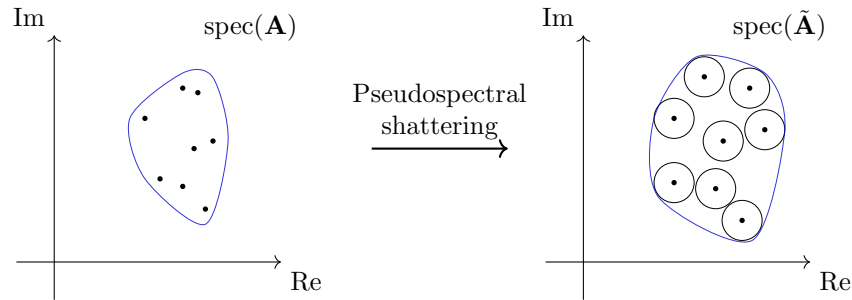4. Randomized dimension reduction

Figure 6.1. Pseudospectral shattering. After perturbation, the eigenvalues of the matrix are well-separated with high probability.

5. Randomized preconditioning
6. General position
7. Smoothed analysis

This list does not pretend to be comprehensive, and the literature contains other strategies that fall outside of this taxonomy. Undoubtedly, there are more techniques waiting to be discovered!

Acknowledgments

References

[AM01]   D. Achlioptas and F. McSherry. "Fast computation of low rank matrix approximations". In: *Proc. 33rd Ann. ACM Symp. Theory of Computing*. ACM, New York, 2001, pages 611–618. DOI: 10.1145/380752.380858.

[AC09]   N. Ailon and B. Chazelle. "The fast Johnson–Lindenstrauss transform and approximate nearest neighbors". *SIAM J. Comput.* 39.1 (2009), pages 302–322. DOI: 10.1137/060673096.

[AHD18]   M. Al Hasan and V. S. Dave. "Triangle counting in large networks: a review". *WIREs Data Mining and Knowledge Discovery* 8.2 (2018), e1226. DOI: https://doi.org/10.1002/widm.1226.

[Ando5]   T. Ando. "Schur complements and matrix inequalities: Operator-theoretic approach". In: *The Schur complement and its applications*. Springer-Verlag, New York, 2005, pages 137–162. DOI: 10.1007/b105056.

[ACW17]   H. Avron, K. L. Clarkson, and D. P. Woodruff. "Faster kernel ridge regression using sketching and preconditioning". *SIAM J. Matrix Anal. Appl.* 38.4 (2017), pages 1116–1138. DOI: 10.1137/16M1105396.

[AT11]   H. Avron and S. Toledo. "Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix". *J. ACM* 58.2 (2011). DOI: 10.1145/1944345.1944349.

[Bab+13] M. Baboulin et al. "Accelerating linear system solutions using randomization techniques". *ACM Trans. Math. Software* 39.2 (Feb. 2013), 8:1–8:13. DOI: 10.1145/2427023.2427025.

[BY88]   Z. D. Bai and Y. Q. Yin. "Necessary and sufficient conditions for almost sure convergence of the largest eigenvalue of a Wigner matrix". *Ann. Probab.* 16.4 (1988), pages 1729–1741. DOI: 10.1214/aop/1176991594.

[BG22]   O. Balabanov and L. Grigori. "Randomized Gram–Schmidt process with application to GMRES". *SIAM J. Sci. Comput.* 44.3 (2022), A1450–A1474. DOI: 10.1137/20M138870X.

[BGVS23] J. Banks, J. Garza-Vargas, and N. Srivastava. "Global convergence of Hessenberg shifted QR I: Exact arithmetic". *Found. Comput. Math.* (2023). Accepted for publication. arXiv: 2111.07976 [math.NA].

[Ban+22]   J. Banks et al. "Pseudospectral shattering, the sign function, and diagonalization in nearly matrix multiplication time". *Found. Comput. Math.* 23.6 (2022), pages 1959–2047. DOI: 10.1007/s10208-022-09577-5.

[Bot10]   L. Bottou. "Large-scale machine learning with stochastic gradient descent". In: *Proc. COMPSTAT'2010*. Physica-Verlag HD, 2010, pages 177–186.

[CH23a]   T. Chen and E. Hallman. "Krylov-aware stochastic trace estimation". *SIAM J. Matrix Anal. Appl.* 44.3 (Sept. 2023), pages 1218–1244. DOI: 10.1137/22M1494257.

[CP12]   X. Chen and A. M. Powell. "Almost sure convergence of the Kaczmarz algorithm with random measurements". *J. Fourier Anal. Appl.* 18.6 (2012), pages 1195–1214. DOI: 10.1007/s00041-012-9237-2.

[Che+22]   Y. Chen et al. "Randomly pivoted Cholesky: Practical approximation of a kernel matrix with few entry evaluations". Available from arXiv. 2022. arXiv: 2207.06503.

[Che+23]   S. Chenakkod et al. "Optimal embedding dimension for sparse subspace embeddings". Available from arXiv. 2023. arXiv: 2311.10680 [cs.DS].

[Coh16]   M. B. Cohen. "Nearly tight oblivious subspace embeddings by trace inequalities". In: *Proc. 27th Ann. ACM–SIAM Symp. Discrete Algorithms*. ACM, New York, 2016, pages 278–287. DOI: 10.1137/1.9781611974331.ch21.

[CH23b]   M. P. Connolly and N. J. Higham. "Probabilistic rounding error analysis of Householder QR factorization". *SIAM J. Matrix Anal. Appl.* 44.3 (2023), pages 1146–1163. DOI: 10.1137/22M1514817.

[CHM21]   M. P. Connolly, N. J. Higham, and T. Mary. "Stochastic rounding and its probabilistic backward error analysis". *SIAM J. Sci. Comput.* 43.1 (2021), A566–A585. DOI: 10.1137/20M1334796.

[DS01]   K. R. Davidson and S. J. Szarek. "Local operator theory, random matrices and Banach spaces". In: *Handbook of the geometry of Banach spaces, Vol. I*. North-Holland, Amsterdam, 2001, pages 317–366. DOI: 10.1016/S1874-5849(01)80010-3.

[DDH07]   J. Demmel, I. Dumitriu, and O. Holtz. "Fast linear algebra is stable". *Numer. Math.* 108.1 (2007), pages 59–91.

[Dem88]   J. W. Demmel. "The probability that a numerical analysis problem is difficult". *Math. Comput.* 50.182 (1988), pages 449–480.

[Dem97]   J. W. Demmel. *Applied Numerical Linear Algebra*. Society for Industrial and Applied Mathematics (SIAM), 1997. DOI: 10.1137/1.9781611971446.

[Día+23]   M. Díaz et al. "Robust, randomized preconditioning for kernel ridge regression". Available from arXiv. 2023. arXiv: 2304.12465.

[Dix83]   J. D. Dixon. "Estimating extremal eigenvalues and condition numbers of matrices". *SIAM J. Numer. Anal.* 20.4 (1983), pages 812–814. DOI: 10.1137/0720053.

[DK01]   P. Drineas and R. Kannan. "Fast Monte-Carlo algorithms for approximate matrix multiplication". In: *Proc. 42nd IEEE Symp. Foundations of Computer Science*. 2001, pages 452–459. DOI: 10.1109/SFCS.2001.959921.

[DM18]   P. Drineas and M. Mahoney. "Lectures on Randomized Linear Algebra". In: *The Mathematics of Data*. Volume 25. IAS/Park City Mathematics Series. American Mathematical Society, 2018. Chapter 4, pages 1–48.

[EY36]   C. Eckart and G. Young. "The approximation of one matrix by another of lower rank". *Psychometrika* 1.3 (1936), pages 211–218.

[Epp23]   E. N. Epperly. "Fast and forward stable randomized algorithms for linear least-squares problems". Available from arXiv. 2023. arXiv: 2311.04362 [math.NA].

[ET24]   E. N. Epperly and J. A. Tropp. "Efficient Error and Variance Estimation for Randomized Matrix Computations". *SIAM J. Sci. Comput.* 46.1 (2024), A508–A528. DOI: 10.1137/23M1558537.

[ETW24]   E. N. Epperly, J. A. Tropp, and R. J. Webber. "XTrace: Making the most of every sample in stochastic trace estimation". *SIAM J. Matrix Anal. Appl.* (Mar. 2024), pages 1–23. DOI: 10.1137/23M1548323.

[Est22]   E. Estrada. "The many facets of the Estrada indices of graphs and networks". *SeMA Journal* 79.1 (2022), pages 57–125. DOI: 10.1007/s40324-021-00275-w.

[FTU23]   Z. Frangella, J. A. Tropp, and M. Udell. "Randomized Nyström preconditioning". *SIAM J. Matrix Anal. Appl.* 44.2 (2023), pages 718–752. DOI: 10.1137/21M1466244.

[FKV98]   A. Frieze, R. Kannan, and S. Vempala. "Fast Monte-Carlo algorithms for finding low-rank approximations". In: *Proc. 39th Ann. Symp. Foundations of Computer Science (Cat. No.98CB36280)*. 1998, pages 370–378. DOI: 10.1109/SFCS.1998.743487.

[GKS23]   Y. Gao, R. Kyng, and D. A. Spielman. "Robust and practical solution of Laplacian equations by approximate elimination". Available from arXiv. Mar. 2023. arXiv: 2303.00709 [cs, math].

[GPW12]   A. C. Gilbert, J. Y. Park, and M. B. Wakin. "Sketched SVD: Recovering spectral features from compressive measurements". *CoRR* abs/1211.0361 (2012). arXiv: 1211.0361.

[Gir89]   D. A. Girard. "A fast "Monte Carlo cross-validation" procedure for large least squares problems with noisy data". *Numer. Math.* 56.1 (1989), pages 1–23. DOI: 10.1007/BF01395775.

[GN51]   H. H. Goldstine and J. von Neumann. "Numerical inverting of matrices of high order. II". *Proc. Amer. Math. Soc.* 2 (1951), pages 188–202. DOI: 10.2307/2032484.

[GL80]   G. H. Golub and C. F. V. Loan. "An analysis of the total least squares problem". *SIAM J. Numer. Anal.* 17.6 (1980), pages 883–893. URL: http://www.jstor.org/stable/2156807.

[GM10]   G. H. Golub and G. Meurant. *Matrices, moments and quadrature with applications*. Princeton University Press, Princeton, NJ, 2010.

[Gv00]   G. H. Golub and H. A. van der Vorst. "Eigenvalue computation in the 20th century". *J. Comput. Appl. Math.* 123.1 (2000). Numerical Analysis 2000. Vol. III: Linear Algebra, pages 35–65. DOI: https://doi.org/10.1016/S0377-0427(00)00413-1.

[GVL13]   G. H. Golub and C. F. Van Loan. *Matrix Computations*. 4th ed. Johns Hopkins University Press, Baltimore, MD, 2013.

[GR15]   R. M. Gower and P. Richtárik. "Randomized iterative methods for linear systems". *SIAM J. Matrix Anal. Appl.* 36.4 (2015), pages 1660–1690. DOI: 10.1137/15M1025487.

[GTP18]   S. Gratton and D. Titley-Peloquin. "Improved bounds for small-sample estimation". *SIAM J. Matrix Anal. Appl.* 39.2 (2018), pages 922–931. DOI: 10.1137/17M1137541.

[GS12]   V. Guruswami and A. K. Sinop. "Optimal column-based low-rank matrix reconstruction". In: *Proc. 2012 Ann. ACM–SIAM Symp. Discrete Algorithms*. Society for Industrial and Applied Mathematics (SIAM), Jan. 2012, pages 1207–1214. DOI: 10.1137/1.9781611973099.95.

[Guţ+20]   M. Guţă et al. "Fast state tomography with optimal error bounds". *J. Phys. A: Math. Theor.* 53.20 (2020), page 204001.

[HMT11]   N. Halko, P. G. Martinsson, and J. A. Tropp. "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions". *SIAM Rev.* 53.2 (2011), pages 217–288. DOI: 10.1137/090771806.

[Hal+11]   N. Halko et al. "An algorithm for the principal component analysis of large data sets". *SIAM J. Sci. Comput.* 33.5 (2011), pages 2580–2594. DOI: 10.1137/100804139.

[HK23]   H. He and D. Kressner. "Randomized joint diagonalization of symmetric matrices". Available from arXiv. 2023. arXiv: 2212.07248 [math.NA].

[Hig90]   N. J. Higham. "Analysis of the Cholesky decomposition of a semi-definite matrix". In: *Reliable Numerical Computation*. Oxford University Press, 1990, pages 161–185. URL: https://eprints.maths.manchester.ac.uk/id/eprint/1193.

[Hig02]   N. J. Higham. *Accuracy and stability of numerical algorithms*. Second. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2002. DOI: 10.1137/1.9780898718027.

[HM19]   N. J. Higham and T. Mary. "A new approach to probabilistic rounding error analysis". *SIAM J. Sci. Comput.* 41.5 (2019), A2815–A2835. DOI: 10.1137/18M1226312.

[HI15]   J. T. Holodnak and I. C. F. Ipsen. "Randomized approximation of the Gram matrix: Exact computation and probabilistic bounds". *SIAM J. Matrix Anal. Appl.* 36.1 (2015), pages 110–137. DOI: 10.1137/130940116.

[Hut90]   M. F. Hutchinson. "A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines". *Comm. Statist. Simulation Comput.* 19.2 (1990), pages 433–450. DOI: 10.1080/03610919008812864.

[Kac37]   S. Kaczmarz. "Angenäherte Auflösung von Systemen linearer Gleichungen". *Bull. Int. Acad. Polon. Sci. Lett. A* (1937), pages 335–357.

[KN14]   D. M. Kane and J. Nelson. "Sparser Johnson–Lindenstrauss Transforms". *J. ACM* 61.1 (2014). DOI: 10.1145/2559902.

[KV17a]   R. Kannan and S. Vempala. "Randomized algorithms in numerical linear algebra". In: *Acta Numerica 2017*. Cambridge Univ. Press, 2017, pages 95–135. DOI: 10.1017/S0962492917000058.

[KV17b]   W. Kong and G. Valiant. "Spectrum estimation from samples". *Ann. Statist.* 45.5 (2017), pages 2218–2247. DOI: 10.1214/16-AOS1525.

[KW92]   J. Kuczyński and H. Woźniakowski. "Estimating the largest eigenvalue by the power and Lanczos algorithms with a random start". *SIAM J. Matrix Anal. Appl.* 13.4 (1992), pages 1094–1122. DOI: 10.1137/0613066.

[KD14]   A. Kundu and P. Drineas. "A note on randomized element-wise matrix sparsification". Available from arXiv. 2014. arXiv: 1404.0320 [cs.IT].

[KS16]   R. Kyng and S. Sachdeva. "Approximate Gaussian elimination for Laplacians—fast, sparse, and simple". In: *57th Ann. IEEE Symp. Foundations of Computer Science—FOCS 2016*. IEEE Computer Soc., Los Alamitos, CA, 2016, pages 573–582.

[LL10]   D. Leventhal and A. S. Lewis. "Randomized methods for linear constraints: convergence rates and conditioning". *Math. Oper. Res.* 35.3 (2010), pages 641–654. DOI: 10.1287/moor.1100.0456.

[Li+17]   H. Li et al. "Algorithm 971: an implementation of a randomized algorithm for principal component analysis". *ACM Trans. Math. Software* 43.3 (2017), Art. 28, 14. DOI: 10.1145/3004053.

[LNW14]   Y. Li, H. L. Nguyen, and D. P. Woodruff. "On sketching matrix norms and the top singular vector". In: *Proc. 25th Ann. ACM–SIAM Symp. Discrete Algorithms*. ACM, New York, 2014, pages 1562–1581. DOI: 10.1137/1.9781611973402.114.

[Liu04]   J. S. Liu. *Monte Carlo Strategies in Scientific Computing*. Springer, 2004. DOI: 10.1007/978-0-387-76371-2.

[Lop19]   M. E. Lopes. "Estimating the algorithmic variance of randomized ensembles via the bootstrap". *Ann. Statist.* 47.2 (2019), pages 1088–1112. DOI: 10.1214/18-AOS1707.

[LP+14]   D. Lopez-Paz et al. "Randomized nonlinear component analysis". In: *Proc. 31st Intl. Conf. Machine Learning*. Volume 32. Proceedings of Machine Learning Research 2. PMLR, 2014, pages 1359–1367. URL: http://proceedings.mlr.press/v32/lopez-paz14.html.

[MRT06]   P.-G. Martinsson, V. Rokhlin, and M. Tygert. *A randomized algorithm for the approximation of matrices*. Yale CS Research Report YALEU/DCS/RR-1361. Yale University, 2006.

[MT20]   P.-G. Martinsson and J. A. Tropp. "Randomized numerical linear algebra: Foundations and algorithms". In: *Acta Numerica 2020*. Volume 29. Cambridge University Press, 2020, pages 403–572.

[Mei+23]   M. Meier et al. "Are sketch-and-precondition least squares solvers numerically stable?" Available from arXiv. 2023. ArXiv: 2302.07202.

[Met87]   N. Metropolis. "The beginning of the Monte Carlo method". *Los Alamos Science* (1987), pages 125–130.

[Mey+21]   R. A. Meyer et al. "Hutch++: Optimal stochastic trace estimation". In: *Symp. Simplicity in Algorithms (SOSA)*. Society for Industrial and Applied Mathematics (SIAM), 2021, pages 142–155. DOI: 10.1137/1.9781611976496.16.

[Mir60]   L. Mirsky. "Symmetric gauge functions and unitarily invariant norms". *Quart. J. Math. Oxford Ser. (2)* 11 (1960), pages 50–59. DOI: 10.1093/qmath/11.1.50.

[Mur+23]   R. Murray et al. "Randomized numerical linear algebra: A perspective on the field with an eye to software". Available from arXiv. 2023. arXiv: 2302.11474 [math.NA].

[MM15]   C. Musco and C. Musco. "Randomized block Krylov methods for stronger and faster approximate singular value decomposition". In: *Adv. Neural Information Processing Systems 28*. Curran Associates, Inc., 2015, pages 1396–1404. URL: http://papers.nips.cc/paper/5735-randomized-block-krylov-methods-for-stronger-and-faster-approximate-singular-value-decomposition.pdf.

[MM20]   C. Musco and C. Musco. "Projection-cost-preserving sketches: Proof strategies and constructions". Available from arXiv. 2020. arXiv: 2004.08434 [cs.DS].

[Mut05]   S. Muthukrishnan. "Data Streams: Algorithms and Applications". *Found. Trends Theor. Comput. Sci.* 1.2 (2005), pages 117–236. DOI: 10.1561/0400000002.

[NST18]   Y. Nakatsukasa, O. Sète, and L. N. Trefethen. "The AAA Algorithm for Rational Approximation". *SIAM J. Sci. Comput.* 40.3 (2018), A1494–A1522. DOI: 10.1137/16M1106122.

[NT24]   Y. Nakatsukasa and J. A. Tropp. "Fast & accurate randomized algorithms for linear systems and eigenvalue problems". *SIAM J. Matrix Anal. Appl.* (2024). Accepted for publication. arXiv: 2111.00113.

[NSW16]   D. Needell, N. Srebro, and R. Ward. "Stochastic gradient descent and the randomized Kaczmarz algorithm". *Math. Prog. Ser. A* 155.1 (2016), pages 549–573.

[NT14]   D. Needell and J. A. Tropp. "Paved with good intentions: Analysis of a randomized block Kaczmarz method". *Linear Algebra Appl.* 441 (2014), pages 199–221.

[Ost59]   A. M. Ostrowski. "A quantitative formulation of Sylvester's law of inertia". *Proc. Natl. Acad. Sci. USA* 45.5 (1959), pages 740–744. URL: http://www.jstor.org/stable/90216.

[PN23]   T. Park and Y. Nakatsukasa. "A fast randomized algorithm for computing an approximate null space". *BIT Numer. Math.* 63.2 (2023), page 36. DOI: 10.1007/s10543-023-00979-7.

[Par95]   D. S. Parker. *A randomizing butterfly transformation useful in block matrix computations*. Computer Science Technical Report. UCLA, 1995.

[Par98]   B. N. Parlett. *The symmetric eigenvalue problem*. Corrected reprint of the 1980 original. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1998. DOI: 10.1137/1.9781611971163.

[PW15]   M. Pilanci and M. J. Wainwright. "Randomized sketches of convex programs with sharp guarantees". *IEEE Trans. Inform. Theory* 61.9 (2015), pages 5096–5115. DOI: 10.1109/TIT.2015.2450722.

[QSS07]   A. Quarteroni, R. Sacco, and F. Saleri. *Numerical Mathematics*. 2nd. Springer, 2007. DOI: 10.1007/b98885.

[RR08]   A. Rahimi and B. Recht. "Random features for large-scale kernel machines". In: *Adv. Neural Information Processing Systems 20*. Curran Associates, Inc., 2008, pages 1177–1184. URL: http://papers.nips.cc/paper/3182-random-features-for-large-scale-kernel-machines.pdf.

[RST09]   V. Rokhlin, A. Szlam, and M. Tygert. "A randomized algorithm for principal component analysis". *SIAM J. Matrix Anal. Appl.* 31.3 (2009), pages 1100–1124.

[RT08]   V. Rokhlin and M. Tygert. "A fast randomized algorithm for overdetermined linear least-squares regression". *Proc. Natl. Acad. Sci. USA* 105.36 (2008), pages 13212–13217. DOI: 10.1073/pnas.0804869105.

[SST06]   A. Sankar, D. A. Spielman, and S.-H. Teng. "Smoothed analysis of the condition numbers and growth factors of matrices". *SIAM J. Matrix Anal. Appl.* 28.2 (2006), pages 446–476. DOI: 10.1137/S0895479803436202.

[Sar06]   T. Sarlós. "Improved approximation algorithms for large matrices via random projections". In: *2006 47th Ann. IEEE Symp. Foundations of Computer Science (FOCS'06)*. 2006, pages 143–152. DOI: 10.1109/FOCS.2006.37.

[Sch86]   R. Schmidt. "Multiple emitter location and signal parameter estimation". *IEEE. Trans. Antennas Propagation* 34.3 (1986), pages 276–280. DOI: 10.1109/TAP.1986.1143830.

[She94]   J. R. Shewchuk. *An introduction to the conjugate gradient method without the agonizing pain*. Technical Report. Carnegie Mellon University, Feb. 1994.

[SS11]   D. A. Spielman and N. Srivastava. "Graph sparsification by effective resistances". *SIAM J. Comput.* 40.6 (2011), pages 1913–1926. DOI: 10.1137/080734029.

[ST02]   D. A. Spielman and S.-H. Teng. "Smoothed analysis of algorithms". In: *Proc. Intl. Cong. Mathematicians. Vol. I*. Plenary lectures and ceremonies, Held in Beijing, August 20–28, 2002, With 1 CD-ROM [Windows]. Higher Education Press, Beijing, 2002, pages 597–606.

[SV09]   T. Strohmer and R. Vershynin. "A randomized Kaczmarz algorithm with exponential convergence". *J. Fourier Anal. Appl.* 15.2 (2009), pages 262–278. DOI: 10.1007/s00041-008-9030-4.

[Tro19]   J. A. Tropp. *Matrix concentration and computational linear algebra*. CMS Lecture Notes 2019-01. Caltech, 2019. DOI: 10.7907/nbx6-vm05.

[Tro20]   J. A. Tropp. *Randomized algorithms for matrix computations*. CMS Lecture Notes 2020-01. Caltech, 2020. DOI: 10.7907/nwsv-df59.

[Tro23]   J. A. Tropp. *Probability Theory & Computational Mathematics*. CMS Lecture Notes 2023-01. Caltech, 2023. DOI: 10.7907/q75sz-e1e79.

[Tro11]   J. A. Tropp. "Improved analysis of the subsampled randomized Hadamard transform". *Adv. Adapt. Data Anal.* 3.1-2 (2011), pages 115–126. DOI: 10.1142/S1793536911000787.

[Tro15]   J. A. Tropp. "An Introduction to Matrix Concentration Inequalities". *Found. Trends Machine Learning* 8.1-2 (2015), pages 1–230. DOI: 10.1561/2200000048.

[Tro22]   J. A. Tropp. "Randomized block Krylov methods for approximating extreme eigenvalues". *Numer. Math.* 150.1 (2022), pages 217–255. DOI: 10.1007/s00211-021-01250-3.

[TW23]   J. A. Tropp and R. J. Webber. "Randomized algorithms for low-rank matrix approximation: Design, analysis, and applications". Available from arXiv. 2023. arXiv: 2306.12418 [math.NA].

[Tro+19]   J. A. Tropp et al. "Streaming low-rank matrix approximation with an application to scientific simulation". *SIAM J. Sci. Comput.* 41.4 (2019), A2430–A2463. DOI: 10.1137/18M1201068.

[UCS17]   S. Ubaru, J. Chen, and Y. Saad. "Fast estimation of $\mathrm{tr}(f(A))$ via stochastic Lanczos quadrature". *SIAM J. Matrix Anal. Appl.* 38.4 (2017), pages 1075–1099. DOI: 10.1137/16M1104974.

[US18]   S. Ubaru and Y. Saad. "Applications of trace estimation techniques". In: *High Performance Computing in Science and Engineering*. Springer International Publishing, 2018, pages 19–33.

[WS01]   C. K. I. Williams and M. Seeger. "Using the Nyström method to speed up kernel machines". In: *Adv. Neural Information Processing Systems 13*. MIT Press, 2001, pages 682–688. URL: http://papers.nips.cc/paper/1866-using-the-nystrom-method-to-speed-up-kernel-machines.pdf.

[Woo22]   D. P. Woodruff. "Lecture 3.2: Affine Embeddings". Course notes for CMU CS 15-859: Algorithms for Big Data. 2022. URL: http://www.cs.cmu.edu/afs/cs/user/dwoodruf/www/teaching/15859-fall22/scribeLec6.pdf.

[Woo14]   D. P. Woodruff. "Sketching as a Tool for Numerical Linear Algebra". *Found. Trends Theor. Comput. Sci.* 10.1-2 (2014), pages 1 –157. DOI: 10.1561/0400000060.

[ZF13]   A. Zouzias and N. M. Freris. "Randomized extended Kaczmarz for solving least squares". *SIAM J. Matrix Anal. Appl.* 34.2 (2013), pages 773–793. DOI: 10.1137/120889897.