Make the Most of What You Have: Resource-Efficient Randomized Algorithms for Matrix Computations

Thesis by
Ethan N. Epperly

In Partial Fulfillment of the Requirements for the
Degree of
Doctor of Philosophy

Caltech

CALIFORNIA INSTITUTE OF TECHNOLOGY
Pasadena, California

2025
Defended May 1, 2025

© 2025

Ethan N. Epperly
ORCID: 0000-0003-0712-8296

ACKNOWLEDGEMENTS

This PhD thesis is the product of the kindness and mentorship provided to me by many people over many years. There are many to thank (and, I'm sure, some who I have regretfully forgotten to acknowledge).

First, I must thank my advisor Joel A. Tropp, from whom I have learned so much—about mathematics, about research, and about writing. I thank Joel for always pushing me to dream bigger, setting my sights on more challenging problems. Thanks also for your detailed feedback on this thesis.

My thanks extends to my other mentors throughout my journey into academic research: Franceso Paesani and Jordi Cirera at UCSD; Don Ward, Bernice Mills, Ryan Sills, and Jonathan Hu at Sandia National Laboratories; Shivkumar Chandrasekaran at UCSB; Andrew Barker at Lawrence Livermore National Laboratory; and Lin Lin at Lawrence Berkeley National Laboratory. I also give thanks to Alf Morales and Tim Shepodd for giving a 15 year-old kid an early shot at pursuing his dream to be a scientist.

I thank the remaining members of my committee, Profs. Venkat Chandrasekaran, Lin Lin, and Franca Hoffmann, for you time in sitting on my dissertation committee and for all I have gotten to learn from you during my PhD.

I feel truly blessed to belong to research community filled with such kind, humble, and thoughtful people. Getting to collaborate with others is truly my favorite part of research. I extend my thanks to the many friends, collaborators, and mentors I have done research with, including Alex Barnett, Chris Camaño, Shivkumar Chandrasekaran, Yifan Chen, Lieven De Lathauwer, Mateo Díaz, Zhiyan Ding, Gil Goldshlager, Zachary Frangella, Nithin Govindarajan, Anne Greenbaum, Alex Hsu, Daniel Kressner, Lin Lin, Maike Meier, Eliza O'Reilly, Kevin Miller, Raphael Meyer, Elvira Moreno, Chris Musco, Yuji Nakatsukasa, Taejun Park, Akash Rao, Heather Wilber, and Ruizhe Zhang. I also thank Deeksha Adil, Haoxuan Chen, Tyler Chen, Alice Cortinovis, Jorge Garza-Vargas, Eitan Levin, and Aleksandros Sobczyk for many insightful conversations. The research process is inherently collaborative, and this thesis is suffused with the creativity of these people.

Special thanks are merited to Rob Webber, in particular, who joined Caltech as a postdoctoral scholar during the second year of my PhD. I truly cannot imagine what

my PhD would have been without your collaboration, your mentorship, and your friendship.

I am extremely grateful to Chris Musco for his invitation to visit NYU in the spring of 2024. It was a very enjoyable and productive visit, and I learned a lot from collaborating with you.

My warmest thanks go to administrative staff at Caltech, most especially Jolene Brink and Maria Lopez. Thanks to all the help you have provided to me and other students in our department.

My PhD experience truly would not have been what it's been without the Department of Energy Computational Science Graduate Fellowship and the community of people I have gotten to meet and interact with through it. I send my thanks to Lindsey Eilts and all the folks at the Krell Institute who manage the fellowship and provide so much support to us fellows.

I give my deepest gratitude to my family. To my parents Meg and Tom, who have always loved and supported me throughout my life and nurtured my love of learning. All of that time playing board games, those long conversations about what I was learning in school, and those *Science on Saturday* talks left a big impression on me. To my brother Aidan, it is truly a gift to have a close friend who loves talking math just as much as I do (possibly even more). To my grandparents Kathie, Bill, Christine, John, Bob, and Sarah, for all the wisdom and life experience you have shared.

Doing a PhD is a long journey, and I truly believe I could not have made it without the love and support of my fiancée, Sierra Williams, DVM. The level of dedication and intensity you bring to everything you do is an inspiration to me, and you have always been there for me when I needed it. This love and thanks extends to our pets: dogs Hulk and Finn and three-legged turtle Shelly. Nothing brought me joy throughout my PhD quite like going on a walk with the dogs or sitting down all five of us to watch an episode of Survivor.

ABSTRACT

In recent years, randomized algorithms have established themselves as fundamental tools in computational linear algebra, with applications in scientific computing, machine learning, and quantum information science. Many randomized matrix algorithms proceed by first collecting information about a matrix and then processing that data to perform some computational task. This thesis addresses the following question: How can one design algorithms that use this information as efficiently as possible, reliably achieving the greatest possible speed and accuracy for a limited data budget? This question is timely, as randomized algorithms are increasingly being deployed in production software and in applications where accuracy and reliability is critical.

The first part of this thesis focuses on the problem of low-rank approximation for positive-semidefinite matrices, motivated by applications to accelerating kernel and Gaussian process machine learning methods. Here, the goal is to compute an accurate approximation to a matrix after accessing as few entries of the matrix as possible. This part of the thesis explores the randomly pivoted Cholesky (RPCHOLESKY) algorithm for this task, which achieves a level of speed and reliability greater than other methods for the same problem.

The second part of this thesis considers the task of estimating attributes of an implicit matrix accessible only by matrix–vector products, motivated by applications in quantum physics, network science, and machine learning. This thesis describes the *leave-one-out approach* to developing matrix attribute estimation algorithms, and develops optimized trace, diagonal, and row-norm estimation algorithms for this computational model.

The third part of this thesis considers randomized algorithms for overdetermined linear least squares problems, which arise in statistics and machine learning. Randomized algorithms for linear-least squares problems are asymptotically faster than any known deterministic algorithm, but recent work of [Meier et al., SIMAX '24] raised questions about the accuracy of these methods when implemented in floating point arithmetic. This thesis shows these issues are resolvable by developing fast randomized least-squares problem achieving backward stability, the gold-standard accuracy and stability guarantee for a numerical algorithm.

PUBLISHED CONTENT AND CONTRIBUTIONS

[CETW25]   Yifan Chen, Ethan N. Epperly, Joel A. Tropp, and Robert J. Webber. "Randomly Pivoted Cholesky: Practical Approximation of a Kernel Matrix with Few Entry Evaluations". In: *Communications on Pure and Applied Mathematics* 78.5 (2025). ENE participated in the conception of this project, proposed the block RPCHOLESKY algorithm variant, wrote most of the code, designed and ran most of the experiments, and was significantly involved in writing. DOI: 10.1002/cpa.22234.

[EMN25]   Ethan N. Epperly, Maike Meier, and Yuji Nakatsukasa. *Fast Randomized Least-Squares Solvers Can Be Just as Accurate and Stable as Classical Direct Solvers*. ENE participated in the conception of this project, designed several algorithms, proved the main theoretical results, wrote the code, designed and ran most of the numerical experiments, and was significantly involved in writing. 2025. URL: https://arxiv.org/abs/2406.03468v2.

[ETW25]   Ethan N. Epperly, Joel A. Tropp, and Robert J. Webber. *Embrace Rejection: Kernel Matrix Approximation by Accelerated Randomly Pivoted Cholesky*. ENE participated in the conception of this project, co-designed the main algorithms, wrote the code, designed and ran many of the numerical experiments, and was significantly involved in writing. Apr. 2025. arXiv: 2410.03969v3.

[DEFT+24] Mateo Díaz, Ethan N. Epperly, Zachary Frangella, Joel A. Tropp, and Robert J. Webber. *Robust, Randomized Preconditioning for Kernel Ridge Regression*. ENE participated in the conception of this project, was involved in writing the code and designing and running numerical experiments, proved some of the theoretical results, and was significantly involved in writing. July 2024. arXiv: 2304.12465v5.

[Epp24]   Ethan N. Epperly. "Fast and Forward Stable Randomized Algorithms for Linear Least-Squares Problems". In: *SIAM Journal on Matrix Analysis and Applications* (Dec. 2024). This paper was solely written by ENE. DOI: 10.1137/23M1616790.

[ET24]   Ethan N. Epperly and Joel A. Tropp. "Efficient Error and Variance Estimation for Randomized Matrix Computations". In: *SIAM Journal on Scientific Computing* 46.1 (Feb. 2024). ENE participated in the conception of this project, co-designed the main algorithms, wrote the code, co-designed and ran the numerical experiments, and was significantly involved in writing. DOI: 10.1137/23M1558537.

[ETW24]   Ethan N. Epperly, Joel A. Tropp, and Robert J. Webber. "XTrace: Making the Most of Every Sample in Stochastic Trace Estimation". In: *SIAM Journal on Matrix Analysis and Applications* (Mar. 2024). ENE participated in the conception of this project, designed several of the algorithms, wrote the code, designed and ran the numerical experiments, and was significantly involved in writing. DOI: 10.1137/23M1548323.

[EM23]   Ethan N. Epperly and Elvira Moreno. "Kernel Quadrature with Randomly Pivoted Cholesky". In: *Advances in Neural Information Processing Systems* 36 (2023). ENE participated in the conception of this project, designed the main algorithms, wrote the code, proved the main theoretical results, designed and ran many of the numerical experiments, and was significantly involved in writing. URL: https://dl.acm.org/doi/10.5555/3666122.3668997.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

LIST OF PROGRAMS

C h a p t e r   1

INTRODUCTION

Our experience suggests that many practitioners of scientific computing view randomized algorithms as a desperate and final resort. Let us address this concern immediately. Classical Monte Carlo methods are highly sensitive to the random number generator and typically produce output with low and uncertain accuracy. In contrast, the algorithms discussed herein are relatively insensitive to the quality of randomness and produce highly accurate results.

Nathan Halko, Per-Gunnar Martinsson, and Joel Tropp, *Finding structure with randomness: Probabilistic algorithms for constructing matrix decompositions* [HMT11]

Randomness has played a fundamental role in digital computation throughout its history, dating back to von Neumann and Ulam's groundbreaking work in the 1940s using Monte Carlo to study neutron diffusion problem. (See [Eck87] for discussion of this history.) Despite this, randomized methods have also been traditionally viewed with great skepticism by many practitioners of scientific computing. The belief was that randomized methods could only yield low-accuracy solutions to computational problems; as such, they were only useful as a "desperate and final resort", typically for problems of such high dimensions that traditional methods were doomed to failure.

In modern scientific computing, this conventional wisdom has been completely upended. Some of the most exciting progress has been for problems in computational linear algebra, the subject of this thesis. Randomized algorithms have established them as the most effective methods for linear algebra problems such as matrix low-rank approximation [HMT11; MM15; Nak20; TW23] and highly overdetermined linear least-squares problems [RT08; AMT10]. Randomized algorithms can produce solutions of high-accuracy (as accurate as the numerical precision, in some case) and with negligible probability of failure.

The modern field of randomized matrix computations has its origins around the turn of the millennium [FKV98; PTRV98], and its historical roots stretch into the

twentieth century [Lan50; Gir87; KW92]. As such, this is not a new subject, and many surveys exist to document the field's successes [HMT11; Woo14; DM16; KV17; MT20; TW23; MDME+23; KT24]. Despite this multi-decade history, this thesis and its aims remain timely. We are in the midst of a migration of the randomized matrix algorithm from its natural habitat in the SIAM journal or computer science theory conference to the realm of production software. Now, more than ever, we need randomized algorithms that are ready for deployment in practice, making maximally efficient use of computational resources and achieving the same level of accuracy and reliability as classical deterministic methods, while maintaining speed. The goal of this thesis is to describe algorithms developed over the course of my PhD research that meet these criteria.

Many randomized algorithms work by first *collecting information* about a matrix, then using this information to produce an (approximate) solution to a linear algebra problem. A unifying theme of this thesis is the design of algorithms that *make the most of what you have*—that is, algorithms that attempt to collect as little information as possible about the matrix and use the collected information in a maximally efficient way to achieve the greatest possible accuracy subject to a limited computational budget. This thesis is divided into three parts, each of which uses the "make the most of what you have" principle in a different way.

1.1   Part I: Random pivoting

Information collected: Entries of a positive-semidefinite matrix A.

Computational goal: Produce a *low-rank* approximation $\widehat{A} \approx A$.

Kernel and Gaussian process methods in machine learning [SS02; RW05] remain some of the most effective tools for scientific machine learning [MCRR20; BDHO24]. However, these methods face a fundamental limitation called the curse of kernelization:

Curse of kernelization. Direct implementations of kernel and Gaussian process methods on n data points requires forming and manipulating an $n \times n$ positive-semidefinite kernel matrix A. Doing so requires $O(n^2)$ storage and $O(n^3)$ operations.

The curse of kernelization makes direct implementation of kernel and Gaussian

process methods infeasible for large data sets.

Randomization offers a path forward: Using randomized algorithms, we can compute a *low-rank approximation* $\widehat{A} \approx A$ to the kernel matrix. By using the low-rank approximation in place of the kernel matrix, the computational cost of kernel and Gaussian process methods can be substantially reduced.

In order to maintain computational efficiency, kernel matrix low-rank approximation algorithms must produce an approximation to the matrix A after only accessing a small number of entries of the matrix; after all, reading the full matrix even a single time incurs an expensive cost of $O(n^2)$ operations. Is it possible to produce an accurate approximation to a matrix after looking at a fraction of its entries? And if so, what is most economical algorithm?

This thesis advocates for the *randomly pivoted Cholesky* algorithm as an answer to these questions. The randomly pivoted Cholesky algorithm produces a *near-optimal* rank-k approximation to any positive-semidefinite matrix using just $(k + 1)n$ entry accesses. Compared to other algorithms for kernel matrix low-rank approximation, randomly pivoted Cholesky is either faster or more accurate when applied to challenging instances.

Part I of this thesis introduces the randomly pivoted Cholesky algorithm, describes its properties, explains how it can be used to accelerate kernel and Gaussian process machine learning algorithms, and compares it to alternative methods. After a thorough investigation of randomly pivoted Cholesky and positive-semidefinite matrix low-rank approximation, this part also discuss extensions of the random pivoting approach to approximating general, rectangular matrices.

1.2 Part II: Leave-one-out randomized matrix algorithms

> ***Information collected:*** Matrix–vector products $B\omega_1, \dots, B\omega_s$.
>
> ***Computational goal:*** Estimate attributes associated to the matrix B such as its trace, diagonal, or row norms.

In fields ranging from quantum physics to network science to machine learning, we work with matrices B that are accessible only indirectly. In such settings, we cannot read the entries b_{ij} directly. Rather, we can access the matrix through matrix–vector products: Given a vector ω, we may access $B\omega$.

Under this implicit access model, linear algebra problems that would ordinarily be

trivial become interesting. For instance, given access to an implicit matrix B, how can we compute its trace? Its diagonal? Its row norms? For all of these questions, we seek algorithms that are maximally efficient, in the sense of requiring as few matrix–vector products to compute these *matrix attributes* to a desired accuracy level.

Part II describes the *leave-one-out approach* to estimating attributes of an implicit matrix. We will then use the leave-one-out approach to derive optimized estimators for the trace, diagonal, and row norms of both general and positive-semidefinite matrices. As additional applications of the leave-one-out approach, we will develop error estimates for randomized low-rank approximations like the randomized SVD and variance estimation techniques for assessing the quality of more general quantities computed by randomized matrix algorithms.

1.3 Part III: Sketching, solvers, and stability

> ***Information collected:*** A "sketch" of a tall matrix $B \in \mathbb{R}^{m \times n}$ with $m \gg n$, defined to be the product $S^* B$ of B with a wide matrix $S^* \in \mathbb{R}^{d \times m}$.
>
> ***Computational goal:*** Solve the overdetermined linear least-squares problem $x = \operatorname{argmin}_{z \in \mathbb{R}^n} \|c - Bz\|$ to high numerical accuracy.

One of the major success stories of randomized algorithms in linear algebra is sketch-and-precondition algorithm [RT08; AMT10] for solving over-determined linear least-squares problems

$$x = \operatorname*{argmin}_{z \in \mathbb{R}^n} \|c - Bz\| \quad \text{where } B \in \mathbb{R}^{m \times n}, \ c \in \mathbb{R}^m. \tag{1.1}$$

Here, $\|\cdot\|$ denotes the vector ℓ_2 norm, and we consider the *highly overdetermined setting* where $m \gg n$. The idea of sketch-and-precondition is to preprocess the matrix B by applying a randomized dimensionality reduction map or "sketching matrix" $S^* \in \mathbb{R}^{d \times m}$, where the *embedding dimension d* is a small multiple of the number of columns n in B, e.g., $d = 2n$. We then compute a QR decomposition of the sketched matrix $S^* B = QR$ and solve the least-squares problem (1.1) using a Krylov iterative method with R as preconditioner. With an appropriate choice of sketching matrix, this procedure reliably reduces the condition number of B below an absolute constant, leading to convergence to high accuracy in a small number of Krylov iterations. As a result, sketch-and-precondition runs in roughly $O(mn + n^3)$ operations, a dramatic improvement to the $O(mn^2)$ cost of classical algorithms for

least squares.

The recent paper [MNTW24] cast doubts on the *numerical stability* of this algorithm. This paper's experiments show that, when implemented using ordinary double-precision floating point arithmetic, some versions of sketch-and-precondition produce errors that exceed classical methods by orders of magnitude. Does this mean the sketch-and-precondition algorithm is too unreliable for practical use?

Part III critically investigates this question. We will see that, by using careful initialization and just a single step of *iterative refinement*, sketch-and-precondition-type algorithms can be made just as accurate as classical direct methods, achieving *backward stability*, the gold-standard notion of accuracy in numerical analysis.

1.4 About this thesis

I have taken non-traditional approach to writing this thesis. As with many theses, this thesis is primarily based on research papers written during my PhD. However, rather then collecting these papers and editing them, this thesis contains a new treatment of this material. As such, this thesis is composed of original writing and constitutes a deeper dive and re-examination of my PhD research.

Rather than trying to include all of my PhD research, I have elected to focus on the three topics described above: random pivoting, leave-one-out randomized matrix algorithms, and numerically stable randomized least-squares algorithms. I have endeavored to provide a an approachable introduction to these three areas of my research that I hope will be accessible to younger researchers. As such, I have included a significant amount of discussion of background material and related work; I have also included open problems at the end of each part of the thesis. This thesis also contains several new results and algorithms that have not previously been published.

The contents of this thesis are diagrammed in Fig. 1.1, including dependencies between chapters. For readers interested primarily in seeing my original research, I have used symbols to distinguish between chapters based on whether they contain primarily exposition, research, or open problems. I have endeavored to have all of the chapters, including the expository ones, present a unique perspective on the material. Chapters containing research that is newly presented in this thesis are marked using a star.

Given my focus on a subset of topics, much of my PhD research is not discussed in this thesis, including my work on quantum eigenvalue algorithms [ELN22;
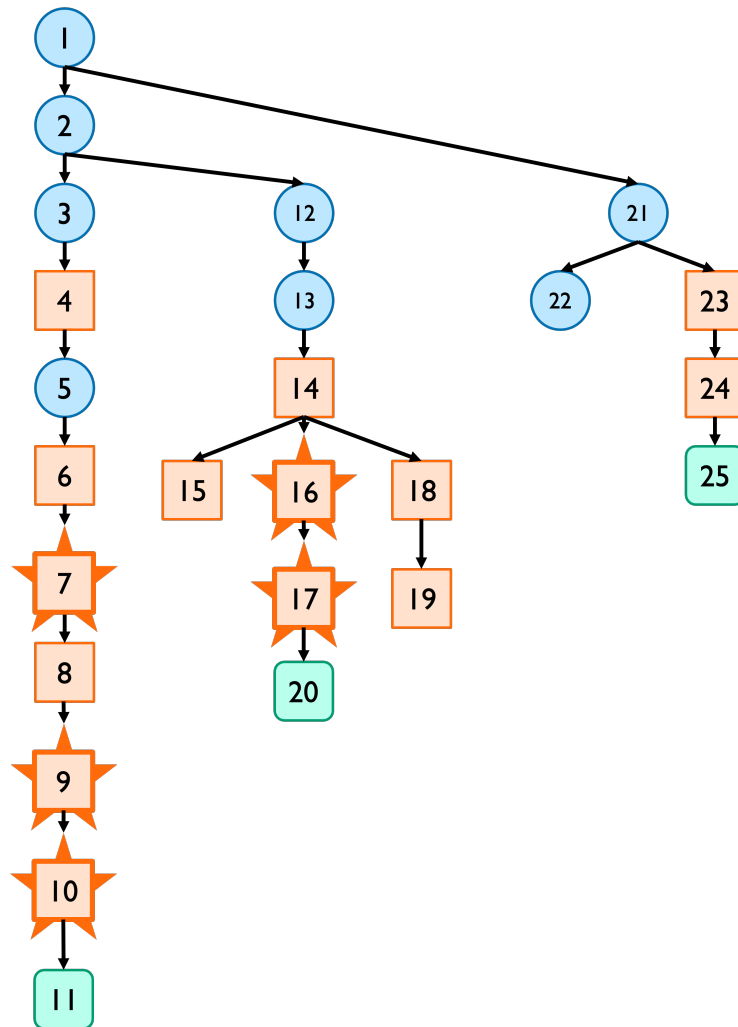
Figure 1.1: List of chapters in this thesis and dependencies between them. Blue circles indicate chapters whose content is primarily introductory or expository, orange squares indicate sections primarily containing research from my PhD, and green squircles indicate open questions. Starred sections contain new research that has not previously been published.

DELZ24], row action methods for large least-squares problems [EGW25], super-fast non-uniform Fourier transform inversion [WEB25], tensor network methods [CET25], and uniqueness of tensor decompositions [GEL22]. All of these topics are near and dear to my heart, and it is will sadness that I do not include them in the present document.

Taking inspiration from Nick Trefethen's essay *Ten Digit Algorithms* [Tre05, §4], I have elected to present algorithms using code segments rather than pseudocode. As a numerical linear algebraist at heart, the programming language will be MATLAB; translation to Python or Julia for those so-inclined will hopefully not be difficult.

1.5 Notation

This thesis will work over the field $\mathbb{K}$ of either real or complex numbers, $\mathbb{K} = \mathbb{R}$ or $\mathbb{K} = \mathbb{C}$. The imaginary unit is denoted i, and the complex conjugate is $\overline{x + \mathrm{i}y} = x - \mathrm{i}y$. The nonnegative reals are denoted $\mathbb{R}_+$.

Matrices and vectors. The symbol * denotes the adjoint, which reduces to the transpose when $\mathbb{K} = \mathbb{R}$. We identify the set of vectors $\mathbb{K}^n$ with the set of $n \times 1$ matrices $\mathbb{K}^{n \times 1}$; row vectors $\boldsymbol{b}^* \in \mathbb{K}^{1 \times n}$ will be denoted as adjoints of column vectors $\boldsymbol{b} \in \mathbb{K}^n$.

Matrices will be denoted by bold capital letters (e.g., $\boldsymbol{B}, \boldsymbol{\Omega}$), and vectors will be denoted by bold lowercase letters (e.g., $\boldsymbol{b}, \boldsymbol{\omega}$). Entries of matrices and vectors are denoted using the corresponding italic letter, e.g., b_i is the ith entry of $\boldsymbol{b}$ and ω_{ij} is the ijth entry of $\boldsymbol{\Omega}$. Columns of matrices are denoted using the corresponding bold lowercase vector, e.g., $\boldsymbol{\omega}_i$ is the ith column of $\boldsymbol{\Omega}$. A matrix $\boldsymbol{\Omega}$ with its ith column deleted is denoted $\boldsymbol{\Omega}_{-i}$ (similarly, $\boldsymbol{\Omega}_{-ij}$ for two deletions).

The identity matrix is $\mathbf{I} \in \mathbb{R}^{n \times n}$; its columns are the standard basis vectors $\mathbf{e}_i \in \mathbb{R}^n$. The vector of all ones is $\mathbf{1} = \sum_{i=1}^{n} \mathbf{e}_i$, and the matrix of zeros is $\mathbf{0} \in \mathbb{R}^{m \times n}$. We will use subscripts $\mathbf{I}_n, \mathbf{1}_n, \mathbf{0}_{m \times n}$ when necessary to indicate the dimensions.

For our purposes, a matrix $\boldsymbol{A} \in \mathbb{K}^{n \times n}$ is *positive semidefinite* (psd) if it is Hermitian ($\boldsymbol{A} = \boldsymbol{A}^*$) and satisfies the condition $\boldsymbol{x}^* \boldsymbol{A} \boldsymbol{x} \geq 0$ for all vectors $\boldsymbol{x} \in \mathbb{K}^n$. The concept of a "real nonsymmetric psd matrix", valuable in other contexts, will not be considered in this thesis. The psd order $\succeq$ is defined so that $\boldsymbol{A} \succeq \boldsymbol{H}$ whenever $\boldsymbol{A} - \boldsymbol{H}$ is psd.

Throughout this thesis, $\boldsymbol{B} \in \mathbb{K}^{m \times n}$ will denote a general matrix, which may be square or rectangular depending on the context. The matrix $\boldsymbol{A} \in \mathbb{K}^{n \times n}$ will always denote a Hermitian matrix, often psd.

Sets of integer indices, general arbitrary sets, and vector spaces will be denoted using sans serif font (e.g., S, V). General inner products, when they arise, will be conjugate linear in the *first* coordinate.

We permit matrices $\boldsymbol{B} \in \mathbb{K}^{\mathsf{S} \times \mathsf{T}}$ and vectors $\boldsymbol{x} \in \mathbb{K}^{\mathsf{S}}$ to be indexed by arbitrary finite sets S and T. Given bivariate and univariate functions $\xi : \mathsf{X} \times \mathsf{Y} \to \mathbb{K}$ and $f : \mathsf{X} \to \mathbb{K}$ and subsets $\mathsf{S} \subseteq \mathsf{X}$ and $\mathsf{T} \subseteq \mathsf{Y}$, $\xi(\mathsf{S}, \mathsf{T}) \in \mathbb{K}^{\mathsf{S} \times \mathsf{T}}$ and $f(\mathsf{S}) \in \mathbb{K}^{\mathsf{S}}$ denote a matrix and vector with elements $\xi(s, t)$ and $f(s)$ for each $s \in \mathsf{S}$ and $t \in \mathsf{T}$.

We use MATLAB notation to index matrices. So $\boldsymbol{B}(i, j) = b_{ij}$ is the ijth entry of $\boldsymbol{B}$, $\boldsymbol{B}(:, i) = \boldsymbol{b}_i$ is the ith column, $\boldsymbol{B}(:, \mathsf{S}) = (\boldsymbol{b}_i : i \in \mathsf{S})$ denotes the submatrix of $\boldsymbol{B}$ indexed by elements of the set S, etc. We adopt the convention that submatrices $\boldsymbol{B}(\mathsf{S}, \mathsf{T}) \in \mathbb{K}^{\mathsf{S} \times \mathsf{T}}$ are indexed by the sets S and T rather than by the sets $\{1, \ldots, |\mathsf{S}|\}$ and $\{1, \ldots, |\mathsf{T}|\}$. So b_{22} is still the $(2, 2)$ entry of the matrix $\boldsymbol{B}(\{2, 3\}, \{2, 5\})$.

For a square matrix $\boldsymbol{B} \in \mathbb{K}^{n \times n}$, the trace is $\mathrm{tr}(\boldsymbol{B}) = \sum_{i=1}^{n} b_{ii} \in \mathbb{K}$ and the diagonal is $\mathbf{diag}(\boldsymbol{B}) = (b_{ii} : 1 \leq i \leq n) \in \mathbb{K}^n$. The diagonal elements of a matrix product $\boldsymbol{G}^* \boldsymbol{F}$ are denoted

$$\mathbf{diagprod}(\boldsymbol{F}, \boldsymbol{G}) := \mathbf{diag}(\boldsymbol{F}^* \boldsymbol{G}).$$

Given a vector $\boldsymbol{b} \in \mathbb{K}^n$, the diagonal matrix with elements b_i is denoted $\mathbf{Diag}(\boldsymbol{b})$. The squared row norms and squared column norms of a matrix $\boldsymbol{B} \in \mathbb{K}^{m \times n}$ are

$$\mathbf{srn}(\boldsymbol{B}) := \left(\|\boldsymbol{B}(i, :)\|^2 : 1 \leq i \leq m \right) \qquad = \mathbf{diag}(\boldsymbol{B}\boldsymbol{B}^*) \qquad \in \mathbb{K}^m,$$
$$\mathbf{scn}(\boldsymbol{B}) := \left(\|\boldsymbol{B}(:, i)\|^2 : 1 \leq i \leq n \right) \qquad = \mathbf{diag}(\boldsymbol{B}^* \boldsymbol{B}) \qquad \in \mathbb{K}^n.$$

The entrywise product of vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{K}^n$ is $\boldsymbol{a} \odot \boldsymbol{b} = (a_i \cdot b_i : 1 \leq i \leq n)$. Nonlinear operations are applied to vectors entrywise. For instance, $\overline{\boldsymbol{a}} = (\overline{a_i} : 1 \leq i \leq n)$ is the entrywise complex conjugate, and $|\boldsymbol{a}|^2 = (|a_i|^2 : 1 \leq i \leq n)$ is the entrywise squared modulus.

The ℓ_2 norm of a vector or spectral norm of a matrix will be denoted $\|\cdot\|$. We will also make use of the Frobenius and trace norms, denoted $\|\cdot\|_{\mathrm{F}}$ and $\|\cdot\|_*$. Schatten p-norms are denoted $\|\cdot\|_{\mathsf{S}_p}$.

We will make frequent use of matrix decompositions. Given a tall matrix $\boldsymbol{B} \in \mathbb{K}^{m \times n}$, an economy-size QR decomposition is a factorization $\boldsymbol{B} = \boldsymbol{Q}\boldsymbol{R}$ into a matrix $\boldsymbol{Q} \in \mathbb{K}^{m \times n}$ with orthonormal columns and an upper triangular matrix $\boldsymbol{R} \in \mathbb{K}^{n \times n}$. An economy-size SVD is a factorization $\boldsymbol{B} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^*$ into a matrix $\boldsymbol{U} \in \mathbb{K}^{m \times n}$ with orthonormal columns, a nonnegative diagonal matrix $\boldsymbol{\Sigma} \in \mathbb{R}_+^{n \times n}$, and a unitary matrix

$V \in \mathbb{K}^{n \times n}$. When not otherwise stated, all QR decompositions and SVDs will be assumed to be economy-size. Given a positive definite matrix A, its (full) Cholesky decomposition refers to one of two matrix decompositions, the *lower triangular Cholesky decomposition* $A = LL^*$ or the *upper triangular Cholesky decomposition* $A = R^*R$. Both conventions will be convenient for use in different places of this thesis.

We will use $\mathbf{Orth}(B)$ to denote a matrix whose columns form an orthonormal basis range(B). When necessary to yield an unambiguous interpretation, $\mathbf{Orth}$ of a full column-rank matrix B will be taken to be the output of an economy-size QR decomposition where the triangular factor has positive diagonal entries.

As usual, rank(B) denotes the dimension of the range of B. We say a matrix is *rank-k* if rank(B) $\leq k$. A rank-k matrix $B \in \mathbb{K}^{m \times n}$ is described by its *compact SVD*

$$B = U\Sigma V^*$$

where $U \in \mathbb{K}^{m \times \mathrm{rank}(B)}$ and $V \in \mathbb{K}^{n \times \mathrm{rank}(B)}$ have orthonormal columns and $\Sigma \in \mathbb{R}_+^{\mathrm{rank}(B) \times \mathrm{rank}(B)}$ is a diagonal matrix listing the *nonzero* singular values. Similarly, a rank-k Hermitian matrix $A \in \mathbb{K}^{n \times n}$ has a compact eigendecomposition

$$A = UDU^*$$

where $U \in \mathbb{K}^{n \times \mathrm{rank}(A)}$ has orthonormal columns and $D \in \mathbb{R}^{\mathrm{rank}(A) \times \mathrm{rank}(A)}$ is a diagonal matrix listing the nonzero eigenvalues.

Probability theory. Probabilities and expectations are denoted $\mathbb{P}$ and $\mathbb{E}$. Nonlinear operations bind before expectations, so that $\mathbb{E}x^2 := \mathbb{E}(x^2)$, *not* $\mathbb{E}x^2 = (\mathbb{E}x)^2$. The covariance is

$$\mathrm{Cov}(a, b) := \mathbb{E}[(a - \mathbb{E}a)\overline{(b - \mathbb{E}b)}],$$

and the variance is $\mathrm{Var}(a) := \mathrm{Var}(a, a)$. Note that, unlike other sesquilinear forms in this thesis, the covariance is conjugate-linear in its *second* argument.

We write $x_1, x_2, \ldots \sim x$ and variations thereof to indicate that $x_1, x_2, \ldots$ are random variables with the same distribution as x. The phrase "independent and identically distributed" carries its usual abbreviation *iid*, and we write $x_1, x_2, \ldots \overset{\text{iid}}{\sim} x$ when $x_1, x_2, \ldots$ are iid copies of x.

Given a nonnegative *weight vector* $w \in \mathbb{R}_+^n$, $i \sim w$ denotes a random integer $i \in \{1, \ldots, n\}$ selected with probability $\mathbb{P}\{i = j\} = w_j / \sum_{k=1}^n w_k$. We *do not assume the normalization* $\sum_{i=1}^n w_i = 1$ when writing $i \sim w$.

The uniform distributions on a set S are denoted $\textsc{Unif}\,\mathsf{S}$. The unit sphere of a vector (sub)space U is $\mathbb{S}(\mathsf{U})$, and $\textsc{Unif}\,\mathbb{S}(\mathsf{U})$ denotes the (Haar) uniform distribution on the sphere. A Gaussian distribution over the field $\mathbb{K}$ with mean $\boldsymbol{m}$ and covariance matrix $\boldsymbol{\Sigma}$ is denoted $\textsc{Normal}_{\mathbb{K}}(\boldsymbol{m}, \boldsymbol{\Sigma})$. If $\boldsymbol{\Sigma}$ is nonsingular, its probability density is

$$p(z) = \frac{1}{(2\pi)^{\alpha/2} \det(\boldsymbol{\Sigma})^{1/2}} \exp\left(-\frac{z^*\boldsymbol{\Sigma}^{-1}z}{2}\right).$$

A random vector $\boldsymbol{\omega} \sim \textsc{Normal}_{\mathbb{K}}(\boldsymbol{m}, \boldsymbol{\Sigma})$ has jointly Gaussian entries with mean $\mathbb{E}[\omega_i] = m_i$ and $\mathrm{Cov}(\omega_i, \omega_j) = \sigma_{ij}$.

Rounding error analysis. In Part III of this thesis, we will investigate the *numerical stability* of randomized least-squares solvers, which will require special notation. This notation will be introduced in Section 22.1.

Chapter 2

LOW-RANK APPROXIMATION FOUNDATIONS

Despite decades of research on Lanczos methods, the theory for the randomized algorithms is more complete and provides strong guarantees of excellent accuracy, whether or not there exist any gaps between the singular values.

Huamin Li, George C. Linderman, Arthur Szlam, Kelly P. Stanton, Yuval Kluger, and Mark Tygert, *Algorithm 971: An Implementation of a Randomized Algorithm for Principal Component Analysis* [LLSS+17]

This thesis is not *only* about randomized algorithms for low-rank approximation, but low-rank approximation will play an important role throughout our discussion. This introductory chapter reviews two types of low-rank approximation, projection approximation and Nyström approximation.

Sources. This chapter is introductory and is not based on any particular research article. The last section is adapted from the blog post [Epp24c].

Outline. Section 2.1 begins by describing (approximately) low-rank matrices and how they can be represented. We then go on to discuss two types of low-rank approximations: projection approximations (Section 2.2) and Nyström approximations (Section 2.5). Examples of the former type of approximation is given by the randomized SVD (Section 2.3), which can be improved using subspace iteration (Section 2.4). Section 2.6 concludes by describing a connection between projection approximations and Nyström approximations I have termed the *Gram correspondence*. The Gram correspondence will be an important tool throughout the first part of this thesis.

2.1 Low-rank approximation basics

The term "low-rank matrix" is an informal one. Colloquially, we say a matrix $B \in \mathbb{K}^{m \times n}$ is low-rank if rank B is much smaller than its dimensions m and n. Low-rank matrices are convenient to work with in computations because a rank-k may be decomposed as an outer product

$$B = FG^* \tag{2.1}$$

of thin matrices $F \in \mathbb{K}^{m \times k}$ and $G \in \mathbb{K}^{n \times k}$. As such, a low-rank matrix B can be represented by storing the factor matrices F and G, resulting in a significant reduction in storage cost $(m + n)k \ll mn$.

The starting point of this part of the thesis is a beautiful and practically useful observation: Among the numerous matrices appearing in applications, many of them possess the property of being *well-approximated by a low-rank matrix*. For such matrices $B \in \mathbb{K}^{m \times n}$, they can be efficiently represented and computed with by storing a low-rank approximation $\widehat{B} \approx B$ in factored form $\widehat{B} = FG^*$. Provided this approximation is accurate (i.e., $\|\|B - \widehat{B}\|\| \ll \|\|B\|\|$ in an appropriate norm), an approximately low-rank matrix can be accurately represented by storing a small number of parameters. Some explanations for the ubiquity of approximately low-rank matrices in applications are provided in [BT19; UT19].

Given a factored low-rank approximation $\widehat{B} = FG^*$ to a matrix B, many computational problems involving B become easy:

- *Matrix–vector products:* Matrix–vector products $z \mapsto \widehat{B}z$ can be computed in $O((m + n)k)$ operations by evaluating the product as $\widehat{B}z = F(G^*z)$.

- *Singular values and vectors:* The low-rank factorization $\widehat{B} = FG^*$ can be upgraded to a compact SVD $\widehat{B} = U\Sigma V^*$ in $O((m + n)k^2)$ operations. From this decomposition, one can obtain the singular values and singular vectors of $\widehat{B}$. From the SVD, one can also compute other quantities such as unitarily invariant norms and projectors onto singular subspaces.

- *Linear systems and least-squares:* Shifted linear systems of the form

$$(\widehat{B} + \mu \mathbf{I})x = c \quad \text{for } \mu \in \mathbb{C} \setminus \{0\}$$

  and regularized least-squares problems

$$x = \underset{z \in \mathbb{K}^n}{\operatorname{argmin}} \|\widehat{B}z - c\|^2 + \mu \|z\|^2 \quad \text{for } \mu > 0$$

  can both be solved in $O((m + n)k^2)$ operations (by the Sherman–Morrison–Woodbury identity in the former case and a compact SVD in the latter case).

Thus, given both the great prevalence of approximately low-rank matrices in applications and the great computational benefits of exploiting low-rank structure, the

matric low-rank approximation problem is of fundamental interest in computational mathematics.

The central theoretical result in low-rank approximation is the Schmidt–Mirsky–Eckart–Young theorem, which characterizes the optimal low-rank approximation [SS90, Thm. IV.4.18].

Fact 2.1 (Optimal low-rank approximations). *Let $B = U\Sigma V^* \in \mathbb{K}^{m \times n}$ be a matrix and its SVD, and let $0 \leq k \leq \min(m, n)$ be an integer. Measured with respect to any unitarily invariant norm $\|\|\cdot\|\|$, an optimal low-rank approximation*

$$\|\|B - \widehat{B}\|\| = \min_{\mathrm{rank}(C) \leq k} \|\|B - C\|\|$$

is given by truncating the SVD to level k:

$$[\![B]\!]_k := U(:, 1:k)\Sigma(1:k, 1:k)V(:, 1:k)^*.$$

In particular, the matrix $[\![B]\!]_k$ is an optimal low-rank approximation with respect to the trace, Frobenius, and spectral norms. With respect to any of these norms, the best approximation is unique if and only if $\sigma_k(B) > \sigma_{k+1}(B)$.

Throughout this thesis, $[\![B]\!]_k$ will denote *any* optimal low-rank approximation in the sense of this theorem. As this result highlights, the best approximation may not be unique.

As Fact 2.1 demonstrates, an optimal rank-k approximation to matrix B can be computed by forming the SVD and truncating to level k. However, this approach is computationally expensive, since computing even an economy-size SVD of a matrix B requires $O(mn \min\{m, n\})$ operations. Therefore, it is natural to seek faster methods for obtaining a *near-optimal* low-rank approximation.

Before moving on, we should say a few words about low-rank approximation of a psd matrix A. When approximating a psd matrix, it is natural to use a low-rank approximation $\widehat{A} \approx A$ that is also psd. The factors F and G for a psd low-rank matrix $\widehat{A}$ can always be taken to be the same, yielding the symmetric decomposition

$$\widehat{A} = FF^*. \tag{2.2}$$

The eigendecomposition of a psd matrix induces an SVD, so an optimal rank-k approximation to a psd matrix can be obtained by truncating its eigendecomposition.

2.2 Projection approximation

How might we construct high-quality low-rank approximations without relying on the SVD? A natural strategy emerges from a re-examining the optimal rank-k approximation $[\![B]\!]_k$. In Fact 2.1, we computed the optimal rank-k approximation by truncating the SVD $B = U\Sigma V^*$. Alternately, we can compute the optimal rank-k approximation using the *projection formula*:

$$[\![B]\!]_k = QQ^*B \quad \text{for } Q = U(:, 1 : k) \tag{2.3}$$

Here, we have projected B onto the span of its k dominant left singular vectors $U(:, 1 : k)$. We remind the reader that QQ^* acts as an orthoprojector onto the range of a matrix Q with orthonormal columns.

The projection formula (2.3) motivates the use of projections as general strategy for constructing low-rank approximations.

Definition 2.2 (Projection approximation)**.** Let $\Omega \in \mathbb{K}^{n \times k}$ be a test matrix, and probe the matrix B by computing the product $B\Omega$. The *projection approximation* to B with *test matrix* Ω is

$$\widehat{B} \coloneqq \Pi_{B\Omega} B = Q(Q^*B) \quad \text{with } Q = \mathbf{Orth}(B\Omega).$$

As we will see, the class of projection approximations contains many good rank-k approximations to a matrix B. Indeed, it is simple to verify that the optimal approximation is, itself, a projection approximation with test matrix $\Omega = V(:, 1 : k)$.

The motivation behind projection approximation is that the range of $B\Omega$ serves as a good, but cheap to compute, proxy for the span of the dominant left singular vectors of B. To see this, first expand B via its SVD

$$B = \sum_{i=1}^{\min(m,n)} \sigma_i(B) u_i v_i^*$$

and consider the product

$$B\Omega = \sum_{i=1}^{\min(m,n)} \sigma_i(B) u_i (v_i^* \Omega). \tag{2.4}$$

The influence of each left singular vector u_i is controlled by the size of the singular value $\sigma_i(B)$ and the product $v_i^* \Omega$. In particular, *provided that $v_i^* \Omega$ is not small for each $1 \leq i \leq k$*, the product $B\Omega$ will have large components in the directions

of all of the dominant left singular vectors $\{\boldsymbol{u}_i : 1 \leq i \leq k\}$. Conversely, the subdominant left singular vectors $\{\boldsymbol{u}_i : i > k\}$ are scaled by smaller singular values $\sigma_i(\boldsymbol{B}) \leq \sigma_k(\boldsymbol{B})$, so $\boldsymbol{B\Omega}$ will have smaller components in these directions. Thus, projecting onto the range of $\boldsymbol{B\Omega}$ provides a computationally cheap alternative to projecting onto the span of the dominant singular vectors.

To assess the cost of projection approximation, we can count the number of *matrix–vector products* (*matvecs*) needed to compute one. Forming the product $\boldsymbol{B\Omega}$ requires k matvecs with $\boldsymbol{B}$, one with each column of $\boldsymbol{\Omega}$:

$$\boldsymbol{B\Omega} = \begin{bmatrix} \boldsymbol{B\omega}_1 & \cdots & \boldsymbol{B\omega}_k \end{bmatrix}.$$

Building the second product $\boldsymbol{Q}^*\boldsymbol{B}$ expends k matvecs with $\boldsymbol{B}^*$:

$$\boldsymbol{Q}^*\boldsymbol{B} = (\boldsymbol{B}^*\boldsymbol{Q})^* = \begin{bmatrix} \boldsymbol{B}^*\boldsymbol{q}_1 & \cdots & \boldsymbol{B}^*\boldsymbol{q}_k \end{bmatrix}^*.$$

Therefore, computing a projection approximation consists of k matvecs with $\boldsymbol{B}$, k matvecs with $\boldsymbol{B}^*$, and $O(mk^2)$ additional arithmetic operations to evaluate $\boldsymbol{Q} = \mathsf{Orth}(\boldsymbol{B\Omega})$ (via economy-size QR decomposition).

Projection approximations satisfy a number of enjoyable properties:

Proposition 2.3 (Properties of projection approximations)**.** *Let $\boldsymbol{B} \in \mathbb{K}^{m\times n}$ and $\boldsymbol{\Omega} \in \mathbb{K}^{n\times k}$ be matrices, and consider the projection approximation $\boldsymbol{\Pi}_{\boldsymbol{B\Omega}}\boldsymbol{B}$. Then*

(a) ***Columnwise orthogonality.*** *The approximation $\boldsymbol{\Pi}_{\boldsymbol{B\Omega}}\boldsymbol{B}$ and its residual $\boldsymbol{B} - \boldsymbol{\Pi}_{\boldsymbol{B\Omega}}\boldsymbol{B}$ are columnwise orthogonal,*

$$(\boldsymbol{\Pi}_{\boldsymbol{B\Omega}}\boldsymbol{B})^*(\boldsymbol{B} - \boldsymbol{\Pi}_{\boldsymbol{B\Omega}}\boldsymbol{B}) = (\boldsymbol{B} - \boldsymbol{\Pi}_{\boldsymbol{B\Omega}}\boldsymbol{B})^*(\boldsymbol{\Pi}_{\boldsymbol{B\Omega}}\boldsymbol{B}) = \boldsymbol{0}.$$

Consequently,

$$(\boldsymbol{B} - \boldsymbol{\Pi}_{\boldsymbol{B\Omega}}\boldsymbol{B})^*(\boldsymbol{B} - \boldsymbol{\Pi}_{\boldsymbol{B\Omega}}\boldsymbol{B}) = \boldsymbol{B}^*\boldsymbol{B} - (\boldsymbol{\Pi}_{\boldsymbol{B\Omega}}\boldsymbol{B})^*(\boldsymbol{\Pi}_{\boldsymbol{B\Omega}}\boldsymbol{B})$$
$$= \boldsymbol{B}^*(\boldsymbol{I} - \boldsymbol{\Pi}_{\boldsymbol{B\Omega}}\boldsymbol{B})\boldsymbol{B}.$$

(b) ***Invariance.*** *The projection approximation is invariant under right multiplication of the test matrix $\boldsymbol{\Omega}$ by a nonsingular matrix $\boldsymbol{T}$, $\boldsymbol{\Pi}_{\boldsymbol{B\Omega T}}\boldsymbol{B} = \boldsymbol{\Pi}_{\boldsymbol{B\Omega}}\boldsymbol{B}$. In particular, $\boldsymbol{\Pi}_{\boldsymbol{B\Omega}}\boldsymbol{B}$ is invariant under reordering of the columns of $\boldsymbol{\Omega}$.*

(c) ***Monotonicity.*** *The singular values of the approximation $\boldsymbol{\Pi}_{\boldsymbol{B\Omega}}\boldsymbol{B}$ are monotone increasing under extension of the test matrix:*

$$\sigma(\boldsymbol{\Pi}_{\boldsymbol{B\Omega'}}\boldsymbol{B}) \geq \sigma(\boldsymbol{\Pi}_{\boldsymbol{B\Omega}}\boldsymbol{B}) \quad \text{for } \boldsymbol{\Omega'} = [\boldsymbol{\Omega} \quad \boldsymbol{\Gamma}].$$

Similarly, the singular values of the residual $B - \Pi_{B\Omega}B$ are monotone decreasing:

$$\sigma(B - \Pi_{B\Omega'}B) \leq \sigma(B - \Pi_{B\Omega}B) \quad for \ \Omega' = [\Omega \quad \Gamma].$$

Consequently, $\||B - \Pi_{B\Omega'}B\|| \leq \||B - \Pi_{B\Omega}B\||$ *for any unitarily invariant matrix norm* $\||\cdot\||$*, such as the trace, Frobenius, and spectral norms.*

(d) **Optimality.** *The projection approximation achieves the lowest Frobenius norm error among all approximations* $C \approx B$ *satisfying* range$(C) \subseteq$ range$(B\Omega)$*:*

$$\|B - \Pi_{B\Omega}B\|_F = \min_{\text{range}(C) \subseteq \text{range}(B\Omega)} \|B - C\|_F.$$

These properties are all more-or-less standard. We omit the proof.

Remark 2.4 (Left versus right). Projection approximations approximate a matrix B by multiplying by an orthoprojector $\Pi_{B\Omega}$ on the left. One can also define *right projection approximations* which apply a projector on the right

$$B\Pi_{B^*\Psi} \quad \text{for } \Psi \in \mathbb{K}^{m \times k}.$$

Left and right projection approximations are formally equivalent, as a right projection approximation of B is the adjoint of a left projection approximation to B^*. We will use right projection approximations in Chapter 17, and we will discuss two-sided projection approximations in Chapter 10. ⋄

2.3 The randomized SVD

The randomized SVD is a simple and popular method for low-rank approximation, and its output is a projection approximation. At its simplest level, the randomized SVD approximation can be computed in four steps:

1. Generate a random matrix $\Omega \in \mathbb{K}^{n \times k}$, constructed without looking at the matrix B. (E.g., Ω could be a standard Gaussian matrix.)

2. Compute the matrix product $Y = B\Omega$.

3. Form an orthogonal basis $Q = \mathbf{Orth}(Y)$ for the column span of Y, say by QR factorization $Y = QR$.

4. Evaluate the projection $C = B^*Q$, defining the low-rank approximation $\widehat{B} = QQ^*B = QC^*$ represented by factors Q and C.

Program 2.1 `rsvd.m`. Implementation of the randomized SVD with a (standard) Gaussian test matrix.

```matlab
function [U,S,V] = rsvd(B,Bt,n,k)
% Input:  Functions B() and Bt() computing matrix products B(X) =
%         B*X and Bt(X) = B'*X, number of columns n, and rank k
% Output: Low-rank approximation Bhat to B, presented as an
%         economy-size SVD Bhat = U*S*V'

Om = randn(n,k);            % Gaussian random test matrix
Y = B(Om);                  % Matvecs with B
[Q,~] = qr(Y,"econ");       % Orthogonalize
C = Bt(Q);                  % Matvecs with B'
[UU,S,V] = svd(C',"econ");  % SVD of factor matrix
U = Q*UU;                   % Get left singular vectors

end
```

Written as so, the name "randomized SVD" is a misnomer for this algorithm, as it does not output a low-rank approximation in SVD form. If desired, one can "upgrade" the approximation $\widehat{B} = QC^*$ to compact SVD form:

5. Compute an economy-size SVD $C^* = W\widehat{\Sigma}\widehat{V}^*$ and set $\widehat{U} := QW$. The low-rank approximation $\widehat{B} = \widehat{U}\widehat{\Sigma}\widehat{V}^*$ is now expressed in compact SVD form, described by factors $\widehat{U}$, $\widehat{\Sigma}$, and $\widehat{V}$.

We recognize the output $\widehat{B} = QC^* = \widehat{U}\widehat{\Sigma}\widehat{V}^*$ of the randomized SVD as the projection approximation of B generated by test matrix Ω. The cost of steps 1–5 is k matvecs with B and k matvecs with B^*, plus an additional $O(k^2(m + n))$ operations to compute a QR decomposition of Y and compute an SVD of C^*. Code for the randomized SVD is provided in Program 2.1. In this thesis, we will use the name "randomized SVD" to refer to the low-rank approximation $\widehat{B}$, regardless of whether it is represented as $\widehat{B} = QC^*$ or $\widehat{B} = \widehat{U}\widehat{\Sigma}\widehat{V}^*$.

The randomized SVD algorithm in its modern form was introduced in the famous paper of Halko, Martinsson, and Tropp [HMT11]. See [HMT11, §2] and [TW23, §3] for a discussion of the history of this algorithm, including earlier references featuring algorithms similar to the modern randomized SVD.

The randomized SVD produces an approximation comparable with the best rank-r approximation, where r is smaller than the parameter k used in the randomized SVD algorithm. Here is an example result [TW23, Thm. 8.7] (see also [HMT11, §10]):

Fact 2.5 (Randomized SVD error analysis)**.** *Let* $\mathbf{\Omega} \in \mathbb{R}^{n \times k}$ *be populated with independent* real *standard Gaussian entries, and let* $\widehat{\mathbf{B}}$ *denote the output of the randomized SVD algorithm. Then*

$$\mathbb{E}\|\mathbf{B} - \widehat{\mathbf{B}}\|_{\mathrm{F}}^2 \le \min_{r \le k-2} \frac{k-1}{k-r-1} \|\mathbf{B} - [\![\mathbf{B}]\!]_r\|_{\mathrm{F}}^2, \tag{2.5}$$

$$\mathbb{E}\|\mathbf{B} - \widehat{\mathbf{B}}\|^2 \le \min_{r \le k-2} \frac{k+r-1}{k-r-1} \left(\|\mathbf{B} - [\![\mathbf{B}]\!]_r\|^2 + \frac{\mathrm{e}^2}{k-r} \|\mathbf{B} - [\![\mathbf{B}]\!]_r\|_{\mathrm{F}}^2 \right). \tag{2.6}$$

The first bound (2.5) shows that the expected Frobenius error of the randomized SVD is no worse with that of the best rank-r approximation for every $r \le k - 2$, up to a prefactor $f(k, r)$ depending on k and r. The second bound (2.6) demonstrates that the *spectral-norm* error of the randomized SVD still depends on the *Frobenius norm* of the best rank-r approximation

$$\|\mathbf{B} - [\![\mathbf{B}]\!]_r\|_{\mathrm{F}}^2 = \sum_{i=r+1}^{\min(m,n)} \sigma_i^2(\mathbf{B}). \tag{2.7}$$

Thus, we see that spectral-norm accuracy of the randomized SVD depends on the entire tail of singular values. Compare with the optimal approximation, whose error is just the $(r + 1)$st singular value:

$$\|\mathbf{B} - [\![\mathbf{B}]\!]_r\| = \sigma_{r+1}(\mathbf{B}).$$

The spectral-norm bound (2.6) demonstrate that the randomized SVD produces a fairly coarse approximation to a matrix $\mathbf{B}$ when its singular values decay at a slow rate. This coarse approximation is nonetheless useful for many purposes.

2.4 Randomized subspace iteration

The randomized SVD can be improved by using powering to build a better test matrix $\mathbf{\Omega}$. Fix a number $q \ge 0$ of powering steps, and assume at first that q is even. To build $\mathbf{\Omega}$, generate a random matrix $\mathbf{\Gamma} \in \mathbb{K}^{n \times k}$ and apply powering

$$\mathbf{\Omega} = (\mathbf{B}^* \mathbf{B})^{q/2} \mathbf{\Gamma}.$$

Now, form the projection approximation $\widehat{\mathbf{B}} := \mathbf{\Pi}_{\mathbf{B}\mathbf{\Omega}} \mathbf{B} = \mathbf{\Pi}_{\mathbf{B}(\mathbf{B}^*\mathbf{B})^{q/2}\mathbf{\Gamma}} \mathbf{B}$, represented as either $\widehat{\mathbf{B}} = \mathbf{Q}\mathbf{C}^*$ or $\widehat{\mathbf{B}} = \widehat{\mathbf{U}}\widehat{\mathbf{\Sigma}}\widehat{\mathbf{V}}^*$. We emphasize that the matrix $\mathbf{\Omega}$ should be formed by successive matrix multiplications

$$\mathbf{\Omega} = \mathbf{B}^*(\mathbf{B}(\mathbf{B}^*(\mathbf{B} \cdots (\mathbf{B}\mathbf{\Gamma}) \cdots))), \tag{2.8}$$

Program 2.2 `rsi.m`. Implementation of the randomized SVD with subspace iteration. Warning: This code can be numerically unstable when the matrix $\boldsymbol{B}$ has rapidly decaying singular values of the number of subspace iteration steps q is large.

```matlab
function [U,S,V] = rsi(B,Bt,m,n,k,q)
% Input:  Functions B() and Bt() computing matrix products B(X) =
%         B*X and Bt(X) = B'*X, dimensions m and n, rank k, and
%         number of subspace iteration steps q
% Output: Low-rank approximation Bhat to B, presented as an
%         economy-size SVD Bhat = U*S*V'

if mod(q,2) == 1               % Odd case
    Om = randn(m,k);           % Gaussian random test matrix with m rows
    Om = Bt(Om);               % First subspace iteration step
else                           % Even case
    Om = randn(n,k);           % Gaussian random test matrix with n rows
end

for i = 1:floor(q/2)
    Om = Bt(B(Om));            % Two subspace iteration steps
end

Y = B(Om);                     % Matvecs with B
[Q,~] = qr(Y,"econ");          % Orthogonalize
C = Bt(Q);                     % Matvecs with B'
[UU,S,V] = svd(C','econ");     % SVD of factor matrix
U = Q*UU;                      % Get left singular vectors

end
```

not by explicitly forming and powering the matrix $\boldsymbol{B}^*\boldsymbol{B}$. When q is odd, the test matrix is instead defined as

$$\boldsymbol{\Omega} = (\boldsymbol{B}^*\boldsymbol{B})^{(q-1)/2}\boldsymbol{B}^*\boldsymbol{\Gamma} \quad \text{for random } \boldsymbol{\Gamma} \in \mathbb{K}^{m\times k}.$$

This algorithm for computing low-rank approximations by powered test matrices is called *randomized subspace iteration* or the *randomized SVD with subspace iteration* [RST10; HMT11; Gu15; TW23]. Code is given in Program 2.2. The cost of randomized subspace iteration is $(q+2)k$ matvecs, split roughly evenly between matvecs with $\boldsymbol{B}$ and $\boldsymbol{B}^*$, plus $O(k^2(m+n))$ additional operations for QRs and SVDs.

Remark 2.6 (Counting subspace iteration steps). We have chosen to count the number of steps of subspace iteration q by the total numbers of matrix products with $\boldsymbol{B}$ or $\boldsymbol{B}^*$ required to form $\boldsymbol{\Omega}$. Be warned! Some authors use a different convention, counting number of multiplications with $\boldsymbol{B}^*\boldsymbol{B}$. ◇

To gain intuition for why subspace iteration helps, consider an expansion $B\Omega$ analogous to (2.4):

$$B\Omega = \sum_{i=1}^{\min(m,n)} \sigma_i(B)^{q+1} u_i(v_i^* \Gamma) \quad \text{for } q \text{ even.}$$

We see that subspace iteration has the effect of powering the singular values, boosting the gap between the "large" singular values $\sigma_1(B), \ldots, \sigma_k(B)$ and the small singular values $\sigma_i(B)$ for $i > k$. The name "subspace iteration" is derived from the *power iteration* method, which computes the dominant eigenvector or singular vector of a matrix by repeatedly multiplying by a matrix B (and possibly its adjoint B^*). The process (2.8) is referred to as *subspace iteration* because powering is performed on a matrix rather than a single vector. In subspace iteration, the object of interest is not really the powered matrix Ω itself but the subspace range(Ω).

Subspace iteration, even with a random initialization, is a classical approach in matrix computations [Par98, Ch. 14]. The modern randomized algorithms literature has sharpened our understanding of subspace iteration by providing sharp probabilistic analysis and emphasizing the computational benefits of using a large block size k with a small number q of subspace iteration steps.

The basic implementation of subspace iteration we've described can be numerically unstable, as the powered matrix Ω given by (2.8) can become singular up to numerical precision. This issue can be addressed by using intermediate orthogonalization during the powering process. That is, instead of using the plain iteration

$$\Omega \leftarrow B^*(B\Omega) \quad \text{repeated } q/2 \text{ times,}$$

as in Program 2.2, orthonormalize after each step:

$$\Omega \leftarrow \mathbf{Orth}(B^*(B\Omega)) \quad \text{repeated } q/2 \text{ times.}$$

(Being even more aggressive, one could even use the update rule $\Omega \leftarrow \mathbf{Orth}(B^* \mathbf{Orth}(B\Omega))$.) *In exact arithmetic*, subspace iteration with and without intermediate orthogonalization produce the same projection approximation as output, in view of the invariance property Proposition 2.3(b). In floating-point arithmetic, intermediate orthogonalization can significantly improve the quality of the computed projection approximation.

Error bounds for randomized subspace iteration, analogous to Fact 2.5, are well-established. See [HMT11; Gu15; TW23].

Remark 2.7 (Block Krylov iteration). An even more powerful type of projection approximation is given by block Krylov iteration, where one uses the entire family of powered approximations to define the test matrix $\mathbf{\Omega}$, i.e.,

$$\mathbf{\Omega} = \begin{bmatrix} \mathbf{\Gamma} & (\mathbf{B}^*\mathbf{B})\mathbf{\Gamma} & \cdots (\mathbf{B}^*\mathbf{B})^{q/2}\mathbf{\Gamma} \end{bmatrix} \quad \text{for } q \text{ even.}$$

References on randomized block Krylov iteration include [RST10; MM15; TW23]. The weaker approximations produced by the randomized SVD, possibly with a few steps of subspace iteration, will suffice for the purpose of this thesis. ⬦

2.5 Nyström approximation

To approximate a psd matrix $\mathbf{A} \in \mathbb{K}^{n \times n}$, we have access to a more efficient class of low-rank approximations known as Nyström approximations.

Definition 2.8 (Nyström approximation). Let $\mathbf{A} \in \mathbb{K}^{n \times n}$ be a psd matrix, and let $\mathbf{\Omega} \in \mathbb{K}^{n \times k}$ be a test matrix. The Nyström approximation to $\mathbf{A}$ is

$$\widehat{\mathbf{A}} = \mathbf{A}\langle \mathbf{\Omega} \rangle = \mathbf{A}\mathbf{\Omega}(\mathbf{\Omega}^*\mathbf{A}\mathbf{\Omega})^\dagger(\mathbf{A}\mathbf{\Omega})^*. \tag{2.9}$$

Here, † is the Moore–Penrose pseudoinverse, which agrees with the ordinary inverse for nonsingular matrices.

Observe that the only access to the matrix $\mathbf{A}$ needed to compute the Nyström approximation $\mathbf{A}\langle \mathbf{\Omega} \rangle$ is the ability to form the single matrix product

$$\mathbf{Y} := \mathbf{A}\mathbf{\Omega}. \tag{2.10}$$

From $\mathbf{Y}$, the approximation $\widehat{\mathbf{A}}$ may be assembled using the formula

$$\widehat{\mathbf{A}} = \mathbf{Y}(\mathbf{\Omega}^*\mathbf{Y})^\dagger\mathbf{Y}^*.$$

This "single-pass" property of the Nyström approximation is an advantage over projection approximations, which require two passes over the matrix $\mathbf{B}$ (one to compute $\mathbf{B}\mathbf{\Omega}$, a second to compute $\mathbf{Q}^*\mathbf{B}$).

To motivate the form of the Nyström approximation (2.9), observe that any approximation $\widehat{\mathbf{A}}$ satisfying $\mathrm{range}(\widehat{\mathbf{A}}) = \mathrm{range}(\mathbf{A}\mathbf{\Omega})$ must take the form

$$\widehat{\mathbf{A}} = (\mathbf{A}\mathbf{\Omega})\mathbf{M}(\mathbf{A}\mathbf{\Omega})^*.$$

The choice $\mathbf{M} = (\mathbf{\Omega}^*\mathbf{A}\mathbf{\Omega})^\dagger$ enforces that the matrix $\mathbf{A}$ and the approximation $\widehat{\mathbf{A}}$ agree when multiplied by $\mathbf{\Omega}$, $\widehat{\mathbf{A}}\mathbf{\Omega} = \mathbf{A}\mathbf{\Omega}$. Indeed, provided $\mathbf{\Omega}^*\mathbf{A}\mathbf{\Omega}$ is nonsingular, $\mathbf{M} = (\mathbf{\Omega}^*\mathbf{A}\mathbf{\Omega})^\dagger = (\mathbf{\Omega}^*\mathbf{A}\mathbf{\Omega})^{-1}$ is the *unique* choice of $\mathbf{M}$ satisfying this condition.

The randomized Nyström approximation enjoys many nice properties:

Proposition 2.9 (Properties of the Nyström approximtion). *Let $A \in \mathbb{K}^{n \times n}$ be psd, let $\Omega \in \mathbb{K}^{n \times k}$ be a matrix, and consider the projection approximation $A\langle\Omega\rangle$. Then*

(a) **Psd.** *The Nyström approximation $A\langle\Omega\rangle$ and its residual $A - A\langle\Omega\rangle$ are psd.*

(b) **Invariance.** *The Nyström approximation is invariant under right multiplication of the test matrix Ω by a nonsingular matrix T, $A\langle\Omega T\rangle = A\langle\Omega\rangle$. In particular, $A\langle\Omega\rangle$ is invariant to reordering of the columns of Ω.*

(c) **Monotonicity.** *The Nyström approximation is monotone increasing with respect to the psd order under enlargement of the matrix Ω*

$$A\,\langle[\Omega \;\; \Gamma]\rangle \succeq A\langle\Omega\rangle.$$

Recall $\succeq$ denotes the psd order on Hermitian matrices. Consequently,

$$\||A - A\,\langle[\Omega \;\; \Gamma]\rangle\|| \leq \||A - A\langle\Omega\rangle\||$$

for any unitarily invariant norm $\||\cdot\||$, such as the trace, Frobenius, or spectral norms.

(d) **Interpolatory.** *The matrix A and its Nyström approximation $A\langle\Omega\rangle$ have the same action on Ω. That is, $A \cdot \Omega = A\langle\Omega\rangle \cdot \Omega$.*

(e) **Optimality.** *Among all Hermitian approximations M satisfying $\mathrm{range}(M) \subseteq \mathrm{range}(A\Omega)$ with a psd residual $A - M$, the residual $A - A\langle\Omega\rangle$ is the smallest in the psd order:*

$$A - \widehat{A} \preceq A - M.$$

Consequently, $\||A - A\langle\Omega\rangle\|| \leq \||A - M\||$ for all such M and for any unitarily invariant norm $\||\cdot\||$.

These properties have all been used, implicitly or explicitly, throughout the literature. The first four properties all have relatively straightforward proofs, and the last property relies on variational properties of Schur complements [And05, Thm. 5.3]; see [Epp22a] for an exposition.

Remark 2.10 (Hermitian indefinite matrices). While it is most naturally justified for psd matrices, the Nyström approximation can also be used to approximate any Hermitian matrix A. A potential issue is that, for an indefinite matrix A (i.e., a matrix for which neither A or $-A$ is psd), the core matrix $\Omega^* A \Omega$ can be singular (or

nearly singular) even if the input matrix A and test matrix Ω are both full-rank. This near-singularity issue can cause degradations in the accuracy of the approximation $A\langle\Omega\rangle \approx A$. This issue is addressed by Nakatsukasa and Park [NP23], who suggest *oversampling* by using a larger test matrix of size $\Omega \in \mathbb{K}^{n \times ck}$, then regularizing the Nyström approximation by truncating the small eigenvalues of the core matrix

$$\widehat{A} := A\Omega[\![\Omega^* A\Omega]\!]_k^\dagger (A\Omega^*).$$

They suggest possible values $c = 1.5$ and $c = 2$. This strategy appears to resolve the numerical issues of applying the ordinary, un-regularized Nyström approximation to Hermitian indefinite matrices, though only partial theoretical explanation for the success of this strategy is available [NP23, Thm. 3.1]. A disadvantage of this strategy it increases the cost of computing a rank-k Nyström approximation to ck matvecs, comparable to the cost of a randomized SVD ($2k$ matvecs) or generalized Nyström approximation ($\approx 2.5k$ matvecs); both of these approximations $\widehat{B} \approx A$ are not Hermitian by default, but can be made Hermitian by symmetrizing $\widehat{A} := (\widehat{B} + \widehat{B}^*)/2$. I believe there is more left to be understood about what the "right" algorithm is for approximation of Hermitian indefinite matrices.   ◇

Choice of test matrix. Analogous to the randomized SVD, a simple way of invoking the Nyström approximation is to choose Ω to be a random matrix independent from A such as a standard Gaussian matrix. We will call this version of Nyström approximation a *single-pass Nyström approximation*, since it requires only one pass over the matrix to compute. Alternatively, one can use subspace iteration [RST10; HMT11; Gu15]

$$\Omega = A^q\Gamma$$

or block Krylov iteration [RST10; MM15; TW23]

$$\Omega = \begin{bmatrix} \Gamma & A\Gamma & \cdots & A^q\Gamma \end{bmatrix},$$

both of which require additional passes over the matrix. All of the warnings about numerical stability and reorthogonalization from Section 2.4 remain in force when randomized subspace iteration (or block Krylov iteration) is combined with Nyström approximation. Part I of this thesis will explore a special class of Nyström approximations where the matrix Ω is a column submatrix of the identity matrix.

Stable implementation. Computing the Nyström approximation must be done with care to ensure accurate results in floating-point arithmetic. Here, we present a

variant of the stable Nyström implementation developed in [TYUC17a] (based on ideas introduced in [LLSS+17]). The idea is to compute a Nyström approximation

$$\widehat{A}_\mu := (A + \mu I)\langle \Omega \rangle$$

of a shifted matrix $A + \mu I$, where μ is a small shift parameter. Begin by computing the matrix Y in (2.10), and define the shift

$$\mu := n^{-1/2}\|Y\|_{\mathrm{F}} u. \tag{2.11a}$$

Here, u denotes the unit roundoff, of size $u \approx 10^{-16}$ in double-precision arithmetic. The shift (2.11a) differs from the one proposed in [TYUC17a] and was introduced in [ETW24] to obtain a stable shift of the minimum possible size while avoiding computing the spectral norm of Y. Now, apply the shift to Y, obtaining

$$Y_\mu := Y + \nu \Omega, \tag{2.11b}$$

and form the matrix

$$H := \Omega^* Y_\mu. \tag{2.11c}$$

Next, compute a Cholesky decomposition

$$H = R^* R \tag{2.11d}$$

and use triangular substitution to form

$$F := YR^{-1}. \tag{2.11e}$$

The factor matrix F gives rise to the shifted Nyström approximation $\widehat{A}_\mu = FF^*$. Code is provided in Program 2.3. In order to reuse this code later in the thesis, we have written it to use a matrix $\Omega \sim \mathrm{U_{NIF}}\{\pm 1\}^{n \times k}$ of random ± 1 values and to return the shift μ, test matrix Ω, and Cholesky factor R as optional outputs.

For many purposes, the shifted Nyström approximation $\widehat{A}_\mu = FF^*$ is a perfectly good substitute for the unshifted approximation $\widehat{A}$, as the shift parameter μ is tiny—on the order of the unit roundoff. To achieve the most accurate results, however, we can attempt to correct for the shift. For some problems, we will have means to correct the shift "exactly"; see Section 14.6 for an example of such a scenario. Alternately, we can upgrade the outer product representation $\widehat{A}_\mu = FF^*$ to an eigendecomposition representation $\widehat{A}_\mu = UD_\mu U^*$ via economy-size SVD $F = U\Sigma V^*$ with $D_\mu := \Sigma^2$. Then, define the shift-corrected Nyström approximation

$$\widehat{A}_{\mathrm{SC}} = UDU^* \quad \text{for } D = \max\{D_\mu - \mu I, 0\}.$$

Program 2.3 `nystrom.m`. Stable implementation of the single-pass Nyström approximation. Low-rank approximation is outputted in the form $\boldsymbol{FF}^*$ and computed using shifting. The `random_signs` subroutine is defined in Program F.2.

```
function [F,mu,Om,R] = nystrom(A,n,k)
% Input:  Function A() computing matrix products A(X) = A*X, number
%         of rows n, rank k
% Output: Shifted Nystrom approximation Ahat = F*F' represented by
%         factor F, shift mu, test matrix Om, triangular factor R

Om = random_signs(n,k);         % Test matrix of random signs
Y = A(Om);                      % Matrix product Y = A*Om
mu = eps*norm(Y,"fro")/sqrt(n); % Compute shift
Y = Y + mu * Om;                % Apply shift to Y
H = Om'*Y;
R = chol(H);
F = Y/R;                        % Triangular substitution

end
```

Program 2.4 `nystrom_shiftcor.m`. Stable implementation of the single-pass Nyström approximation $\boldsymbol{UDU}^*$ using shift correction. The `nystrom` subroutine is defined in Program 2.3.

```
function [U,D] = nystrom_shiftcor(A,n,k)
% Input:  Function A() computing matrix products A(X) = A*X, number
%         of rows n, rank k
% Output: Shift-corrected Nystrom approximation Ahat = U*D*U',
%         represented by orthonormal factor U and nonnegative
%         diagonal factor D

[F,mu] = nystrom(A,n,k); % Compute (shifted) Nystrom approximation
[U,S,~] = svd(F,"econ"); % Economy-size SVD
D = max(S.^2 - mu, 0);   % Apply shift correction

end
```

The maximum is taken entrywise. The shift-corrected Nyström approximation $\widehat{A}_{\mathrm{SC}}$ is not exactly equal to the original Nyström approximation $\widehat{A}$, but it is typically more accurate then the uncorrected approximation $\widehat{A}_\mu$. Code for the shift-corrected Nyström approximation (with a standard Gaussian test matrix $\boldsymbol{\Omega}$) is provided in Program 2.4.

Nyström versus projection approximation. Measured using the Frobenius norm, the projection approximation $\boldsymbol{\Pi}_{A\boldsymbol{\Omega}}A$ is more accurate than the Nyström approximation

$A\langle\mathbf{\Omega}\rangle$,

$$\|A - \mathbf{\Pi}_{A\mathbf{\Omega}}A\|_{\mathrm{F}} \le \|A - A\langle\mathbf{\Omega}\rangle\|_{\mathrm{F}}. \tag{2.12}$$

This conclusion follows from the optimality property Proposition 2.3(d) of the projection approximations. However, this comparison is not really a fair one since the Nyström approximation $A\langle\mathbf{\Omega}\rangle$ can be computed using k matvecs and a single pass over the matrix, whereas the projection approximation $\mathbf{\Pi}_{A\mathbf{\Omega}}A$ requires $2k$ matvecs and two passes. Therefore, the fair comparison is between the projection approximation $\mathbf{\Pi}_{A\mathbf{\Omega}}A$ and Nyström approximation computed with one step of subspace iteration $A\langle A\mathbf{\Omega}\rangle$. In this case, a result of Tropp and Webber [TW23, Lem. 5.2] ensures the Nyström approximation is always more accurate:

$$\|\|A - A\langle A\mathbf{\Omega}\rangle\|\| \le \|\|A - \mathbf{\Pi}_{A\mathbf{\Omega}}A\|\| \tag{2.13}$$

for any unitarily invariant norm.

Error analysis. Analogous to Fact 2.5, we have error bounds for the single-pass Nyström approximation [TW23, Cor. 8.8]:

Fact 2.11 (Single-pass Nyström error analysis)**.** *Let* $\mathbf{\Omega} \in \mathbb{R}^{n\times k}$ *be populated with independent (real) standard Gaussian entries, and let* $\widehat{A}$ *denote the single-pass Nyström approximation with test matrix* $\mathbf{\Omega}$*. Then*

$$\mathbb{E}\|A - \widehat{A}\|_* \le \min_{r \le k-2} \frac{k-1}{k-r-1}\|A - [\![A]\!]_r\|_*, \tag{2.14}$$

$$\left(\mathbb{E}\|A - \widehat{A}\|_{\mathrm{F}}^2\right)^{1/2} \le \min_{r \le k-4} \frac{k-2}{k-r-1}\left(\|A - [\![A]\!]_r\|_{\mathrm{F}} + \frac{\|A - [\![A]\!]_r\|_*}{\sqrt{k-r}}\right), \tag{2.15}$$

$$\left(\mathbb{E}\|A - \widehat{A}\|^2\right)^{1/2} \le \min_{r \le k-4} \frac{k+r-1}{k-r-3}\left(\|A - [\![A]\!]_r\| + \frac{\sqrt{3}\mathrm{e}^2}{k-r}\|A - [\![A]\!]_r\|_*\right). \tag{2.16}$$

With these error bounds, we can again consider the question: Which is more accurate, the randomized SVD (a projection approximation) or the single-pass randomized Nyström approximation? Recall that the error of the randomized SVD, measured in both the Frobenius and spectral norms, depends on the *Frobenius norm* of the best rank-r approximation (2.7). By contrast, the error of the single-pass Nyström approximation—measured in either the trace, Frobenius, or spectral norm—depends on the *trace norm* of the best rank-r approximation

$$\|A - [\![A]\!]_r\|_* = \sum_{i=r+1}^{n} \lambda_i(A).$$

The trace norm is always larger than the Frobenius norm. Therefore, when single-pass Nyström approximation and the randomized SVD are both used to produce a rank-k approximation to a psd matrix A, the randomized SVD is usually more accurate. However, measured in matvecs, randomized Nyström approximation (k matvecs) is cheaper than the randomized SVD ($2k$ matvecs). Thus, for a fixed budget of s matvecs, one can either compute a rank-s randomized Nyström approximation or a rank-$(s/2)$ randomized SVD; because of the higher approximation rank, the former is often preferable to the latter. The consequences of this comparison will be explored in Part II of this thesis.

2.6 The Gram correspondence

The *Gram correspondence* is a powerful fact that links projection approximations and Nyström approximations.

Theorem 2.12 (Gram correspondence). *Let $A \in \mathbb{K}^{n \times n}$ be a psd matrix, and let $B \in \mathbb{K}^{m \times n}$ be any matrix for which $A = B^*B$. For a given test matrix $\Omega \in \mathbb{K}^{n \times k}$, instantiate the projection approximation $\widehat{B} := \Pi_{B\Omega}B$ and Nyström approximation $\widehat{A} := A\langle\Omega\rangle$. These approximations are related $\widehat{A} = \widehat{B}^*\widehat{B}$.*

Proof. Since $A = B^*B$, the Nyström approximation is

$$A\Omega = B^*[(B\Omega)((B\Omega)^*(B\Omega))^{\dagger}(B\Omega)^*]B$$

Observe that the bracketed matrix is a formula for the projector $\Pi_{B\Omega}$, which equals its square. Therefore,

$$A\Omega = B^*\Pi_{B\Omega}^2 B = (\Pi_{B\Omega}B)^*(\Pi_{B\Omega}B) = \widehat{B}^*\widehat{B}.$$

We have obtained the advertised conclusion. □

The Gram correspondence has an interesting history, which we will describe later in this section. Early references in the randomized matrix computations literature include [BW09a; Git11].

The Gram correspondence can be stated more concisely by using the following definitions.

Definition 2.13 (Gram matrix and Gram square root). Let $A \in \mathbb{K}^{n \times n}$ be a psd matrix. *Any* matrix $B \in \mathbb{K}^{m \times n}$ for which $B^*B = A$ is called a *Gram square root* of A. Similarly, the matrix $A = B^*B$ is called the *Gram matrix* of B.

The Gram matrix, $A = B^*B = (b_i^* b_j)_{1 \leq i,j \leq n}$, named after Jørgen Pedersen Gram of Gram–Schmidt fame, tabulates the pairwise inner products of a matrix B. Every psd matrix A has many Gram square roots, including the output R of a (pivoted) Cholesky factorization and the positive-semidefinite matrix square root $A^{1/2}$.

Using Definition 2.13, the Gram correspondence may be rewritten:

> **Gram correspondence (rephrased).** If B is a Gram square root of A, the projection approximation $\widehat{B} := \mathbf{\Pi}_{B\Omega}B$ is a Gram square root of the Nyström approximation $\widehat{A} := A\langle\Omega\rangle$. (Just as well, the Nyström approximation $\widehat{A}$ is the Gram matrix of the projection approximation $\widehat{B}$.)

Several important observations are special cases of the Gram correspondence, including the equivalence of single-pass Nyström approximation and the randomized SVD and the equivalence of column-pivoted QR and Cholesky decompositions. The latter equivalence will play an important role in Part I of this thesis.

The Gram correspondence has important implications for algorithm design. At a high-level, the principle is as follows:

> **Gram correspondence: Transference of algorithms.** Every algorithm producing a projection approximation to a general matrix should have an analogous algorithm that produces a Nyström approximation to a psd matrix.

An elementary example of a pair of algorithms are the randomized SVD for approximating a general matrix and the single-pass Nyström approximation for approximating a psd matrix. More sophisticated examples of this principle will be explored in Part I of thesis, most particularly in Chapter 9.

The Gram correspondence also has a consequence for error analysis of algorithms:

Corollary 2.14 (Gram correspondence: Transference of error bounds). *Let $B \in \mathbb{K}^{m \times n}$ be a Gram square root of a psd matrix $A \in \mathbb{K}^{n \times n}$, and fix a test matrix $\Omega \in \mathbb{K}^{n \times k}$. Then the errors for the Nyström approximation $\widehat{A} = A\langle\Omega\rangle$ and projection approximation $\widehat{B} = \mathbf{\Pi}_{B\Omega}B$ are related: For any $p \geq 1$, we have*

$$\|A - \widehat{A}\|_{\mathsf{S}_p} = \|B - \widehat{B}\|_{\mathsf{S}_{2p}}^2. \tag{2.17}$$

Here, $\|\cdot\|_{\mathsf{S}_p}$ denotes the Schatten p-norm, of which the trace, Frobenius, and spectral norms are the special cases $p = 1$, $p = 2$, and $p = \infty$. The special cases $p = 1$ and

p = ∞ in (2.17) *yield the useful relations*

$$\|A - \widehat{A}\|_* = \mathrm{tr}(A - \widehat{A}) \quad = \|B - \widehat{B}\|_F^2,$$
$$\|A - \widehat{A}\| \qquad\qquad\quad = \|B - \widehat{B}\|^2.$$

Proof. By the columnwise orthogonality property Proposition 2.3(a),

$$(B - \widehat{B})^*(B - \widehat{B}) = B^*B - \widehat{B}^*\widehat{B} = A - \widehat{A}.$$

Take Schatten p-norms of both sides, and invoke the identity $\|C^*C\|_{\mathsf{S}_p} = \|C\|_{\mathsf{S}_{2p}}^2$ to obtain (2.17). □

This for any given (random) test matrix $\mathbf{\Omega}$, one only has to analyze the error $\|B - \mathbf{\Pi}_{B\mathbf{\Omega}}\|_{\mathsf{S}_{2p}}$ or $\|A - A\langle\mathbf{\Omega}\rangle\|_{\mathsf{S}_p}$ once, with a bound on the other quantity coming for free. We will use fact several times in this thesis. As an example we have already seen, the pair of bounds (2.5) and (2.14) can be derived from each other in this way.

Remark 2.15 (History). The Gram correspondence is implicit in much of the literature on low-rank approximation and pivoted matrix decompositions [Hig90; BW09a; Git11; GS12; GM13; MW17; TW23; PBK25]. The equivalence between Cholesky and QR decompositions, a consequence of the Gram correspondence, is classical [Hig90]. Transferring algorithms and analysis between a matrix B and its Gram matrix $A = B^*B$ has been a standard technique for column-based matrix decompositions and approximations over many years [Hig90; BW09a; GS12; CK24]; see Chapter 9 for examples and discussion. A version of the Gram correspondence for the particular Gram square root $B := A^{1/2}$ appears in works of Gittens [Git11; GM13] starting in 2011, and this analytical approach is used to obtain sharp bounds for single-pass Nyström approximations in [TYUC17a]. Musco and Woodruff [MW17] provide a clear statement of the transference of algorithms principle. Some aspects of the Gram correspondence are highlighted by Dereziński, Khanna, and Mahoney in [DKM20, Rem. 2], who draw attention to algorithmic implications in the work of Belabbas and Wolfe [BW09a]. The connection between the randomized SVD and randomized Nyström approximation has been used in a very explicit and direct way in recent papers [TW23; PBK25]. In an effort to make Theorem 2.12 and its consequences known beyond the community of experts familiar with it, I described the principle in a general way and suggested the name *Gram correspondence* in the blog post [Epp24c]. ◇

Part I

Random pivoting

Dedicated to my fiancée Sierra, our dogs Hulk and Finn, and our turtle Shelly.

C h a p t e r 3

LOW-RANK APPROXIMATION OF PSD MATRICES

> *Symmetric positive definiteness is one of the highest accolades to which a matrix can aspire.*
>
> Nicholas J. Higham, *Accuracy and Stability of Numerical Algorithms*
> [Hig02, §10.1]

The first part of this thesis will largely be concerned with the low-rank approximation of *psd* matrices, though we will return to general matrices in Chapters 9 and 10. We will focus on a very limited model of computation where we only have access to a small number of *entries* of the input matrix $A \in \mathbb{K}^{n \times n}$. The main algorithm of this part of the thesis will be *randomly pivoted Cholesky*, which produces near-optimal rank-k approximations to a psd matrix after reading only $(k+1)n$ entries. Our main application for psd low-rank approximation will involve efficient computations with kernel matrices and covariance matrices of Gaussian processes. These matrices are the core objects in a wide class of machine learning algorithms; see Chapter 5 for an introduction to kernel and Gaussian methods in machine learning.

Sources. This chapter largely serves to introduce the psd low-rank approximation problem and summarize the existing literature. It is a significantly extended version of the literature survey from the following paper:

Yifan Chen, Ethan N. Epperly, Joel A. Tropp, and Robert J. Webber. "Randomly Pivoted Cholesky: Practical Approximation of a Kernel Matrix with Few Entry Evaluations". In: *Communications on Pure and Applied Mathematics* 78.5 (2025), pp. 995–1041. DOI: 10.1002/cpa.22234.

Outline. Sections 3.1 and 3.2 discuss the entry access model for matrix computations and the positive-semidefinite low-rank approximation problem. Section 3.3 introduces pivoted partial Cholesky decompositions and shows how they can be used to compute low-rank approximations to psd matrices; the outputs of pivoted partial Cholesky decompositions are called *column Nyström approximations*, which are discussed in Section 3.4. Section 3.5 describes subset selection problems in machine learning and computational mathematics and relates them to the psd low-rank

approximation task. Section 3.6 describes a connection between Cholesky decomposition, Nyström approximation, and Gaussian random variables. Section 3.7 concludes with a discussion of sampling methods for psd low-rank approximation, which are some of the main alternatives to the randomly pivoted Cholesky method.

3.1 The entry access model

In computational linear algebra, we usually work with matrices stored directly in memory, with all of the matrix entries immediately available to us to perform whatever operations we so choose. There are also computational settings where we have much more limited access to the matrix, introducing constraints on algorithm design that must be accommodated. The first part of this thesis will work in one such limited framework, the entry access model.

> **Entry access model.** We are given a matrix $B \in \mathbb{K}^{m \times n}$ that may be accessed by requesting individual entries b_{ij}. The (dominant) cost of an algorithm is the total number of entries accessed.

The Part II of this thesis works in a different computational model, the matvec model, which is a natural model for other types of computational linear algebra problems.

The following definition provides a natural example of a matrix for which the entry access model is appropriate:

Definition 3.1 (Function matrix). Let $D \subseteq X$ and $E \subseteq Y$ be finite subsets of base sets X and Y, and let $\xi : X \times Y \to \mathbb{K}$ be a bivariate function. The *function matrix* associated with this data is $B = \xi(D, E) \coloneqq (\xi(x, y) : x \in D, y \in E) \in \mathbb{K}^{D \times E}$.

A function matrix B is described implicitly by the elements $x \in D, y \in E$ and the function ξ. Accessing each entry of B requires computing the function $\xi(x, y)$. In particular, reading all entries of B requires $|D| \cdot |E|$ function evaluations, much greater than the number $|D| + |E|$ of input elements $D \cup E$. Function matrices, and slight variations thereof, occur in discretizations of integral operators in computational physics [SS11] and in the design of fast algorithms for classical structured matrices [CGSX+08; Wil21].

The main motivating example for this part of the thesis will be kernel matrices, a subclass of function matrices.

Definition 3.2 (Kernel function and matrix). Let X be a set. A function $\kappa : \mathsf{X} \times \mathsf{X} \to$ $\mathbb{K}$ is said to be a *(positive definite) kernel function* if the *kernel matrix* $A \coloneqq \kappa(\mathsf{D}, \mathsf{D})$ is psd for every finite subset $\mathsf{D} \subseteq \mathsf{X}$.

Kernel functions and kernel matrices are flexible tools that can be used to design algorithms for learning from data, and they also appear in the theory of Gaussian processes. See Chapter 5 for examples of kernel functions and an introduction to kernel and Gaussian process methods in machine learning.

Like other function matrices, an $n \times n$ kernel matrix A is defined by a dataset D, say, of size $|\mathsf{D}| = n$, but generating all of the entries of A requires $n \times n = n^2$ function evaluations. Particularly when the base space $\mathsf{X} = \mathbb{K}^d$ is a Euclidean space of large dimension $d \gg 1$, each entry evaluation of A is expensive, which motivates the search for algorithms for psd low-rank approximation that require a small number of entry evaluations.

Even for the examples we have seen so far, function matrices and kernel matrices, the entry access model might not be the most appropriate abstraction for algorithm design; see Section 8.1 for a discussion of an alternative model for these use cases.

Low-rank approximation in the entry access model

On its face, it may seem impossible to accurately approximate a matrix from a limited number of entry accesses, at least without prior information. In general, this intuition is correct:

Proposition 3.3 (Impossibility of general matrix approximation from entry accesses). *Let $\|\|\cdot\|\|$ denote either the spectral, Frobenius or trace norms (or, indeed, any unitarily invariant matrix norm). Consider an algorithm that queries an input matrix B in t positions and outputs an approximation $\widehat{B}$ to B. Any such algorithm applied to a matrix B with a single nonzero entry in a random position must produce an approximation of high relative error*

$$\frac{\|\|B - \widehat{B}\|\|}{\|\|B\|\|} \geq \frac{1}{2} \tag{3.1}$$

with probability at least $1 - t/mn - 1/(mn - t)$. In particular, even querying half of the matrix entries ($t = mn/2$) still produces a poor approximation (3.1) with probability at least $1/2 - 2/mn$.

This result establishes that even an algorithm that reads a large fraction of a general matrix B's entries (say, half), still is prone to producing an approximation of high

error for some inputs. The failure mode is intuitive; if a single large entry is placed in B at a random position, no algorithm can be guaranteed to find it without exhuming a large number of entries. This observation dates back to the earliest days of randomized matrix approximation [FKV98].

Proof of Proposition 3.3. Let $B \in \{0, 1\}^{m \times n}$ be a random matrix constructed by placing a single nonzero entry in a uniformly random position,

$$B = \mathbf{e}_{i_\star} \mathbf{e}_{j_\star}^* \quad \text{for } i_\star \sim \text{Unif}\{1, \dots, m\}, \ j_\star \sim \text{Unif}\{1, \dots, n\}.$$

Without loss of generality, we take this nonzero entry to have value 1. Consider any deterministic algorithm which queries this matrix B at t adaptively chosen positions $(i_1, j_1), \dots, (i_t, j_t)$. Except with probability t/mn, the algorithm only queries entries with value 0. Conditional on the locations, B is equally likely to be any matrix in the collection $\mathsf{C} \coloneqq \{\mathbf{e}_i \mathbf{e}_j^* : (i, j) \notin \mathsf{P}\}$. For the output $\widehat{B}$ of the algorithm, at most one point in C is within distance $(1/2)\|\!|B|\!\|$ of B. Therefore, except with probability $1/(mn - t)$, $\|\!|B - \widehat{B}|\!\| / \|\!|B|\!\| \geq 1/2$. $\qquad\square$

Remarkably, the psd low-rank approximation problem is much better behaved in the entry access model than the general low-rank approximation problem, and we will see several examples in this thesis of algorithms that produce near-optimal low-rank approximations to a psd matrix while reading a fraction its entries.

The main structural property that makes psd low-rank approximation feasible in the entry access model is the off-diagonal inequality [HJ12, 7.1.P1]:

Fact 3.4 (Off-diagonal inequality). *Let $A \in \mathbb{K}^{n \times n}$ be a psd matrix, and let $1 \leq i, j \leq n$ be indices. Then the magnitude $|a_{ij}|$ of the off-diagonal entry is bounded by the geometric mean of the diagonal entries:*

$$|a_{ij}| \leq a_{ii}^{1/2} \cdot a_{jj}^{1/2} \leq \max(a_{ii}, a_{jj}).$$

In particular, the largest entry of a psd matrix must occur on its diagonal.

Proof. The proof is standard and beautiful. Since A is psd, the principal submatrix

$$A(\{i, j\}, \{i, j\}) = \begin{bmatrix} a_{ii} & a_{ij} \\ \overline{a_{ij}} & a_{jj} \end{bmatrix} \quad \text{is psd as well.}$$

Therefore, the determinant $\det A(\{i, j\}, \{i, j\}) = a_{ii}a_{jj} - |a_{ij}|^2 \geq 0$ is nonnegative. Rearrange to obtain the stated conclusion. $\qquad\square$

The off-diagonal inequality shows that large entries cannot "hide" in a psd matrix. By generating the n entries of the diagonal, one obtains a "heat map" of all possible places a large entry of A can lie. In particular, large entries of A can only exist in columns of a_i containing a large diagonal entry $a_{ii} \gg 0$. This observation suggests a strategy for psd low-rank approximation: Extract columns of A with large diagonal entries. This strategy forms the basis for the most effective methods for psd low-rank approximation in the entry access model.

Remark 3.5 (Algorithms for general low-rank approximation from entry accesses). The impossibility result Proposition 3.3 has not stopped research into algorithms for low-rank approximation of non-psd matrices that use a small number of entry accesses. In order to approximate a general matrix from a small number of entry accesses, one needs either *additional information* (such as the location of large entries, the norms of columns, etc.) or *additional assumptions* (such as *incoherence*, the property that the information in the matrix is "evenly spread across the rows/columns"). Discussion of such methods is beyond the scope of this thesis; see [CD13; CY25] for more information. ◇

3.2 The psd low-rank approximation problem

This thesis will consider the following version of the psd low-rank approximation:

> **Psd low-rank approximation problem (entry access model):** Given a psd matrix $A \in \mathbb{K}^{n \times n}$ and a target rank $k \geq 1$, compute *the description* of a *nearly optimal* rank-k psd approximation $\widehat{A}$ to A.

This problem statement contains embedded in it two phrases that require elaboration: "the description" and "nearly optimal". Let us begin with the former. For most of this thesis, "the description" of a psd rank-k approximation $\widehat{A} = FF^*$ will be provided by a factor matrix $F \in \mathbb{K}^{n \times k}$. Our approximations will be randomized, so $\widehat{A}$ will be a random matrix.

To substantiate the phrase "nearly optimal", we employ the following definition:

Definition 3.6 ((r, ε, p)-approximation). Let $B \in \mathbb{K}^{m \times n}$ be a matrix, let $1 \leq r \leq \min(m, n)$ be an integer, and $p \geq 1$ and $\varepsilon \geq 0$ be real numbers. An (r, ε, p)-*approximation* is a random matrix $\widehat{B}$ for which

$$\mathbb{E}\|B - \widehat{B}\|_{\mathsf{S}_p} \leq (1 + \varepsilon)\|B - [\![B]\!]_r\|_{\mathsf{S}_p}. \tag{3.2}$$

Here, $\|C\|_{\mathsf{S}_p} := (\sum_i \sigma_i(C)^p)^{1/p}$ denotes the Schatten p-norm. If $p = 1$, we suppress the dependence on p and refer to simply an (r, ε)-*approximation*.

This definition motivates the question "for a particular algorithm, for what value of k can we guarantee it produces an (r, ε)-approximation?" As we will see, the algorithms we consider will achieve a nearly optimal dependence of k on r and ε in a precise sense outlined below in Fact 3.11. This will be the sense in which the algorithms considered by this thesis will be "near-optimal".

Let us now speak to the choice of $p = 1$ in the definition of an (r, ε)-approximation. First let us note that the problem of computing an (r, ε, p) becomes more difficult as p becomes larger. As a simple example, consider an $n \times n$ matrix with eigenvalues 10 and 1, the latter with multiplicity $n - 1$. For this matrix, even the zero matrix is a near-optimal rank-1 approximation for $p = 1$ in the sense that it is a $(1, O(1/n), 1)$-approximation; even the most trivial approximation, the zero matrix, obtains an accuracy parameter ε vanishing in the limit $n \to \infty$. However, for $p = \infty$, the zero matrix is merely a $(1, 9, \infty)$ approximation, and we must work harder to obtain a small accuracy parameter ε.

The example above demonstrates that the psd low-rank approximation problem gets harder as p gets larger, but why $p = 1$ specifically? Why not $p = 2$ or $p = 4$? These questions will be explored later in the thesis, most particularly in Section 11.2. For now, let us just mention one reason why working with $p = 1$ is convenient. Recall that many of the most effective algorithms for psd low-rank approximation output a Nyström low-rank approximation $\widehat{A}$; see Definition 2.8. For such an approximation, Proposition 2.9(a) ensures that $\mathbf{0} \preceq \widehat{A} \preceq A$, so the Schatten 1-norm and trace of the residual matrix $A - \widehat{A}$ coincide:

$$\|A - \widehat{A}\|_{\mathsf{S}_1} = \|A - \widehat{A}\|_* = \operatorname{tr}(A - \widehat{A}).$$

Thus, *for a Nyström approximation $\widehat{A}$*, the (r, ε)-approximation condition (3.2) can be written as

$$\mathbb{E} \operatorname{tr}(A - \widehat{A}) \leq (1 + \varepsilon) \operatorname{tr}(A - [\![A]\!]_r).$$

The trace is a linear functional, which will be very useful during algorithm analysis.

Remark 3.7 (High probability bounds). Another standard way of analyzing psd low-rank approximations is to define an (r, ε, p) *approximation with failure probability*

δ as a random matrix $\widehat{\boldsymbol{B}} \approx \boldsymbol{B}$ for which

$$\|\boldsymbol{B} - \widehat{\boldsymbol{B}}\|_{\mathsf{S}_p} \le (1 + \varepsilon)\|\boldsymbol{B} - [\![\boldsymbol{B}]\!]_r\|_{\mathsf{S}_p} \quad \text{with probability at least } 1 - \delta. \quad (3.3)$$

In a sense, the high probability guarantee (3.3) is stronger than the expectation guarantee (3.2), though the guarantees are formally incomparable. For the purposes of this thesis, we will consider an algorithm to be theoretically supported if it admits either type of guarantee. ⋄

3.3 Pivoted partial Cholesky decompositions

A classical approach in computational linear algebra for computing low-rank approximations of a matrix is *partial matrix decomposition*. A partial matrix decomposition refers to any standard matrix decomposition (QR, Cholesky, SVD) where rows or columns of the factor matrices have been deleted or simply not computed at all. The most famous example of low-rank approximation by partial matrix decomposition is furnished by the Schmidt–Mirsky–Eckart–Young theorem (Fact 2.1), which states that partial singular value decompositions yield optimal low-rank approximations.

For efficient psd low-rank matrix approximation, we use a partial version of a different matrix decomposition, the *Cholesky decomposition*. The Cholesky decomposition of a psd matrix $\boldsymbol{A}$ represents the matrix as a product $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^*$ of a lower triangular matrix $\boldsymbol{L}$ and its adjoint. The Cholesky decomposition is sometimes written $\boldsymbol{A} = \boldsymbol{R}^*\boldsymbol{R}$ using an *upper triangular* matrix $\boldsymbol{R}$. The two conventions are easily inter-converted, $\boldsymbol{R} = \boldsymbol{L}^*$. By truncating the Cholesky decomposition to just the first k columns, we obtain a rank-k approximation $\widehat{\boldsymbol{A}}^{(k)} := \boldsymbol{L}(:, 1 : k)\boldsymbol{L}(:, 1 : k)^* \approx \boldsymbol{A}$. The factored approximation $\widehat{\boldsymbol{A}}^{(k)} = \boldsymbol{L}(:, 1 : k)\boldsymbol{L}(:, 1 : k)^*$ is known as a *partial Cholesky decomposition of $\boldsymbol{A}$*.

We compute the Cholesky decomposition by Gaussian elimination. To describe the algorithm, let $\widehat{\boldsymbol{A}}^{(i)}$ denote the approximation produced at step i, and let $\boldsymbol{A}^{(i)} := \boldsymbol{A} - \widehat{\boldsymbol{A}}^{(i)}$ denote the residual. The entries of $\boldsymbol{A}^{(i)}$ are $\boldsymbol{A}^{(i)}(j, k) = a_{jk}^{(i)}$. Under this convention, $\boldsymbol{A}^{(0)} := \boldsymbol{A}$ denotes the initial matrix. The procedure goes as follows: For $i = 1, \ldots, n$,

1. **Rescale.** Extract the ith column $\boldsymbol{a}_i^{(i-1)}$ of $\boldsymbol{A}^{(i-1)}$ and rescale

$$\boldsymbol{\ell}_i := \boldsymbol{a}_i^{(i-1)}/(a_{ii}^{(i-1)})^{1/2}.$$

The vectors $\boldsymbol{\ell}_i$ comprise the columns of the lower triangular matrix $\boldsymbol{L}$.

2. **Eliminate.** Update the residual $A^{(i)} := A^{(i-1)} - \ell_i \ell_i^*$. This step has the effect of zeroing out the matrix in the ith row and column.

The procedure succeeds provided all of the diagonal entries $a_{ii}^{(i-1)}$ encountered during factorization are nonzero. Since the procedure zeros out the ith row and column of the matrix at every iteration, and the update rule does not introduce nonzero entries into previously nonzero rows. The procedure terminates with a decomposition $A = LL^*$.

Pivoting, in general

As a method of low-rank approximation, the standard partial Cholesky decomposition can be ineffective, as it always forms an approximation based on the first k columns. The procedure can be modified to eliminate the columns in a general order. The resulting procedure is called a *pivoted Cholesky decomposition*, and the positions $s_i \in \{1, \ldots, n\}$ that are eliminated at each step i are called *pivot indices*. Concretely, beginning from $A^{(0)} = 0$, do the following for $i = 1, \ldots, n$:

1. **Select a pivot.** Choose a *pivot index* $s_i \in \{1, \ldots, n\}$ associated with a nonzero *pivot entry* $a_{s_i s_i}^{(i-1)} \neq 0$.

2. **Rescale.** Extract and rescale the pivot column

$$\ell_i := a_{s_i}^{(i-1)} / (a_{s_i s_i}^{(i-1)})^{1/2}.$$

3. **Eliminate.** Update the residual $A^{(i)} := A^{(i-1)} - \ell_i \ell_i^*$. This step has the effect of zeroing out the matrix in the s_ith row and column.

The ordered list of pivots $\mathsf{S} = \{s_1, \ldots, s_n\}$ gives rise to a reordering of the rows and columns of the matrix A. The matrix L produced by this procedure is typically not triangular, but it becomes lower triangular after rearranging its rows $L(\mathsf{S}, :)$ according to S. The reordered factor matrix $L(\mathsf{S}, :)$ is the traditional Cholesky factor of the *symmetrically* reordered psd matrix $A(\mathsf{S}, \mathsf{S})$. As with the standard Cholesky decomposition, pivoted partial Cholesky decompositions yield low-rank approximations of the matrix, i.e., $\widehat{A} = FF^*$ for $F = L(:, 1:k)$.

To make pivoted partial Cholesky decomposition an effective method for low-rank approximation in the entry access model, we make a final optimization. The standard Cholesky procedure overwrites the entire residual matrix at every step, at a cost of

Program 3.1 `pivpartchol.m`. Pivoted partial Cholesky for computing a low-rank approximation to a psd matrix.

```
function F = pivpartchol(Acol,n,s)
% Input:  Function Acol for producing columns Acol(i) = A(:,i) of A,
%         size n of A, list s = [s(1) ... s(k)] of k pivots to
%         eliminate
% Output: Factor F defining a rank-k approximation Ahat = F*F'

F = zeros(n,length(s));                    % To store output
for i = 1:length(s)
    ai = Acol(s(i)) - F(:,1:i-1)*F(i,1:i-1)'; % ith col of A - F*F'
    F(:,i) = ai / sqrt(ai(s(i)));          % Rescale
end

end
```

$O(n^2)$ operations. But this is wasteful, as we only ever need to evaluate the residual in the selected pivot columns. As a more efficient procedure, we avoid updating the residual explicitly, instead generating columns of $A^{(i)}$ as-needed using the formula $A^{(i)} = A - \widehat{A}^{(i)}$, where $\widehat{A}^{(i)}$ denotes the low-rank approximation produced at step i.

Code for this optimized version of the pivoted partial Cholesky decomposition appears as Program 3.1. As with other programs in this part of the thesis, this code interacts with the matriz A through a function `Acol`, defined so that `Acol(i)` outputs the ith column a_i.

Greedy pivoting

The pivoted partial Cholesky decomposition gives us a general procedure for solving psd low-rank approximation problems in the entry access model. But how should we pick the pivots? The main algorithm of this part of the thesis, randomly pivoted Cholesky, uses a *randomized rule* for pivot selection. Before getting to this method in Chapter 4, we review a more classical approach: *greedy pivoting* (also known as *diagonal* or *complete pivoting*) [Hig90; FS01].

The idea of greedy pivoting follows from our discussion in Section 3.1; large entries of the matrix can only lie in columns of the matrix with large diagonal entries. This observation suggests a greedy approach: Always choose a maximal diagonal entry of the residual matrix $A^{(i)}$ as pivot:

$$s_{i+1} \in \underset{j}{\operatorname{argmin}} \, a_{jj}^{(i)}.$$

Program 3.2 `greedy_chol.m`.   Pivoted partial Cholesky decomposition with greedy pivoting to compute a low-rank approximation to a psd matrix.

```matlab
function [F,S] = greedy_chol(Acol,d,k)
% Input:  Function Acol for producing columns Acol(i) = A(:,i) of A,
%         diagonal d of A, rank k
% Output: Factor F defining a rank-k approximation Ahat = F*F', pivot
%         set S

F = zeros(length(d),k);                      % To store output
S = zeros(k,1);                              % To store pivots
for i = 1:k
    [~,S(i)] = max(d);                       % Largest diag entry
    as = Acol(S(i)) - F(:,1:i-1)*F(S(i),1:i-1)'; % sth col of A-F*F'
    F(:,i) = as / sqrt(as(S(i)));            % Rescale
    d = d - abs(F(:,i)).^2;                  % Update diagonal
    d = max(d,0); % Ensure nonnegative diagonal in floating point
end

end
```

We emphasize that the greedy method always picks the largest diagonal entry of the *current* residual matrix $A^{(i)}$, which evolves as the iteration counter i increases. The residual matrix is zero in the columns of previously selected pivot indices, so greedy selection ensures the same pivot is never selected twice.

An implementation of greedy pivoted (partial) Cholesky is given in Program 3.2. The main difference with the generic pivoted partial Cholesky decomposition implementation in Program 3.1 is that we track the diagonal $\mathbf{diag}(A^{(i)})$ of the residual matrix. The diagonal of A is provided to the program as an input d, and it is updated every iteration using the identity

$$\mathbf{diag}\left(A^{(i)}\right) = \mathbf{diag}\left(A^{(i-1)} - f_i f_i^*\right) = \mathbf{diag}\left(A^{(i-1)}\right) - |f_i|^2.$$

The greedy pivoted Cholesky algorithm reads $(k + 1)n$ entries of the matrix and expends $O(k^2 n)$ operations. The procedure outputs the factor F and the set of pivots $S = \{s_1, \ldots, s_k\}$.

Remark 3.8 (History). Low-rank approximation by pivoted partial Cholesky decompositions is classical. The use of greedy pivoting for Cholesky decomposition can be traced at least as far back as the work of Lawson and Hanson in 1974 [LH74]. The greedy pivoting strategy is also classical, and it goes under the traditional names of *diagonal pivoting* and *complete pivoting*. The procedure was incorporated into

both LINPACK software package in 1979 [DMBS79] and its replacement LAPACK [ABBB+99]. Stability analysis and analysis of the approximation quality was done by Higham [Hig90]. The greedy method received new attention in the kernel machine learning community following the work of Fine and Scheinberg [FS01].   ◇

Failure of greedy pivoting

The greedy method seems very natural, but it can have significant deficiencies on some examples. Consider, for instance, the matrix

$$A = \begin{bmatrix} (1+\varepsilon)\mathbf{I}_{n/10} & \mathbf{0} \\ \mathbf{0} & \mathbf{1}_{9n/10}\mathbf{1}^*_{9n/10} \end{bmatrix} \quad \text{for } \varepsilon > 0 \text{ small.}$$

Selecting any pivot entry from the (2,2) block produces the rank-one approximation

$$\widehat{A}_{\text{good}} := \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1}_{9n/10}\mathbf{1}^*_{9n/10} \end{bmatrix}$$

which achieves a relative trace error of roughly 10%:

$$\frac{\text{tr}(A - \widehat{A}_{\text{good}})}{\text{tr}(A)} = (1 + O(\varepsilon)) \cdot \frac{1}{10}.$$

However, when the greedy pivoted Cholesky algorithm is run on A, the diagonal entries in the (1,1) block are slightly larger than the entries in the (2,2) block, so it proceeds by eliminating entries in the (1,1) block one at a time. In particular, the relative trace error of the output $\widehat{A}_{\text{greedy}}$ of the greedy method remains above roughly 90% until over one tenth of the matrix entries have been read:

$$\frac{\text{tr}(A - \widehat{A}_{\text{greedy}})}{\text{tr}(A)} \geq (1 - O(\varepsilon)) \cdot \frac{9}{10} \quad \text{as long as } k \leq \frac{n}{10}.$$

This is a dismal performance for an algorithm; a good choice of pivot will approximate the matrix to $\approx 10\%$ relative error in one step, but the greedy pivoted Cholesky method fails to obtain error better than $\approx 90\%$ even after reading a tenth of the matrix! The greedy method can fail in ways that are perhaps even more striking. The examples are somewhat sophisticated and rely on variations of Kahan's famous matrix; we refer the interested reader to [Hig90] for details.

The greedy pivoting strategy is natural and often works well, but it has a fatal flaw. The greedy approach focuses entirely on *exploiting* large diagonal entries, but fails to *explore* potentially valuable pivot choices outside of the numerically largest diagonal entry. This issue is rectified by the randomly pivoted Cholesky algorithm, which uses randomization to preferentially select large diagonal entries while investigating a broader range of pivot choices.

3.4 Column Nyström approximation

We now have a procedure, the pivoted partial Cholesky decomposition, for computing a low-rank approximation to a psd matrix. But what *is* the output of this procedure? Is there a formula for it? What are its properties?

To answer these questions, we begin by asking a more abstract question: *How should we approximate a matrix A from a subset of columns $a_{s_1}, \ldots, a_{s_k}$?* For notational convenience, these columns can be packaged into a submatrix $A(:, S)$, indexed by the set $S = \{s_1, \ldots, s_k\}$. Once we know the columns of a psd matrix, we also know its rows $A(S, :) = A(:, S)^*$. We can build an approximation $\widehat{A} \approx A$ by interpolating the known rows and columns, resulting in an approximation of the form

$$\widehat{A} = A(:, S) W A(S, :) \quad \text{for some } W \in \mathbb{K}^{k \times k}.$$

It is natural to expect that $\widehat{A}$ agrees with A in the selected columns,

$$\widehat{A}(:, S) = A(:, S). \tag{3.4}$$

The condition (3.4) may be ensured by setting $W \coloneqq A(S, S)^\dagger$. If $A(S, S)$ is invertible, W is the *unique* matrix producing an approximation $\widehat{A}$ satisfying (3.4). This reasoning motivates the following definition:

Definition 3.9 (Column Nyström approximation). Let $A \in \mathbb{K}^{n \times n}$ be a psd matrix and let $S \subseteq \{1, \ldots, n\}$ be a set of indices. The *column Nyström approximation* with *pivot set* S is:

$$A\langle S \rangle \coloneqq A(:, S) A(S, S)^\dagger A(S, :). \tag{3.5}$$

As the name suggests, a column Nyström approximation $\widehat{A} = A\langle S \rangle$ is a Nyström approximation $A\langle \Omega \rangle$ in the sense of Definition 2.8. The associated test matrix is $\Omega = I(:, S)$. Consequently, column Nyström approximations enjoy all of the properties of Nyström approximations presented in Proposition 2.9.

As one might hope, the output of a pivoted partial Cholesky decomposition is a column Nyström approximation.

Fact 3.10 (Nyström and Cholesky). *Let $\widehat{A}$ be the approximation to A produced by the pivoted partial Cholesky algorithm (Program 3.1) with pivots $S = \{s_1, \ldots, s_k\}$. Then $\widehat{A} = A\langle S \rangle$ is the column Nyström approximation with pivot set S.*

Having identified the class of column Nyström approximations, it is natural to ask: How accurate can these approximations be? This question admits a precise answer using the concept of an (r, ε)-approximation.

Fact 3.11 (Column Nyström approximation: Approximation quality). *Fix parameters $r \geq 1$ and $0 < \varepsilon \leq r$. For any psd matrix $A \in \mathbb{K}^{n \times n}$, there exists a set $\mathsf{S} \subseteq \{1, \ldots, n\}$ of size*

$$k = \min \left\{ \left\lceil \frac{r}{\varepsilon} + r - 1 \right\rceil, n \right\} \tag{3.6}$$

for which $A\langle \mathsf{S} \rangle$ is an (r, ε)-approximation to A. Conversely, there exists a psd matrix A for which $k \geq r/\varepsilon$ columns are necessary *to produce a column Nyström approximation that is an (r, ε)-approximation.*

A version of this result for projection approximation of general matrices was proven by Guruswami and Sinop [GS12]. Their result extends to (r, ε)-approximation approximations by the Gram correspondence; see [CETW25] for a self-contained proof in the psd setting. The upper bound (3.6) is proven by using DPP sampling; see Fact 3.22 below.

Fact 3.11 shows that $k \approx r/\varepsilon$ columns are necessary for a column Nyström approximation to be an (r, ε)-approximation, at least for a worst-case matrix A. Therefore, for most of this thesis, a "near-optimal" algorithm for the psd low-rank approximation problem will be one which produces a rank-k column Nyström approximation satisfying the (r, ε)-approximation guarantee where k is nearly equal to r/ε.

Incidentally, we note that there is an even sharper approximation guarantee for the case $r = k$:

Theorem 3.12 (Column Nyström approximation: Approximation quality, $r = k$). *For any psd matrix $A \in \mathbb{K}^{n \times n}$, there exists a column subset S of size k such that $A\langle \mathsf{S} \rangle$ is a (k, k)-approximation, i.e.,*

$$\operatorname{tr}(A - A\langle \mathsf{S} \rangle) \leq (k + 1) \operatorname{tr}(A - [\![A]\!]_k).$$

Moreover, for every $\gamma > 0$, there exists a psd matrix $A \in \mathbb{R}^{(k+1) \times (k+1)}$ such that

$$\operatorname{tr}(A - A\langle \mathsf{S} \rangle) \geq (k + 1 - \gamma) \operatorname{tr}(A - [\![A]\!]_k) \quad \textit{for every } k\textit{-element subset } \mathsf{S}.$$

Observe that the existence result is the $\varepsilon = r$ case of Fact 3.11. The lower bound requires a separate argument, which follows from [DRVW06, Prop. 3.3] and the Gram correspondence. I provide a proof in Appendix D.1.

3.5 Subset selection problems

Before moving on, we draw a connection between the low-rank approximation techniques we have been studying and a different type of computational problem:

subset selection. Subset selection is more of a qualitative problem than a quantitative one: Given a (multi)set X of n items, we wish to identify a subset $S \subseteq X$ of $k \ll n$ *representative items*. Typically, we want a set this set of representatives to be *diverse*; if one element x is repeated many times in X, only one copy of x is needed in the subset S as a representative.

There are several applications for subset selection:

1. ***Information retrieval.*** For designing user interfaces to large databases, it can be important to surface a small number of "recommended" items. Examples include product recommendation [WMG19] and document retrieval [CK06].

2. ***Optimal sensor placement, optimal experimental design, and active learning.*** Given multiple information sources—possible locations to place sensors, possible scientific experiments to run, or possible unlabeled data points to collect labels for—which (small number) should I use to learn the maximum possible amount of information? These are the subjects of the related, but distinct, problems of optimal sensor placement [RCV14], optimal experimental design [Puk06], and active learning [AKGH+14]. All of these problems are examples of subset selection problems.

3. ***Genetics.*** A basic question in biology is to determine a small set of genetic markers that predict an observed trait or that characterize variation in a population. This application has served as a main motivation for the development of randomized algorithms for subset selection [PZBC+07; MD09].

4. ***Computational mathematics.*** There are many applications of subset selection to computational mathematics itself. Often, these applications require the selection of a small number of columns from a matrix. Examples include rank-structured matrix computations [Mar11; Wil21] tensor network algorithms [OT10; TSL24], and recovery of rational functions from measurements [WDT22].

In order to design and analyze algorithms for the subset selection problem, several different ways of mathematizing the problem have been proposed. One approach is based on linear algebra. We represent the items in X by columns of a matrix B and seek a subset S of columns that span a good low-rank approximation to B. We call this problem the *column subset selection* problem.

There are variants of the column subset selection problem both for general, rectangular matrices and for psd matrices. We focus on the latter problem; the former is discussed in Chapter 9. The *psd column subset selection problem* is as follows:

> **Psd column subset selection problem.** Given a psd matrix A and a subset size k, find a subset S of k pivots such that the trace-error of the Nyström approximation $\operatorname{tr}(A - A\langle S \rangle)$ is as small as possible.

Phrased in this way, the column subset selection problem seems like a reformulation of the psd low-rank approximation problem. However, there are reasons to consider these problems as distinct. For psd low-rank approximation, the output of interest is the low-rank approximation $\widehat{A}$, possibly generated as a column Nyström approximation $\widehat{A} = A\langle S \rangle$. For psd column subset selection, the relevant output is the subset S itself; the trace-error $\operatorname{tr}(A - A\langle S \rangle)$ is useful only instrumentally as a way of measuring subset quality. Another distinction is that, in many subset selection applications like genetics or product recommendation, one wants *every* element of the subset S to be "good". (It would be considered a large failure to incorrectly suggest a genetic marker is linked to cancer, for instance.) By contrast, bad pivots in low-rank approximation are more of a missed opportunity than a negative: Bad pivots are not helpful in improving the approximation quality, but they do not hurt it either. Throughout this part of the thesis, we will use pivoted partial Cholesky decompositions both for low-rank approximation and subset selection.

3.6   Column Nyström approximation and Gaussian random variables

Partial Cholesky decomposition and column Nyström approximation are closely related to conditional distributions of (jointly) Gaussian random variables. Consider a vector $z \sim \textsc{Normal}_{\mathbb{K}}(\mathbf{0}, A)$, and let $S \subseteq \{1, \ldots, n\}$ be a subset of indices. What is the distribution of z *conditional* on observing the coordinates $z(S)$? This question is answered by the following classical result:

Theorem 3.13 (Conditional expectations of Gaussian random vectors). *Let $z \sim \textsc{Normal}_{\mathbb{K}}(\mathbf{0}, A)$ be a Gaussian random vector with psd covariance matrix A, and let $S \subseteq \{1, \ldots, n\}$ be a subset of indices. Assume $A(S, S)$ is nonsingular. Then*

$$\mathbb{E}[z \mid z(S)] = A(:, S)A(S, S)^{-1}z(S) \sim \textsc{Normal}_{\mathbb{K}}(\mathbf{0}, A\langle S \rangle),$$

$$z \mid z(S) \sim \textsc{Normal}_{\mathbb{K}}(A(:, S)A(S, S)^{-1}z(S), A - A\langle S \rangle).$$

This result is carefully developed for real Gaussians in [Tro23, Ch. 21], and the extension to the complex case is straightforward. We recognize the Nyström approximation $A\langle S\rangle$ as the covariance matrix of the conditional expectation $\mathbb{E}[z \mid z(S)]$, and its residual $A - A\langle S\rangle$ as the covariance of z *conditional* on $z(S)$.

A simple model of experimental design

To put this connection between Cholesky factorization, Nyström approximation, and Gaussian random variables into action, consider the following very basic experimental design problem. A scientist has n experiments she could run, but only the budget to run k of them. Which experiments should she run to maximize the knowledge she learns? Equivalently, which experiments should she run to minimize her uncertainty about the outcome of the remaining experiments she did not run?

Suppose that we can model the outcomes of the experiments as Gaussian random variables $z \sim \text{NORMAL}(m, A)$ with *known* mean m and covariance matrix A. Choosing the optimal set of k experiments amounts to choosing k entries of z to observe with the goal of minimizing the sum of the variances of the remaining experimental outcomes, conditional on these measurements:

$$\underset{S\subseteq\{1,\dots,n\}}{\text{minimize}} \sum_{j=1}^{n} \text{Var}(z_j \mid z(S)) \quad \text{such that } |S| = k. \tag{3.7}$$

By Theorem 3.13, the sum of posterior variances is precisely the trace error of the Nyström approximation

$$\sum_{j=1}^{n} \text{Var}(z_j \mid z(S)) = \text{tr}(A - A\langle S\rangle).$$

Therefore, this experimental design problem is fully equivalent to selecting a column subset generating a good Nyström approximation, measured using the trace error.

Connection to Cholesky decomposition

Let us now apply a pivoted partial Cholesky decomposition to solve the experimental design problem (3.7). At each step, we have a subset $S_i \subseteq \{1, \dots, n\}$ of experiments we have already decided to run, and we must choose the next experiment. The diagonal entries of the residual $A^{(i)} = A - A\langle S_i\rangle$ store the conditional variances

$$a_{jj}^{(i)} = \text{Var}(z_j \mid S_i).$$

The greedy method, introduced in Section 3.3, chooses the largest diagonal entry as pivot at each step. Equivalently, it chooses to run the experiment with the *highest variance*, conditional on the already-run experiments.

This strategy—always run the experiment over which there is the most uncertainty—is very natural. However, it has a flaw in that it doesn't take into account the *correlations* between experiments. Consider the following matrix

$$A = \begin{bmatrix} 1 + 2\varepsilon & 0 & 0 & 0 \\ 0 & 1 + \varepsilon & 1 & 1 \\ 0 & 1 & 1 + \varepsilon & 1 \\ 0 & 1 & 1 & 1 + \varepsilon \end{bmatrix} \quad \text{for } \varepsilon > 0 \text{ small.} \tag{3.8}$$

The first experiment has slightly higher variance than the other experiments, so the greedy method will choose to run experiment 1. However, the outcomes of experiments 2, 3, and 4 are highly correlated; running any one of these experiments will leave the scientist with tiny uncertainty about the outcome of the other experiments. This example provides another demonstration of why greed isn't always good for column subset selection, and demonstrates how injecting randomness can help improve column subset selection algorithms. For this example, just picking an experiment to run at random would give better results than the greedy method 75% of the time. As we'll see in the next section and in Chapter 4, there are much better algorithms for column subset selection than uniform random selection.

A peek forward. The connection between Cholesky factorization, Nyström approximation, and Gaussian random variables is a powerful tool. It forms the basis of Chapters 5 and 6, where we will use Nyström approximation to accelerate algorithms for learning from data based on *Gaussian processes*.

3.7 Non-adaptive random sampling methods

Interest in psd low-rank approximation and column subset selection was renewed in the early twenty-first century, driven by efforts to accelerate kernel methods in machine learning [WS00; FS01; DM05]. In addition to continued focus on deterministic methods like greedy selection (and variations thereof), this wave of interest also spurred the development of randomized methods.

This section will summarize non-adaptive random sampling methods for psd low-rank approximation methods that were developed prior to our work on randomly pivoted Cholesky [CETW25]. As we will detail, randomly pivoted Cholesky is related to but distinct from the methods presented in this section.

Uniform sampling and diagonal-power sampling

The most basic randomized method for computing a column Nyström approximation is to select the pivot set S uniformly at random, without replacement. (Sampling with replacement is fine as well, as duplicated columns have no effect on a column Nyström approximation.) For many practical problems, uniform sampling produces approximations of high-enough quality, though it can product significantly worse approximations on other problems.

One class of problems for which uniform sampling is poorly suited are problems with diagonal entries that span a wide range of magnitudes. To obtain better results in this setting, we should adapt the sampling distribution to the size of the diagonal entries. Choosing a power $p > 0$, we may draw pivots $s_1, \ldots, s_k$ iid from the *diagonal-power sampling* distribution

$$s_1, \ldots, s_k \overset{\text{iid}}{\sim} \mathbf{diag}(A)^p. \tag{3.9}$$

The diagonal $\mathbf{diag}(A) \geq 0$ is entrywise nonnegative since A is psd, and the power p is applied to the vector $\mathbf{diag}(A)$ entrywise. Recall that we write $s \sim w$ to denote a sample $\mathbb{P}\{s = j\} = w_j / \sum_{k=1}^{n} w_k$ from any *unnormalized* weight vector $w \in \mathbb{R}_+^n$. The power p in diagonal-power sampling controls the amount of "greediness" of the procedure; large values of p lead to sampling the large elements with high probability, and smaller values of p lead to a more uniform distribution.

In 2005, Drineas and Mahoney [DM05] proposed the diagonal-power sampling distribution with $p = 2$. The choice $p = 2$ was motivated by a line of work initiated by Frieze, Kannan, and Vempala [FKV98], who proved results for approximating a general matrix $B \in \mathbb{K}^{m \times n}$ by projecting onto selected columns $B(:, s_i)$ sampled iid from the squared column-norm distribution

$$s_1, \ldots, s_k \sim \mathbf{scn}(B).$$

Recall that $\mathbf{scn}(B) \in \mathbb{R}_+^n$ denotes the squared column norms of B. Computing the full column norms in the entry access model is expensive, so Drineas and Mahoney suggested sampling from the squared diagonal entries as an alternative.

The Gram correspondence (Section 2.6) suggests a different value for p. If we treat $A = B^*B$ as the Gram matrix for a general matrix B, computing a low-rank approximation by projecting onto a subset of B's columns is equivalent to computing a Nyström approximation of A, and sampling the squared column norms of B is equivalent to the diagonal-power sampling rule (3.9) with $p = 1$, in view of the

Program 3.3 `diag_sample_nys.m`. Diagonal-power sampling with power $p = 1$ for computing a Nyström approximation to a psd matrix.

```
function [F,S] = diag_sample_nys(Acol,d,k)
% Input:  Function Acol for producing columns Acol(i) = A(:,i) of A,
%         diagonal d of A, rank k
% Output: Factor F defining a rank-k approximation Ahat = F*F', pivot
%         set S

S = datasample(1:length(d),k,"Replace",false,"Weights",d);
AS = Acol(S);              % Columns of A
F = AS / chol(AS(S,:));    % Factor matrix

end
```

identity $\mathbf{diag}(A) = \mathbf{scn}(B)$. As such, the power $p = 1$ could be regarded as the more "natural" power for psd low-rank approximation (insofar as squared column norm sampling is the "natural" approach for general matrix approximation). Code for $p = 1$ diagonal-power sampling is provided in Program 3.3.

The choice of power p is often moot for the basic diagonal-power sampling scheme because kernel matrices and covariance matrices in machine learning are often normalized to have a constant diagonal. However, the choice of p will have more influence when we consider adaptive procedures like RPCHOLESKY.

Fact 3.14 (Diagonal power sampling, $p = 1$). *Let $A \in \mathbb{K}^{n \times n}$ be a psd matrix, fix $r \geq 1, \varepsilon > 0$, and introduce the relative error of the best rank-r approximation:*

$$\eta := \frac{\mathrm{tr}(A - [\![A]\!]_r)}{\mathrm{tr}(A)}.$$

Diagonal-power sampling Nyström approximation with $p = 1$ produces a (r, ε)-approximation provided that the pivot set has size

$$k \geq \frac{r-1}{\varepsilon\eta} + \frac{1}{\varepsilon}.$$

Moreover, the $k = O(r/\varepsilon\eta)$ complexity is necessary for a worst-case A matrix.

This result is [CETW25, Thm. C.3], which adapts [FKV98, eq. (4)] using the Gram correspondence. The statement in [CETW25] focuses on the real case, but the proof transfers without issue to complex numbers.

This result demonstrates fundamental limitations of the uniform and diagonal-power sampling approaches, which require $O(1/\eta)$ pivots to produce a column Nyström

approximations of relative error η. We will substantially improve on this result with randomly pivoted Cholesky, which has a much lower cost of $\mathcal{O}(\log(1/\eta))$. Table 4.1 below presents experiments demonstrating this failure mode for the uniform and diagonal-power sampling methods.

Ridge leverage score sampling

To motivate the ridge leverage score sampling approach, let us take a brief digression to the subject of *ridge-regularized linear regression*. Consider the task of fitting a *conjugate linear* mapping $\chi \mapsto \chi^* \boldsymbol{\beta}$ from $\mathbb{K}^n$ to $\mathbb{K}$ from input–output pairs $(\chi^{(1)}, y_1), \ldots, (\chi^{(m)}, y_m) \in \mathbb{K}^n \times \mathbb{K}$. Assemble the inputs $\chi^{(i)}$ as *rows* of a matrix X, $X(i,:) = (\chi^{(i)})^*$, and collect the outputs into a vector $\boldsymbol{y} \in \mathbb{K}^m$. One natural approach to fitting a linear model is *ridge-regularized linear regression*, which chooses the coefficients $\boldsymbol{\beta} \in \mathbb{K}^n$ as the solution to an optimization problem

$$\boldsymbol{\beta} = \operatorname*{argmin}_{\boldsymbol{\beta} \in \mathbb{K}^n} \|X\boldsymbol{\beta} - \boldsymbol{y}\|^2 + \lambda \|\boldsymbol{\beta}\|^2. \tag{3.10}$$

The *ridge parameter* $\lambda \geq 0$ sets the amount of regularization. For $\lambda = 0$, the coefficients $\boldsymbol{\beta}$ are taken to be the limiting value of (3.10) as $\lambda \downarrow 0$. The solution $\boldsymbol{\beta}$ to (3.10) is given by the formulas

$$\boldsymbol{\beta} = (X^* X + \lambda \mathbf{I})^\dagger X^* \boldsymbol{y} \tag{3.11}$$

$$= X^* (X X^* + \lambda \mathbf{I})^\dagger \boldsymbol{y}. \tag{3.12}$$

The solution formulas (3.11) and (3.12) are known as the *normal equations* and *adjoint normal equations* for the ridge regression problem (3.10).

The coefficients $\boldsymbol{\beta}$ give rise to the predicted values

$$\widehat{\boldsymbol{y}} = X\boldsymbol{\beta}. \tag{3.13}$$

for the input data elements $\chi^{(1)}, \ldots, \chi^{(m)}$. The ridge leverage scores measure the sensitivity of the predictions $\widehat{\boldsymbol{y}}$ to the data $\boldsymbol{y}$:

Definition 3.15 ((Ridge) leverage scores of a general matrix)**.** Let $\lambda \geq 0$. The λ-*ridge leverage scores* $\boldsymbol{\ell}^\lambda$ of a matrix $X \in \mathbb{K}^{m \times n}$ are

$$\boldsymbol{\ell}^\lambda := \left(\frac{\partial \widehat{y}_i}{\partial y_i} : 1 \leq i \leq m \right) = \mathbf{diag}(X X^* (X X^* + \lambda \mathbf{I})^\dagger). \tag{3.14}$$

The *leverage scores* $\boldsymbol{\ell}$ are defined as the 0-ridge leverage scores.

The characterization of the ridge leverage scores as a matrix diagonal follows from the definition (3.13) of the predicted values and the adjoint normal equations (3.12).

While they are not our main focus for now, the leverage scores are an important object in randomized matrix computations. The leverage scores can be computed as the squared row norms of an orthonormal basis matrix for X, $\boldsymbol{\ell} = \mathbf{srn}(\mathbf{Orth}(X))..$ The ith leverage score is a measure of how "important" row i is to the matrix X. These scores have a decades-long history in statistics, where they are used to quantify sensitivity of a regression model to changes in output values $\boldsymbol{y}$ [JWHT21, §3.3.3].

The equation (3.14) shows that the ridge leverage scores of a matrix X depend only on the matrix $A := XX^*$, which is the Gram matrix *of X^**. This motivates the following definition.

Definition 3.16 (Ridge leverage scores of a psd matrix). Let $\lambda \geq 0$ be a number and $A \in \mathbb{K}^{n \times n}$ be a *psd* matrix. The λ-*ridge leverage scores* of A are $\boldsymbol{\ell}^{\lambda} := \mathbf{diag}(A(A + \lambda\mathbf{I})^{\dagger})$, and the λ-*effective dimension* of A is $\mathrm{d}_{\mathrm{eff}}(\lambda) := \sum_{i=1}^{n} \ell_i^{\lambda}$.

In principle, this definition could be ambiguous since a psd matrix A has two sets of ridge leverage scores: its leverage scores as a general matrix under Definition 3.15 and its leverage scores as a psd matrix using Definition 3.16. For our purposes, however, the intended meaning should always be clear, and the ridge leverage scores of a matrix that is stated to be psd will always be given by Definition 3.16. The ridge leverage scores were originally proposed by Alaoui and Mahoney [AM15].

The effective dimension $d_{\mathrm{eff}}(\lambda)$ a continuous proxy for the rank of a psd matrix A, where eigenvalues of A that are much smaller than level λ are treated as negligible. The 0-effective dimension is the algebraic rank, $d_{\mathrm{eff}}(0) = \mathrm{rank}\, A$, and the effective dimension decreases to zero as $\lambda \uparrow +\infty$.

The ridge leverage scores give natural sampling probabilities for selecting columns for Nyström approximation. We have the following result, slightly simplified from [MM17, Thm. 3]:

Fact 3.17 (Ridge leverage score sampling: Spectral norm). *Let $A \in \mathbb{K}^{n \times n}$ be a psd matrix, $\lambda > 0$ be a ridge parameter, $\boldsymbol{\ell}^{\lambda}$ be the ridge leverage scores, and $\delta \in (0, 1)$ be a specified failure probability. Let $\widehat{\boldsymbol{\ell}}^{\lambda}$ be over-approximations to the ridge leverage scores*

$$\boldsymbol{\ell}^{\lambda} \leq \widehat{\boldsymbol{\ell}}^{\lambda} \leq c\boldsymbol{\ell}^{\lambda} \quad \text{with } c \geq 1, \tag{3.15}$$

and define sampling probabilities

$$p_i := \max\left\{1, 16\widehat{\ell}_i^\lambda \log\left(\delta^{-1} \sum_{j=1}^n \widehat{\ell}_j^\lambda\right)\right\}.\tag{3.16}$$

Define pivots S *by including each* $1 \leq i \leq n$ *in* S *independently with probability* p_i. *With probability at least* $1 - \delta$, *the pivot set is not too large* $|\mathsf{S}| \leq 32c\, \mathrm{d}_{\mathrm{eff}}(\lambda) \log(c\, \mathrm{d}_{\mathrm{eff}}(\lambda)/\delta)$ *and*

$$A\langle\mathsf{S}\rangle \preceq A \preceq A\langle\mathsf{S}\rangle + \lambda\mathbf{I}.\tag{3.17}$$

Observe that the lower bound $A\langle\mathsf{S}\rangle \preceq A$ in (3.17) is true for any Nyström approximation, in view of Proposition 2.9(a). This result shows that, if we sample $O(\mathrm{d}_{\mathrm{eff}}(\lambda) \log \mathrm{d}_{\mathrm{eff}}(\lambda))$ pivots S using the ridge leverage score (RLS) distribution, then we get a matching upper bound up to additive error $\lambda\mathbf{I}$. The bound (3.17) is very strong; in particular, it applies the *spectral norm* error bound $\|A - A\langle\mathsf{S}\rangle\| \leq \lambda$.

Fact 3.18 (Ridge leverage score sampling: Trace norm). *Let* $A \in \mathbb{K}^{n \times n}$ *be a psd matrix,* $\varepsilon > 0$ *be a real number, and* $1 \leq r \leq n$ *be an integer. Set* $\lambda := (2\varepsilon/r) \mathrm{tr}(A - [\![A]\!]_r)$, *and suppose we have approximate ridge leverage scores* $\widehat{\ell}^\lambda$ *satisfying* (3.15) *and defining sampling probabilities* $\boldsymbol{p}$ *by* (3.16). *Define pivots* S *by including each* $1 \leq i \leq n$ *in* S *independently with probability* p_i. *With probability at least* $1 - \delta$, *the pivot set is not too large* $|\mathsf{S}| = O(\frac{r}{\varepsilon} \log(\frac{r}{\varepsilon\delta}))$ *and* $A\langle\mathsf{S}\rangle$ *and* $A\langle\mathsf{S}\rangle$ *is an* (r, ε)*-approximation with failure probability* δ *(as in Remark 3.7).*

Musco and Musco proposed the recursive RLS (RRLS) algorithm for performing approximate ridge leverage score sampling [MM17]. They also develop versions of their algorithm that produce a set of pivots S of a prescribed size k and provide an automatic mechanism for selecting the hyperparameter λ. The cost of the algorithm is $O(nk)$ entry evaluations and $O(nk^2)$ additional arithmetic operations. MATLAB and Python implementations of RRLS are available [Van19]. Alternative algorithms for approximate RLS sampling are SQUEAK [CLV17] and BLESS [RCCR18].

RLS sampling is a mathematically elegant strategy for psd column subset selection, and it is amazing that is even *possible* to perform approximate RLS sampling using a small number of entry evaluations. Still, there are reasons to continue searching for a more performant algorithm. First, in our empirical testing, the available RLS sampling implementations require roughly $2kn$ to $3kn$ entry evaluations to produce

a rank-k approximation [CETW25, §2.4]; pivoted Cholesky-based approaches typically take just $(k + 1)n$ evaluations. Second, empirical and theoretical analysis suggests the constants in the O-notation for RLS sampling are moderately large, even when RLS sampling is performed exactly; for some examples, RLS sampling can be many orders of magnitude less accurate than alternative approaches for producing approximations of a given rank k; see Table 4.1. Finally, RLS sampling requires $O(r \log r)$ columns to produce a low-rank approximation comparable with the best rank-r approximation. As the following example shows, this logarithmic overhead is a real property of the algorithm, not an artifact of the analysis.

Example 3.19 (Collecting coupons). Fix parameter r and consider the psd matrix

$$
A = \begin{bmatrix}
\mathbf{1}_{n/r}\mathbf{1}_{n/r}^* & \mathbf{0} & \mathbf{0} \cdots & \mathbf{0} \\
\mathbf{0} & \mathbf{1}_{n/r}\mathbf{1}_{n/r}^* & \mathbf{0} & \cdots & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & \mathbf{1}_{n/r}\mathbf{1}_{n/r}^* & \cdots & \mathbf{0} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{1}_{n/r}\mathbf{1}_{n/r}^*
\end{bmatrix} \in \mathbb{K}^{n \times n}. \tag{3.18}
$$

This matrix is block diagonal, with r equally sized diagonal blocks of all ones. This matrix has rank r, and achieving a column Nyström approximation comparable to the best rank-r approximation requires selecting a pivot in each block. Regardless of the ridge parameter $\lambda \geq 0$, the ridge leverage scores are constant $\ell^\lambda = c(\lambda)\mathbf{1}$, so ridge leverage score sampling coincides with iid uniform sampling. This is an example of the well-known coupon collector problem, so it takes $\Theta(r \log r)$ pivots to attain a pivot in each block with high probability; see [MR95, §3.6] for an introduction to the coupon collector problem. ◇

Determinantal point process sampling

The example (3.18) presents a challenging example for any column subset selection methods based on drawing pivots iid from *any* distribution. For this reason, we generically expect iid sampling methods to require $k = O(r \log r)$ columns to produce an approximation comparable with the best rank-r approximation. To obtain better results, we can move beyond iid sampling. Pivoted partial Choleksy methods are one type of non-iid selection strategy, as these methods select columns one of a time in an *adaptive* way. Determinantal point process (DPP) sampling is an alternate approach that draws a random sample from a *joint distribution* on all possible subsets of k pivots.

To motivate DPP sampling, let us first motivate why the determinant is a useful metric in measuring the quality of a set S of pivots. The following result is informative:

Proposition 3.20 (Determinants and pivot entries). *Let $A \in \mathbb{K}^{n \times n}$ be a psd matrix and let $\mathsf{S} = \{s_1, \ldots, s_k\}$ be a set of pivots. Consider the partial Cholesky decomposition with this pivot set, and introduce the residuals $A^{(i)} := A - A\langle\{s_1, \ldots, s_i\}\rangle$. We have the identity*

$$\det[A(\mathsf{S}, \mathsf{S})] = a_{s_1 s_1}^{(0)} a_{s_2 s_2}^{(1)} a_{s_3 s_3}^{(2)} \cdots a_{s_k s_k}^{(k-1)}.$$

Proof. The matrix $A(\mathsf{S}, \mathsf{S})$ has Cholesky decomposition

$$A(\mathsf{S}, \mathsf{S}) = R^* R,$$

where the diagonal entries of R are $|r_{ii}|^2 = a_{s_i s_i}^{(i-1)}$. Ergo,

$$\det[A(\mathsf{S}, \mathsf{S})] = |\det(R)|^2 = \prod_{i=1}^{k} |r_{ii}|^2 = a_{s_1 s_1}^{(0)} a_{s_2 s_2}^{(1)} a_{s_3 s_3}^{(2)} \cdots a_{s_k s_k}^{(k-1)}.$$

The desired result is proven. $\square$

We recognize the determinant of the submatrix $\det[A(\mathsf{S}, \mathsf{S})]$ as the product of all the pivot diagonal entries of the matrix A during pivoted Cholesky decomposition. As such, choosing a pivot set S yielding a submatrix $A(\mathsf{S}, \mathsf{S})$ of large determinant corresponds to selecting an order for pivoted partial Cholesky decomposition in which all of the pivot diagonal entries are *simultaneously* large.

This result suggests a computational strategy of selecting the pivot set S by maximizing the determinant $\det[A(\mathsf{S}, \mathsf{S})]$. Unfortunately, the problem of finding the largest-determinant submatrix of a matrix is NP-hard [ÇM09]. To circumvent this impossibility result, we can instead use the determinants to define a *sampling distribution*, motivating the following definition.

Definition 3.21 (Fixed-size DPP). Let $A \in \mathbb{K}^{n \times n}$ be a psd matrix, and fix a number $1 \le k \le n$. A *determinantal point process of fixed size k* or *k-DPP* is a random subset $\mathsf{S} \subseteq \{1, \ldots, n\}$ of size k with distribution

$$\mathbb{P}\{\mathsf{S} = \mathsf{T}\} = \frac{\det[A(\mathsf{T}, \mathsf{T})]}{\sum_{|\mathsf{R}|=k} \det[A(\mathsf{R}, \mathsf{R})]} \quad \text{for each subset } \mathsf{T} \subseteq \{1, \ldots, n\} \text{ of } k \text{ elements.}$$

We write a k-DPP S as $\mathsf{S} \sim \mathrm{DPP}_k(A)$.

Determinantal point processes have a long history. DPPs were originally introduced in 1975 by Macchi [Mac75] in the context of quantum statistical physics, and they have been studied by mathematicians for decades. There are many subvarietals of DPPs. In particular, the fixed-size DPPs in Definition 3.21 were introduced in 2011 by Kulesza and Taskar as a more useful construction for use in machine learning [KT11]. Further, Kulesza and Taskar's definition is equivalent to the *volume sampling distribution* introduced five years earlier by Deshpande, Rademacher, Vempala, and Wang [DRVW06] applied to a Gram square root of A. Motivated by subset selection problems in machine learning, interest in DPPs has exploded over the past two decades [KT12]. More recently, DPPs have been deployed as a tool in matrix computations [DM21]. See the surveys [KT12; DM21] for more on DPPs.

Pivot sets selected from a k-DPP produce excellent Nyström approximations. We have the following result:

Fact 3.22 (k-DPPs for Nyström approximation). *Let $A \in \mathbb{K}^{n \times n}$ be a psd matrix, and let $1 \leq k \leq n$. Draw a pivot set $\mathsf{S} \sim \mathrm{DPP}_k(A)$. The trace error of the Nyström approximation admits the exact formula*

$$\mathbb{E}\,\mathrm{tr}(A - A\langle \mathsf{S}\rangle) = (k+1)\frac{\mathrm{e}_{k+1}(\lambda_1(A), \ldots, \lambda_n(A))}{\mathrm{e}_k(\lambda_1(A), \ldots, \lambda_n(A))}.$$

Here, e_j is the jth elementary symmetric polynomial. For any $0 \leq r \leq k$, this error may be bounded as

$$\mathbb{E}\,\mathrm{tr}(A - A\langle \mathsf{S}\rangle) \leq \frac{k+1}{k-r+1}\,\mathrm{tr}(A - [\![A]\!]_r).$$

Consequently, k-DPP sampling produces an (r, ε)-approximation for any k satisfying $k \geq r/\varepsilon + r - 1$ for any $\varepsilon \in (0, r]$.

An analog of this general bound was first established for column projection approximations computed by volume sampling distribution of Deshpande et al. [DRVW06] by Guruswami and Sinop [GS12]. Using the Gram correspondence, it was transplanted to Nyström approximation by k-DPP sampling in [DM21]. The case $r = k$ was proven first by [DRVW06] (for column projection approximations and volume sampling) and [BW09b] (for Nyström approximation by k-DPPs).

Fact 3.22 achieves the strongest theoretical bounds (and indeed, the strongest *existence results*) for any column Nyström method. But turning DPP sampling into a computational strategy is a challenging enterprise. Standard algorithms for sampling a general k-DPP expend $O(n^3)$ operations and require a full decomposition of

the matrix (see, e.g., [KT11, §3]). Responding to this computational bottleneck, researchers investigated Markov chain Monte Carlo algorithms for *approximate* k-DPP sampling [AOR16; RO19]; even with efficient implementations, these algorithms are significantly more expensive than pivoted Cholesky methods. (The most recent algorithms [ALV22] do achieve the same *asymptotic* runtime as pivoted Cholesky algorithms, but they are complicated, and I am not aware of any evaluations of this approach for use in practice.) An alternate line of work has investigated exact k-DPP sampling algorithms that do not require reading the entire input matrix [Der19; DCV19; CDV20]. These algorithms are impressive, but they are more expensive in both runtime and number of entry evaluations than pivoted Cholesky methods. In my experience with the k-DPP software package [GPBV19], k-DPP sampling has been significantly slower and more resource intensive than other methods for Nyström approximation. In addition, I have found this software tends to fail by throwing exceptions on challenging problems. Therefore, while k-DPPs are mathematically beautiful and set the mathematical standard to which other methods are compared, I have not found k-DPP sampling to be a competitive approach to large-scale psd matrix approximation with the available algorithms and software implementations.

C h a p t e r   4

RANDOMLY PIVOTED CHOLESKY

> *So first she tasted the porridge of the Great, Huge Bear, and that was too hot for her. And then she tasted the porridge of the Middle Bear, and that was too cold for her. And then she went to the porridge of the Little, Small, Wee Bear, and tasted that; and that was neither too nor too cold, but just right, and she liked it so well that she ate it all up.*

> *The Story of the Three Bears* (1905)

Last chapter, we introduced the pivoted partial Cholesky decomposition as a way of computing a low-rank approximation to a psd matrix, and we saw two extreme strategies, uniform sampling and greedy selection, for selecting the pivots. The greedy strategy always selects the largest diagonal entry of the residual as pivot, and the uniform strategy selects pivots at random without using any information about the diagonal entries sizes. This chapter will strike a balance between these approaches, selecting a random pivot at each iteration using sampling probabilities weighted by the diagonal entries. The resulting algorithm is called *randomly pivoted Cholesky* (RPCHOLESKY).

Sources. This paper is based on the randomly pivoted Cholesky paper:

Yifan Chen, Ethan N. Epperly, Joel A. Tropp, and Robert J. Webber. "Randomly Pivoted Cholesky: Practical Approximation of a Kernel Matrix with Few Entry Evaluations". In: *Communications on Pure and Applied Mathematics* 78.5 (2025), pp. 995–1041. DOI: 10.1002/cpa.22234.

Outline. Section 4.1 introduces the randomly pivoted Cholesky algorithm and discusses its implementation. Section 4.2 provides numerical experiments and Section 4.3 discusses analysis. We conclude by discussing an extension to Gibbs RPCHOLESKY (Section 4.4) and connections between RPCHOLESKY and DPPs (Section 4.5).

4.1 Algorithm and implementation

Randomly pivoted Cholesky (RPCHOLESKY) is an algorithm for low-rank approximation of psd matrices. It employs a pivoted partial Cholesky decomposition, but

is distinguished from other pivoted Cholesky methods by using the diagonal of the residual matrix each iteration as a *sampling distribution* to select the next pivot. Conceptually, RPCHOLESKY executes the following iteration. With initial residual $A^{(0)} := A$, do the following for $i = 1, 2, \ldots, k$:

1. **Draw a random pivot.** Draw a *random* pivot index

$$s_i \sim \mathbf{diag}(A^{(i-1)}).$$

   This random pivoting strategy preferentially selects larger diagonal entries as pivots, but has a nonzero probability of selecting any nonzero diagonal entry as pivot. (Recall that we have defined $s \sim w$ as a random index sampled from the *unnormalized* weight vector w. That is, $\mathbb{P}\{s = j\} = w_j / \sum_{i=1}^{n} w_i$.)

2. **Rescale.** Extract and rescale the pivot column

$$f_i := a_{s_i}^{(i-1)} / (a_{s_i s_i}^{(i-1)})^{1/2}.$$

3. **Eliminate.** Update the residual $A^{(i)} := A^{(i-1)} - f_i f_i^*$, zeroing out the matrix in the s_ith row and column.

The outputs of RPCHOLESKY are a factor matrix $F \in \mathbb{K}^{n \times k}$ defining a low-rank approximation $FF^* \approx A$ and a set $\mathsf{S} = \{s_1, \ldots, s_k\}$ of pivot entries.

In practice, we do not update the entire residual matrix at every iteration, instead tracking the diagonal $\mathbf{diag}(A^{(i)})$ and generating entries from $A^{(i)}$ as needed using the formula $A^{(i)} = A - F(:, 1 : i)F(:, 1 : i)^*$, as in Program 3.2. With these optimizations, RPCHOLESKY reads $(k+1)n$ entries of the matrix and expends $O(k^2 n)$ arithmetic operations. See Program 4.1 for RPCHOLESKY code. More efficient block versions of RPCHOLESKY will be developed in Chapter 8.

The RPCHOLESKY algorithm can be seen as a midpoint between greedy selection and uniform sampling. The greedy method is based on entirely *exploiting* large diagonal entries, without *exploring* smaller diagonal entries as possible pivots. Conversely, uniform sampling randomly explores the set of all pivots but does not exploit information about the size of the diagonal entries. RPCHOLESKY combines both exploration and exploitation, making it more robust than either strategy individually.

Another way of interpreting the RPCHOLESKY method comes from the Gaussian framing developed in Section 3.6. Introduce a Gaussian vector $z \sim \text{NORMAL}_{\mathbb{K}}(0, A)$

Program 4.1 `rpcholesky.m`. Randomly pivoted Cholesky for psd low-rank approximation and column subset selection.

```matlab
function [F,S] = rpcholesky(Acol,d,k)
% Input:  Function Acol for producing columns Acol(i) = A(:,i) of A,
%         diagonal d of A, rank k
% Output: Factor F defining a rank-k approximation Ahat = F*F'

F = zeros(length(d),k);                    % To store output
S = zeros(k,1);                            % To store pivots
for i = 1:k
    % Random sample using current diagonal as sampling weights
    S(i) = datasample(1:length(d),1,"Weights",d);
    as = Acol(S(i)) - F(:,1:i-1)*F(S(i),1:i-1)'; % sth col of A-F*F'
    F(:,i) = as / sqrt(as(S(i)));          % Rescale
    d = d - abs(F(:,i)).^2;                % Update diagonal
    d = max(d,0); % Ensure nonnegative diagonal in floating point
end

end
```

with covariance matrix A. As in Section 3.6, we adopt an *experimental design* perspective where we wish to identify a set of coordinates S that minimizes the sum of conditional variances

$$\operatorname{tr}(A - A\langle \mathsf{S}\rangle) = \sum_{j=1}^{n} \operatorname{Var}(z_j \mid z(\mathsf{S}))$$

for the unseen coordinates. The greedy method builds the subset S one pivot at a time, always choosing the *maximum-variance* coordinate as pivot

$$s_{i+1} \in \underset{1 \le j \le n}{\operatorname{argmax}} \operatorname{Var}(z_j \mid z(\{s_1, \ldots, s_i\})).$$

The RPCHOLESKY method instead uses the variances as a *sampling distribution*:

$$\mathbb{P}\{s_{i+1} = j\} = \frac{\operatorname{Var}(z_j \mid z(\{s_1, \ldots, s_i\}))}{\sum_{\ell=1}^{n} \operatorname{Var}(z_\ell \mid z(\mathsf{S}))}.$$

Randomizing the procedure in this way balances the need to explore and exploit, and makes the method robust to bad instances like (3.8) where the greedy method can be fooled by tiny differences in variances between coordinates.

Software. A high-performance implementation of a version of RPCHOLESKY is under development in the RandLAPACK software project [MDME+23]. At present it may be found at

```
https://github.com/BallisticLA/RandLAPACK/blob/main/Rand
LAPACK/comps/rl_rpchol.hh
```

4.2 Experiments

We will see several comparisons of RPCHOLESKY with alternative methods throughout this part of the thesis. Here, we provide one initial set of data on RPCHOLESKY's performance, reproduced from [CETW25, Tab. 1]. Here, we evaluate the relative trace error of RPCHOLESKY against other column selection methods for approximation of psd kernel matrices associated with 20 datasets. (See Chapter 5 for an introduction to kernel matrices.) In addition to RPCHOLESKY, we test greedy selection and uniform and ridge leverage score (RLS) sampling. We omit DPP sampling because of its high computational cost. The Schmidt–Mirsky–Eckart–Young-optimal approximation error is shown for reference. As the optimal approximation is typically not a column Nyström approximation, it provides a lower limit on the best-possible approximation error for a column Nyström approximation that *typically cannot be attained*.

Results are shown in Table 4.1. The performance of RPCHOLESKY is uniformly good, achieving the lowest trace error on all twenty examples. We also see that RPCHOLESKY consistently achieves error close to the Schmidt–Mirsky–Eckart–Young optimal low-rank approximation. These experiments confirm that RPCHOLESKY is among the best available approaches for constructing a column Nyström approximation to a psd matrix. We will see further examples of RPCHOLESKY's success throughout this part of the thesis.

4.3 Error analysis

As we have seen and will continue to see, the RPCHOLESKY method consistently produces near-optimal low-rank approximations, often more accurate than competing methods by orders of magnitude. This excellent performance can be supported by theoretical analysis. This section will present analysis of the approximation of RPCHOLESKY in the trace norm and a weak (but still useful!) result for the spectral norm. It will then provide proofs of these results.

Trace-norm bounds

Our first bounds characterize the number of iterations needed to produce a good low-rank approximation to a psd matrix when measured using the trace norm.

Table 4.1: Relative trace error of rank-1000 Nyström approximation of psd kernel matrices by four alternative column selection methods: uniform and ridge leverage score (RLS) sampling, greedy selection, and RPCHOLESKY. Each trace error is computed as a median of ten trials. Error for the optimal rank-1000 approximation is shown for reference. Data is taken from [CETW25, Tab. 1].

	Uniform	RLS	Greedy	RPCHOL	Optimal
sensit_vehicle	1.57e-1	1.40e-1	2.07e-1	**1.37e-1**	8.77e-2
yolanda	1.46e-1	1.39e-1	2.08e-1	**1.36e-1**	8.41e-2
YearPredictionMSD	1.30e-1	1.20e-1	1.73e-1	**1.16e-1**	6.79e-2
w8a	1.42e-1	1.17e-1	1.91e-1	**1.05e-1**	6.09e-2
MNIST	1.21e-1	1.10e-1	1.67e-1	**1.06e-1**	5.83e-2
jannis	1.11e-1	1.10e-1	1.28e-1	**1.09e-1**	5.45e-2
HIGGS	6.73e-2	6.31e-2	8.51e-2	**6.07e-2**	2.96e-2
connect_4	5.81e-2	5.07e-2	6.48e-2	**4.81e-2**	2.25e-2
volkert	5.35e-2	4.42e-2	5.92e-2	**4.17e-2**	1.99e-2
creditcard	4.83e-2	3.71e-2	5.77e-2	**3.01e-2**	1.31e-2
Medical_Appointment	1.74e-2	1.43e-2	1.92e-2	**1.29e-2**	4.59e-3
sensorless	1.20e-2	7.78e-3	8.70e-3	**5.80e-3**	2.11e-3
ACSIncome	9.93e-3	5.55e-3	8.35e-3	**4.02e-3**	1.27e-3
Airlines_DepDelay_1M	4.19e-3	2.37e-3	2.64e-3	**1.78e-3**	5.08e-4
covtype_binary	9.10e-3	2.12e-3	1.41e-3	**1.04e-3**	2.97e-4
diamonds	1.31e-3	2.40e-4	1.12e-4	**5.85e-5**	1.30e-5
hls4ml_lhc_jets_hlf	3.78e-4	7.30e-5	6.68e-5	**4.38e-5**	1.04e-5
ijcnn1	3.91e-5	3.09e-5	2.86e-5	**2.13e-5**	4.67e-6
cod_rna	6.00e-4	1.36e-5	9.19e-6	**5.05e-6**	9.86e-7
COMET_MC_SAMPLE	3.65e-3	2.44e-7	1.2e-10	**4.3e-11**	3.5e-12

Theorem 4.1 (Randomly pivoted Cholesky: Trace norm). *Let $r \geq 1$, $\varepsilon > 0$, and $A \in \mathbb{K}^{n \times n}$ be a psd matrix. Introduce the relative error of the best rank-r approximation*

$$\eta := \operatorname{tr}(A - [\![A]\!]_r)/\operatorname{tr}(A).$$

Randomly pivoted Cholesky produces an (r, ε)-approximation (Definition 3.6) to A as long as the number of steps satisfies

$$k \geq \frac{r}{\varepsilon} + r \log\left(\frac{1}{\varepsilon \eta}\right). \tag{4.1}$$

We observe that RPCHOLESKY achieves theoretical guarantees that are nearly optimal within the class of Nyström approximations. (Recall from Fact 3.11 that at least $k \geq r/\varepsilon$ columns are needed to produce an (r, ε)-approximation to a worst-case psd matrix.)

The dependence of the number of columns k on the relative error η is significantly improved for RPCHOLESKY over diagonal sampling (Section 3.7), which requires $\Theta(r/\eta)$ column accesses to produce an approximation comparable with the best rank-r approximation for some input matrices. RPCHOLESKY yields an *exponential improvement* in the dependence on $1/\eta$.

The logarithmic factor $\log(1/\eta)$ is, for the purposes of practical computation, a modest constant. Due to numerical errors, the relative error η is effectively bounded from below by the *unit roundoff u*, which captures the size of rounding errors ($u \approx 10^{-16}$ in double precision). Thus, in double precision, $\log(1/\eta) \lessgtr 37$.

Remark 4.2 (What if the relative error is small?). One somewhat unappealing feature of this result is that when A is rank-r, the relative error is $\eta = 0$ and the right-hand side of (4.1) becomes infinite. Here, the bound (4.1) badly mischaracterizes the actual behavior of the RPCHOLESKY algorithm, which recovers a rank-r matrix A *with zero error* after precisely $k = \text{rank } A$ steps. Developing improved bounds for small η is an open problem; see Section 11.1 for discussion.   ◇

Weak spectral norm-type bounds

It is natural to desire error bounds for RPCHOLESKY that hold in the spectral norm. Perhaps, similar to Fact 3.17, we could show that RPCHOLESKY achieves spectral-norm error λ in roughly $d_{\text{eff}}(\lambda)$ steps, where $d_{\text{eff}}(\lambda)$ denotes the λ-effective dimension (Definition 3.16). While such bounds are not yet known (see Section 11.1), we can establish bounds on the *spectral norm of the expected error*:

Theorem 4.3 (Randomly pivoted Cholesky: Spectral norm). *Fix parameters $b > 0$ and $\varepsilon > 0$. For any psd matrix A, the k-step residual $A^{(k)}$ of RPCHOLESKY satisfies*

$$\left\| \mathbb{E}\, A^{(k)} \right\| \leq b + \varepsilon \operatorname{tr}(A - [\![A]\!]_r)$$

provided the number of steps k satisfies

$$k \geq \frac{1}{\varepsilon} + r \log\left(\frac{\|A\|}{b}\right).$$

This result has a significant limitation in that the expectation occurs *inside* the norm. Pulling the expectation inside the norm can be done at a great cost. Indeed, for any random psd matrix X, we have the bound

$$\|\mathbb{E}\, X\| \leq \mathbb{E}\|X\| \leq \mathbb{E}\operatorname{tr}(X) = \operatorname{tr}(\mathbb{E}\, X) \leq n \cdot \|\mathbb{E}\, X\|. \tag{4.2}$$

The first inequality is Jensen's. The factor n in the upper bound $\mathbb{E}\|X\| \leq n \cdot \|\mathbb{E} X\|$ is sharp, as demonstrated by the matrix $X = \mathbf{e}_i \mathbf{e}_i^*$ for $i \sim \text{UNIF}\{1, \ldots, n\}$.

The main use of Theorem 4.3 is to bound quadratic forms $x^* A^{(k)} x$ in the residual matrix:

$$\mathbb{E}[x^* A^{(k)} x] = x^* [\mathbb{E} A^{(k)}] x \leq \|x\|^2 \cdot \|\mathbb{E} A^{(k)}\|.$$

We will use this type of bound to analyze quadrature methods in Section 7.5.

Ignoring the (significant) norm of expectation vs. expectation of norm issue, how good is Theorem 4.3? To get some insight into this question, consider a matrix with rapidly polynomially decaying eigenvalues $\lambda_j(A) = j^{-q}$ for a *fixed* parameter $q > 1$. The sum of tail eigenvalues is

$$\sum_{j=r+1}^{n} \lambda_j(A) \leq \sum_{j=r+1}^{\infty} j^{-q} = O(r^{1-q}).$$

Setting $\varepsilon = O(1/r)$ and $b = (r+1)^{-q}/2$, we obtain

$$\left\| \mathbb{E} A^{(k)} \right\| \leq \lambda_{r+1}(A) = (r+1)^{-q} \quad \text{after } k = O(r \log r) \text{ steps}.$$

RPCHOLESKY requires at most $O(r \log r)$ steps for the spectral norm of the expected residual to drop below the spectral norm error of the best rank-r approximation—not bad!

Proofs

Let us begin with the proof of Theorems 4.1 and 4.3. As a first step, we introduce the *expected residual function*

$$\Phi(A) := \mathbb{E}[A^{(1)}]$$

which measures the expected value of the residual matrix $A^{(1)}$ after one step of RPCHOLESKY applied to input matrix A. By direct computation, we observe

$$\Phi(A) = \sum_{j=1}^{n} \left(A - \frac{a_j a_j^*}{a_{jj}} \right) \cdot \mathbb{P}\{s_1 = j\} = A - \sum_{j=1}^{n} \frac{a_j a_j^*}{a_{jj}} \frac{a_{jj}}{\text{tr}(A)} = A - \frac{A^2}{\text{tr}(A)}.$$

The map Φ enjoys a number of properties.

Proposition 4.4 (Expected residual function). *The expected residual function* Φ *satisfies the following properties:*

(a) **Unitarily covariant:** *For a unitary matrix* $U \in \mathbb{K}^{n \times n}$ *and a psd matrix* A,
$$\Phi(UAU^*) = U\Phi(A)U^*.$$

(b) **Positive:** *For any psd matrix* A, $\Phi(A)$ *is psd.*

(c) **Concave:** *The map* Φ *is concave with respect to the psd order.* *For psd matrices* A, H,
$$\Phi(\theta A + (1 - \theta)H) \succeq \theta\Phi(A) + (1 - \theta)\Phi(H) \quad \text{for each } \theta \in [0, 1]. \quad (4.3)$$

(d) **Monotone:** *The map* Φ *is monotone with respect to the psd order.* *For psd matrices* A, H,
$$\Phi(A + H) \succeq \Phi(A). \tag{4.4}$$

Proof. Throughout this proof, let A and H denote psd matrices.

The unitary covariance property (a) is immediate from the formula $\Phi(A) = A - A^2/\mathrm{tr}(A)$.

The positivity property (b) follows by observing $\Phi(A)$ has nonnegative eigenvalues
$$\lambda_i(\Phi(A)) = \lambda_i(A) - \frac{\lambda_i(A)^2}{\mathrm{tr}(A)} = \lambda_i(A)\left(1 - \frac{\lambda_i(A)}{\sum_{j=1}^n \lambda_j(A)}\right) \geq 0$$

for each $i = 1, 2, \ldots, n$.

The concavity claim (c) follows from direct computation: For $\theta \in [0, 1]$ and $\overline{\theta} = 1 - \theta$, we have

$$\Phi(\theta A + \overline{\theta}H) - \theta\Phi(A) - \overline{\theta}\Phi(H) = \frac{\theta\overline{\theta}}{\theta\,\mathrm{tr}(A) + \overline{\theta}\,\mathrm{tr}(H)}\left(\sqrt{\frac{\mathrm{tr}(H)}{\mathrm{tr}(A)}}A - \sqrt{\frac{\mathrm{tr}(A)}{\mathrm{tr}(H)}}H\right)^2,$$

which is manifestly psd.

To show the monotonicity property (d), first observe that Φ is positive homogeneous $\Phi(\alpha \cdot A) = \alpha \cdot \Phi(A)$ for $\alpha \geq 0$. Consequently, by (4.3),

$$\Phi(A + H) = 2\Phi(0.5A + 0.5H) \succeq \Phi(A) + \Phi(H) \succeq \Phi(A).$$

The last inequality holds since $\Phi(H)$ is psd. We have established (4.4). $\qquad\square$

Because of the monotonicity and concavity of the expected residual function, we can bound the i-step residual $A^{(i)}$ by iterating the expected one-step residual function:

Proposition 4.5 (Multistep residual). *Let $A^{(i)}$ denote the residual of the matrix A after running i steps of RPCHOLESKY. Then*

$$\mathbb{E}[A^{(i)}] \preceq \Phi^i(A).$$

Here, Φ^i denotes the i-fold composition of Φ.

Proof. First, apply an appropriate matrix version of Jensen's inequality [Car10, Thm. 4.16] conditionally on the $(i-1)$-step residual $A^{(i-1)}$ to obtain

$$\mathbb{E}[A^{(i)}] = \mathbb{E}[\mathbb{E}[A^{(i)} \mid A^{(i-1)}]] = \mathbb{E}[\Phi(A^{(i-1)})] \preceq \Phi(\mathbb{E}[A^{(i-1)}]).$$

Here, we used concavity (4.3) of Φ. Next, we iterate this inequality using the monotonicity property (4.4):

$$\mathbb{E}[A^{(i)}] \preceq \Phi(\mathbb{E}[A^{(i-1)}]) \preceq \Phi^2(\mathbb{E}[A^{(i-2)}]) \preceq \cdots \preceq \Phi^{i-1}(\mathbb{E}[A^{(1)}]) \preceq \Phi^i(A).$$

The desired claim is established. □

Proposition 4.5 shows that the eigenvalues of $\mathbb{E}[A^{(k)}]$ are bounded above by the eigenvalues of the k-fold composition of the Φ transformation to the matrix A. Introduce the *vector* function $\phi : \mathbb{R}_+^n \to \mathbb{R}_n^+$

$$\phi(\alpha) = \alpha - \frac{\alpha^2}{\sum_{j=1}^n \alpha_j}.$$

Here, α^2 denotes the entrywise square. The function ϕ tracks the eigenvalues after applying the one-step residual function Φ:

$$\lambda(\Phi(A)) = \phi(\lambda(A)) \quad \text{for a psd matrix} A.$$

Consequently,

$$\lambda(\mathbb{E}[A^{(i)}]) \leq \lambda(\Phi^i(A)) = \phi^i(\lambda(A)). \tag{4.5}$$

In this display, ϕ^k is the i-fold composition of ϕ, and the inequality holds entrywise. The inequality in (4.5) follows by Proposition 4.5 and Weyl monotononicity principle [Tro22, Cor. 9.9]:

$$A \preceq H \implies \lambda(A) \leq \lambda(H).$$

The equality in (4.5) is unitary covariance (Proposition 4.4(a)). We now provide a proof for Theorems 4.1 and 4.3.

Proof of Theorems 4.1 and 4.3. Introduce $\lambda^{(i)} := \phi^i(\lambda(A))$ to track the right-hand side of (4.5). Since $\lambda^{(i)}$ is defined as the iteration of the ϕ map, it obeys the recurrence

$$\lambda^{(i+1)} = \phi(\lambda^{(i)}) = \lambda^{(i)} - \frac{[\lambda^{(i)}]^2}{\sum_{j=1}^n \lambda_j^{(i)}}. \tag{4.6}$$

We make two observations. First, we the vectors $\lambda^{(i)}$ are entrywise decreasing: For each i, $\lambda^{(i+1)} \leq \lambda^{(i)}$. This observation is immediate from the recurrence (4.6). Second, the entries of each vector $\lambda^{(i)}$ are sorted in nonincreasing order for each i.

Let us verify this second observation by induction. For the base case $i = 0$, note that $\lambda^{(0)} = \lambda(A)$ is sorted in nonincreasing order by definition. Next, inductively suppose that $\lambda^{(i-1)}$ is sorted in nonincreasing order. Then for $j \geq 1$,

$$\lambda_{j+1}^{(i)} - \lambda_j^{(i)} = \left[\lambda_{j+1}^{(i-1)} - \lambda_j^{(i-1)}\right] - \frac{\left[(\lambda_{j+1}^{(i-1)})^2 - (\lambda_j^{(i-1)})^2\right]}{\sum_{\ell=1}^n \lambda_\ell^{(i-1)}}$$

$$= \left[\lambda_{j+1}^{(i-1)} - \lambda_j^{(i-1)}\right]\left(1 - \frac{\lambda_{j+1}^{(i-1)} + \lambda_j^{(i-1)}}{\sum_{\ell=1}^n \lambda_\ell^{(i-1)}}\right) \geq 0.$$

This completes the inductive argument, showing that the entries of $\lambda^{(i)}$ are sorted in nonincreasing order for every i.

Having established some basic properties of the recurrence (4.6), we now shall reason about the average

$$a^{(i)} := \frac{1}{\ell} \sum_{j=1}^\ell \lambda_j^{(i)}$$

of the top ℓ eigenvalues of $\lambda^{(i)}$. We take the parameter ℓ to be between 1 and r, and we will primarily be interested in the edge cases $\ell = 1$ (top eigenvalue) and $\ell = r$ (top r eigenvalues). By averaging the first ℓ entries of the recurrence (4.6), we obtain a recurrence for $a^{(i)}$:

$$a^{(i+1)} = a^{(i)} - \frac{\ell^{-1} \sum_{j=1}^\ell [\lambda_j^{(i)}]^2}{\sum_{j=1}^n \lambda_j^{(i)}}. \tag{4.7}$$

We may bound the dominator of the second term as

$$\sum_{j=1}^n \lambda_j^{(i)} = \sum_{j=1}^r \lambda_j^{(i)} + \sum_{j=r+1}^n \lambda_j^{(i)} \leq ra^{(i)} + \sum_{j=r+1}^n \lambda_j(A).$$

Here, we use the fact that the numbers $\lambda_j^{(i)}$ are nonincreasing in both i and j together with the initial condition $\lambda_j^{(0)} = \lambda_j(A)$. To bound the numerator of the second term of (4.7), we use Jensen's inequality,

$$\frac{1}{\ell} \sum_{j=1}^{\ell} \left[\lambda_j^{(i)}\right]^2 \geq \left[\frac{1}{\ell} \sum_{j=1}^{\ell} \lambda_j^{(i)}\right]^2 = \left[a^{(i)}\right]^2 .$$

Substituting the two previous displays into (4.7) yields

$$a^{(i+1)} \leq a^{(i)} - \frac{\left[a^{(i)}\right]^2}{r a^{(i)} + \sum_{j=r+1}^{n} \lambda_j} . \tag{4.8}$$

To bound the recurrence (4.8), we compare to an ODE model

$$\frac{\mathrm{d}}{\mathrm{d}t} x(t) = -\frac{x(t)^2}{r x(t) + \sum_{j=r+1}^{n} \lambda_j} \quad \text{with } x(0) = \frac{1}{\ell} \sum_{j=1}^{\ell} \lambda_j . \tag{4.9}$$

For each i, we have $a^{(i)} \leq x(i)$ because $x \mapsto -x^2/(rx + \sum_{j=r+1}^{n} \lambda_j)$ is decreasing on $\mathbb{R}_+$. Fixing a level $\gamma > 0$, we may solve (4.9) to obtain the time $t_\star$ at which $x(t_\star) = \gamma$ using separation of variables:

$$
\begin{aligned}
t_\star &= \int_{\gamma}^{\ell^{-1} \sum_{j=1}^{\ell} \lambda_j} \frac{rx + \sum_{j=r+1}^{n} \lambda_j}{x^2} \, \mathrm{d}x \\
&= \frac{\sum_{j=r+1}^{n} \lambda_j}{\gamma} - \frac{\ell \sum_{j=r+1}^{n} \lambda_j}{\sum_{j=1}^{\ell} \lambda_j} + r \log\left(\frac{\sum_{j=1}^{\ell} \lambda_j}{\ell \gamma}\right) \\
&\leq \frac{\operatorname{tr}(A - [\![A]\!]_r)}{\gamma} + r \log\left(\frac{\sum_{j=1}^{\ell} \lambda_j}{\ell \gamma}\right) .
\end{aligned}
\tag{4.10}
$$

In the last inequality, we recall that $\operatorname{tr}(A - [\![A]\!]_r) = \sum_{j=r+1}^{n} \lambda_j$.

We can use (4.10) in two ways. First consider the case when $\ell = r$ and bound

$$
\begin{aligned}
\mathbb{E} \operatorname{tr}(A^{(k)}) &= \sum_{j=1}^{r} \lambda_j(\mathbb{E}[A^{(k)}]) + \sum_{j=r+1}^{n} \lambda_j(\mathbb{E}[A^{(k)}]) \\
&\leq \sum_{j=1}^{r} \lambda_j^{(k)} + \sum_{j=r+1}^{n} \lambda_j^{(k)} \leq r a^{(i)} + \operatorname{tr}(A - [\![A]\!]_r) .
\end{aligned}
$$

The first inequality uses (4.5) and instates the definition of $\lambda_j^{(i)}$, and the second inequality uses the definition of $a^{(i)}$, the fact that $\lambda_j^{(i)}$ is nonincreasing in i, and the

boundary condition $\lambda_j^{(0)} = \lambda_j(A)$. Apply (4.10) with $\ell = r$ and $\gamma := \varepsilon/r \cdot \text{tr}(A - [\![A]\!]_r)$ to conclude that $\mathbb{E}\,\text{tr}(A^{(k)}) \leq (1 + \varepsilon)\,\text{tr}(A - [\![A]\!]_r)$ when

$$k \geq \frac{r}{\varepsilon} + r \log\left(\frac{\text{tr}([\![A]\!]_r)}{\varepsilon\,\text{tr}(A - [\![A]\!]_r)}\right).$$

This result is stronger than as stated in Theorem 4.1. Second, consider the case $\ell = 1$ and set $\gamma := b + \varepsilon\,\text{tr}(A - [\![A]\!]_r)$. Then (4.5) and (4.10) imply that

$$\left\|\mathbb{E}(A^{(k)})\right\| = \lambda_1(\mathbb{E}[A^{(k)}]) \leq \lambda_1^{(k)} \leq b + \varepsilon\,\text{tr}(A - [\![A]\!]_r)$$

when

$$k \geq \frac{\text{tr}(A - [\![A]\!]_r)}{b + \varepsilon\,\text{tr}(A - [\![A]\!]_r)} + r \log\left(\frac{\|A\|}{b + \text{tr}(A - [\![A]\!]_r)}\right).$$

This is stronger than the result of Theorem 4.3. $\square$

4.4 Extension: Gibbs RPCHOLESKY

The uniform, RPCHOLESKY, and greedy pivoting strategies can be unified into a common framework. The Gibbs RPCHOLESKY method [CETW25, §2.3.4] selects a random pivot at each iteration according to the rule

$$s_{i+1} \sim \mathbf{diag}(A^{(i)})^p.$$

We remind the reader that $s \sim w$ denotes a random sample from the (unnormalized) weight vector w. The power $p \in (0, \infty)$ controls the level of "greediness". The extreme case $p \downarrow 0$ corresponds to uniform sampling from the support of $\mathbf{diag}(A^{(i)})$; the other extreme $p \uparrow +\infty$ selects the largest diagonal entries, with exact ties broken uniformly at random. We recognize the cases $p = 0$ and $p = \infty$ are variants of uniform and greedy selection. RPCHOLESKY sits at the intermediate value $p = 1$.

As the name suggests, the Gibbs RPCHOLESKY method samples a pivot index from a Gibbs distribution $\mathbb{P}\{s_{i+1} = j\} = \exp(-\beta v_j)$ with "energies" $v_j = -\log a_{jj}^{(i)}$ and "inverse-temperature" $\beta = p$. For this reason, the symbol β has been used to refer to the power p in previous literature [Ste24; CETW25]. Under the interpretation of $\beta = p$ as inverse-temperature, the uniform, greedy, and RPCHOLESKY strategies can be analogized to the three bowls of porridge in the story of the three bears—too hot, too cold, and just right. The recent paper [DPPL24] by Dong, Pan, Phan, and Lei has explored an alternate definition of the energies $v_j = -a_{jj}^{(i)}$, leading to sampling probabilities $s_{i+1} \sim \exp(\beta\,\mathbf{diag}(A^{(i)}))$.

Our original paper [CETW25] introduced the Gibbs RPCHOLESKY algorithm but did not provide any numerical experiments. Stefan Steinerberger [Ste24] took up the task of empirically evaluating the Gibbs RPCHOLESKY method. Steineberger's experiments consider several examples, some of which benefit from higher p and others which favor lower p. However, except on specially constructed examples, the differences between different methods are not dramatic (particularly away from the extreme values $p \in \{0, \infty\}$). Steinerberger's paper also includes theoretical results, including a theorem showing that a single step of the $p = 2$ Gibbs RPCHOLESKY reduces the squared Frobenius norm of a psd matrix by at least a factor $1 - 1/n$.

My preliminary conclusion from Steinerberger and Dong et al.'s investigations is that, for non-pathological matrices, any sensible random pivoting selection *that incorporates information from the diagonal of the residual matrix* should yield decent performance for low-rank approximation. To achieve the best possible performance, one can treat the power p (or inverse-temperature β for the Dong et al. scheme) as a hyperparameter and test multiple p or β values.

Even though Gibbs RPCHOLESKY seems to work reliably on most matrices with $p > 1$, significant failures can occur on synthetic, worst-case examples. Fix $p > 1$, and consider the matrix

$$A = \begin{bmatrix} n^{1/p}\mathbf{I}_{100} & \mathbf{0} \\ \mathbf{0} & \mathbf{1}_{n-100}\mathbf{1}^*_{n-100} \end{bmatrix}.$$

The optimal rank-one approximation to this matrix is

$$[\![A]\!]_1 = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1}_{n-100}\mathbf{1}^*_{n-100} \end{bmatrix},$$

and this approximation is computed by a partial pivoted Cholesky decomposition in one step if any pivot $s \in \{101, 102, \dots, n\}$ is selected. For large n, this optimal rank-one approximation to this matrix has vanishingly small relative error

$$\frac{\operatorname{tr}(A - [\![A]\!]_1)}{\operatorname{tr}(A)} = \frac{100n^{1/p}}{n + 100(n^{1/p} - 1)]} = (100 + o(1))n^{1/p-1}.$$

and RPCHOLESKY produces this optimal approximation in one step with near-certainty:

$$\mathbb{P}\{s \geq 101\} = \frac{n - 100}{n + 100(n^{1/p} - 1)} = 1 - o(1) \quad \text{when } s \sim \mathbf{diag}(A).$$

With with Gibbs RPCHOLESKY with power p, a bad pivot is selected with roughly 99% probability:

$$\mathbb{P}\{s \leq 100\} = \frac{100n}{(n - 100) + 100n} = \frac{100}{101} + o(1).$$

For large n, it takes Gibbs RPCHOLESKY with power p roughly 50 steps to compute a good low-rank approximation to this matrix A, where RPCHOLESKY ($p = 1$) produces an excellent approximation with near certainty in a single step. Given bad examples like these, we believe the power $p = 1$ is a sensible default for general-purpose use, particular if only a single p value is to be used.

4.5   Connection to determinantal point processes

The RPCHOLESKY algorithm has a number of connections to (fixed-size) determinantal point processes (DPPs, Definition 3.21). This section reviews these connections.

RPCHOLESKY as iterative 1-DPP sampling. The first interpretation is the most trivial, but still yields some insight. A single step of RPCHOLESKY performs one step of diagonal sampling on the current residual matrix $A^{(i)}$, which coincides with the *1-DPP distribution* (since the determinant of a 1×1 matrix—i.e., a number—is just itself). As a consequence of this interpretation, we can derive a single-step RPCHOLESKY error bound by invoking Fact 3.22:

$$\mathbb{E}\operatorname{tr}(A^{(1)}) \leq 2\operatorname{tr}(A - [\![A]\!]_1).$$

RPCHOLESKY as iterative conditional DPP sampling. Another, much more powerful connection between RPCHOLESKY and DPPs interprets the RPCHOLESKY procedure as iteratively performing *conditional* DPP sampling.

Proposition 4.6 (RPCHOLESKY as conditional DPP sampling). *Suppose we have run the RPCHOLESKY algorithm for i steps on psd matrix A, selecting pivots $s_1, \ldots, s_i$. The $(i + 1)$st pivot $s_{i+1} \sim \mathbf{diag}(A^{(i)})$ satisfies*

$$\mathbb{P}\{s_{i+1} = j \mid s_1, \ldots, s_i\} = \mathbb{P}\left\{\mathsf{T} = \{s_1, \ldots, s_{i+1}\} \,\middle|\, \mathsf{T} \supseteq \{s_1, \ldots, s_i\}\right\},$$

where $\mathsf{T} \sim \mathrm{DPP}_{i+1}(A)$.

We see that the $(i + 1)$st step of RPCHOLESKY can be seen as sampling from an $(i + 1)$-DPP, *conditional* on the already-selected pivots belonging to that DPP. This result follows directly from the definition of fixed-size DPPs and Proposition 3.20.

This connection between DPPs and RPCHOLESKY suggests an *algorithm* for approximate k-DPP sampling. First, run RPCHOLESKY for k steps, producing pivots $s_1, \ldots, s_k$. Then, for steps $t = 0, 1, \ldots, T$, select a random pivot index $j \sim \text{UNIF}\{1, \ldots, k\}$, evict pivot s_j from the pivot set, and resample s_j using a single step of the RPCHOLESKY procedure

$$s_j \sim \mathbf{diag}(A - A\langle s_1, \ldots, s_{j-1}, s_{j+1}, \ldots, s_k \rangle).$$

The reason behind this procedure's success is intuitive. Initially, we generate the pivots $s_1, \ldots, s_k$ sequentially, and they do not follow the k-DPP distribution. At each step, we freeze all but one pivot s_j and resample s_j from the k-DPP distribution conditional on the other pivots $\{s_i : i \neq j\}$. By repeating this procedure enough time, the distribution of the pivots converges to the k-DPP distribution.

A sampling algorithm for a multivariate distribution of this type (freeze all but one coordinate and sample the conditional distribution) is known as a *Gibbs Markov chain Monte Carlo (MCMC) sampler*. This Gibbs DPP sampling procedure was proposed by Rezaei and Oveis Gharan [RO19] to sample from a generalization of k-DPPs that can be defined on general, possibly *continuous* state spaces. Fortunately, ordinary k-DPPs on the finite set $\{1, \ldots, n\}$ are contained in this general setting. Rezaei and Oveis Gharan's theoretical bounds show that the RPCHOLESKY-based Gibbs MCMC sampler converges in at most $T = O(k^5 \log k)$ steps, and their numerical results suggest a more modest $T = O(k^2)$ steps suffice. To implement their algorithm efficiently, one should use Cholesky downdating techniques to handle the pivot evictions; we omit the details.

To raise a potentially provocative point, even Rezaei and Oveis Gharan's empirical results suggest that sampling a k-DPP using this algorithm requires *quadratically* more work than executing RPCHOLESKY. It is natural to ask: Is that work worth it? Is the output of RPCHOLESKY substantially worse than a sample from a k-DPP for practical purposes? We leave these questions to the reader to ponder.

Projection DPPs. There is an important class of matrices for which RPCHOLESKY produces *exact* samples from a k-DPP. We make the following definition.

Definition 4.7 (Projection DPP)**.** Let A be an orthorpojector of *rank exactly k*. Then the k-DPP $\mathsf{S} \sim \text{DPP}_k(A)$ is referred to as a *projection DPP*.

We emphasize that, in a projection DPP, the rank of the orthoprojector A must be equal to the size k of the DPP S. Projection DPPs are fundamental to both the

theory and applications of DPPs. Theoretically, every (k-)DPP can be realized as a mixture of projection DPPs [HKPV06, p. 213], which gives rise to procedures for sampling from k-DPPs.

Remarkably, the RPCHOLESKY algorithm, applied to an orthoprojector, generates exact samples from a projection DPP.

Proposition 4.8 (RPCHOLESKY samples projection DPPs)**.** *Let A be an orthoprojector of rank exactly k. The pivot set S produced by k steps of RPCHOLESKY applied to A is a sample from the projection DPP $\mathsf{S} \sim \mathrm{DPP}_k(A)$.*

This proposition appears in its essence in Gillenwater's thesis [Gil14, sec. 2.2.4] and a very clear version of the RPCHOLESKY pseudocode (more-or-less the same as our Program 4.1) appears in the work of Poulson [Pou20]. We emphasize that Poulson only used the RPCHOLESKY procedure to sample from projection DPPs: In his work, the input matrix A is always an orthoprojector, he runs for exactly rank A steps, and the output is the pivot set S. The factor matrix F, critical to use cases of RPCHOLESKY for low-rank approximation, is discarded in Poulson's work.

Chapter 5

KERNELS AND GAUSSIAN PROCESSES

> *Although William of Occam first wielded his famous razor against the superfluous elaborations of his Scholastic predecessors, his principle of parsimony has since been incorporated into the methodology of experimental science in the following form: given two explanations of the data, all other things being equal, the simpler explanation is preferable. This principle is very much alive today in the emerging science of machine learning, whose expressed goal is often to discover the simplest hypothesis that is consistent with the sample data.*
>
> Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth, *Occam's razor* [BEHW87]

The twin theories of reproducing kernel Hilbert spaces and Gaussian processes are among the most beautiful topics in applicable mathematics, and they form the basis for a class of simple and user-friendly machine learning methods. In this section, we review these theories and how they are used in machine learning. In the following chapter, we explain how RPCHOLESKY can be used to accelerate these learning algorithms and provide computational experiments.

Sources. This section serves to provide an introduction to kernel methods and Gaussian processes. It reflects my own approach to this material and is not explicitly based on any of particular published work, of mine or others. The content of this section is influenced by Houman Owhadi's excellent class "Stochastic processes and regression", which I took in Winter of 2022. Useful references on this material include the books [SS02; RW05] and the survey [KHSS18].

Outline. Section 5.1 introduces the theory of reproducing kernel Hilbert spaces, and Section 5.2 uses this formalism to derive the *kernel interpolation* method for fitting data. Section 5.3 describes the parallel theory of Gaussian processes and reformulates kernel interpolation as Gaussian process interpolation. Section 5.4 concludes by discussing regularized kernel and Gaussian process methods.

5.1 Reproducing kernel Hilbert spaces

One of the great annoyances in mathematical analysis is the fact that functions $u \in L^2(\mathbb{R}^d)$ do not have definite values at any given point. Indeed, the space $L^2(\mathbb{R}^d)$ is formally defined to consist of all square-integrable functions on $\mathbb{R}^d$, *modulo almost everywhere equality* (with respect to the Lebesgue measure). So for any given $u \in L^2(\mathbb{R}^d)$, there are elements of the equivalence class of u taking every possible value at any given point x, since the singleton set $\{x\}$ has measure zero.

If we restrict ourselves to *continuous* functions $u \in C(\mathbb{R}^d) \cap L^2(\mathbb{R}^d)$, things are a bit better, since $u(x)$ has a defined value at each point $x \in \mathbb{R}^d$. However, even for continuous L^2 functions, function values are not *stable*: For any $\varepsilon > 0$, any point $x \in \mathbb{R}^d$, and any value $\alpha \in \mathbb{K}$, there exists a continuous perturbation e of norm $\|e\|_{L^2(\mathbb{R})^d} \leq \varepsilon$ such that $(u + e)(x) = \alpha$. The conclusion is that the value of any L^2 function at any point can be changed to any value by an arbitrarily small perturbation.

These limitations of the L^2 space motivates the concept of a reproducing kernel Hilbert space (RKHS), which may be informally defined as follows.

> An RKHS is a Hilbert space of functions that have definite values at every point, and these values are stable under small perturbations.

Formally, we have the following definition:

Definition 5.1 (Reproducing kernel Hilbert space)**.** Let X be a set. A *reproducing kernel Hilbert space* (RKHS) on X is a Hilbert space of functions $f : X \rightarrow \mathbb{K}$ over which the evaluation map $x \mapsto f(x)$ is a bounded linear functional for each x. That is, for each $x \in X$, there exists a prefactor $c(x) \in \mathbb{R}_+$ such that

$$|f(x)| \leq c(x)\|f\|_H \quad \text{for each } f \in H.$$

The definition of the RKHS encompasses (and is equivalent to) the notion of stability of function values under small perturbations. Indeed, for $e \in H$ in an RKHS H,

$$|(f + e)(x) - f(x)| = |e(x)| \leq c(x)\|e\|_H.$$

As we will see, the seemingly innocuous property of point evaluation being a bounded linear functional has surprising and far-reaching consequences.

Reproducing kernel Hilbert spaces (RKHSs) may initially appear exotic, but they encompass many of the fundamental function spaces in mathematical analysis.

Example 5.2 (A Sobolev space). Consider, for instance, the space $\mathsf{H}^1([0,1])$, consisting of all square-integrable functions with a square-integrable (weak) derivative. Its norm is

$$\|f\|_{\mathsf{H}^1([0,1])}^2 = \int_0^1 \left(|f(x)|^2 + |f'(x)|^2 \right) \, \mathrm{d}x.$$

This space is an RKHS. To see this, first consider a continuous representative $f \in \mathsf{H}^1([0,1]) \cap \mathsf{C}([0,1])$. By the intermediate value theorem, there exists a value $x_\star$ at which f achieves its mean value

$$f(x_\star) = \int_0^1 f(x) \, \mathrm{d}x.$$

By the Cauchy–Schwarz inequality,

$$|f(x_\star)|^2 = \left| \int_0^1 f(x) \, \mathrm{d}x \right|^2 \leq \int_0^1 |f(x)|^2 \, \mathrm{d}x \cdot \int_0^1 1 \, \mathrm{d}x = \int_0^1 |f(x)|^2 \, \mathrm{d}x.$$

For any two values $x \leq y$ in $[0,1]$,

$$|f(y) - f(x)|^2 = \left| \int_x^y f'(a) \, \mathrm{d}a \right|^2 \leq \int_x^y |f'(a)|^2 \, \mathrm{d}a \cdot \int_x^y 1 \, \mathrm{d}a \leq \int_0^1 |f'(a)|^2 \, \mathrm{d}a.$$

The same conclusion holds if $y \leq x$. Employing the two previous displays, we conclude that for any $x \in [0,1]$, we have

$$|f(x)|^2 \leq 2|f(x_\star)|^2 + 2|f(x) - f(x_\star)|^2$$

$$\leq 2 \int_0^1 \left(|f(x)|^2 + |f'(x)|^2 \right) \, \mathrm{d}x = 2\|f\|_{\mathsf{H}^1([0,1])}^2.$$

We have established the RKHS property with $c(x) \leq \sqrt{2}$ for all $x \in [0,1]$. This conclusion extends to all $f \in \mathsf{H}^1([0,1])$ by density of $\mathsf{C}([0,1])$ in $\mathsf{H}^1([0,1])$. We conclude that $\mathsf{H}^1([0,1])$ is an RKHS. ◇

This example is illustrative. It suggests that functions in an RKHS should be smoother (in the sense of possessing well behaved derivatives) than typical functions in the space $\mathsf{L}^2(\mu)$. This intuition will prove valuable.

The reproducing kernel

Whenever one has a linear functional over a Hilbert space, is is generally worth invoking the Riesz representation theorem to see if there are any interesting consequences. In the cases of RKHSs, this is most certainly the case.

For each point $x \in \mathsf{X}$, the Riesz representation theorem furnishes a function $\kappa(\cdot, x) \in \mathsf{H}$ for which

$$f(x) = \langle \kappa(\cdot, x), f \rangle_{\mathsf{H}}. \tag{5.1}$$

We have denoted this function by $\kappa(\cdot, x)$ to indicate that applying this construction at each $x \in \mathsf{X}$ generates a parametric class of univariate functions $(\kappa(\cdot, x) : x \in \mathsf{X})$. Just as well, this class of univariate functions comprise a bivariate function $\kappa : \mathsf{X} \times \mathsf{X} \to \mathbb{K}$. The property (5.1) is referred to as the *reproducing property* of the function κ. We emphasize to the reader that, under our conventions, inner products are conjugate linear in their *first* coordinate.

An especially interesting thing happens if we invoke the reproducing property (5.1) for the special choice $f = \kappa(\cdot, x')$, which gives

$$\kappa(x, x') = \langle \kappa(\cdot, x), \kappa(\cdot, x') \rangle_{\mathsf{H}}. \tag{5.2}$$

We see that the bivariate function κ tabulates the pairwise inner products of its univariate restrictions $\kappa(\cdot, x)$. By Hermiticity of the H-inner product, we conclude that

$$\kappa(x, x') = \langle \kappa(\cdot, x), \kappa(\cdot, x') \rangle_{\mathsf{H}} = \overline{\langle \kappa(\cdot, x'), \kappa(\cdot, x) \rangle_{\mathsf{H}}} = \overline{\kappa(x', x)}.$$

The function κ is Hermitian, i.e., conjugate symmetric. Additionally, given any (finite) set of points $\mathsf{D} \subseteq \mathsf{X}$, the function matrix $\kappa(\mathsf{D}, \mathsf{D})$ is psd (see Definition 3.1). To see thus, observe that for any $\boldsymbol{c} \in \mathbb{K}^{\mathsf{D}}$,

$$\boldsymbol{c}^* \kappa(\mathsf{D}, \mathsf{D}) \boldsymbol{c} = \sum_{x,x' \in \mathsf{D}} \overline{c_x} c_{x'} \, \kappa(x, x') = \sum_{x,x' \in \mathsf{D}} \overline{c_x} c_{x'} \, \langle \kappa(\cdot, x), \kappa(\cdot, x') \rangle_{\mathsf{H}}$$

$$= \left\langle \sum_{x \in \mathsf{D}} c_x \kappa(\cdot, x), \sum_{x' \in \mathsf{D}} c_{x'} \kappa(\cdot, x') \right\rangle \geq 0.$$

The inequality is positive definiteness of the H-inner product. We conclude that $\kappa(\mathsf{D}, \mathsf{D})$ is psd. Consequently, the function κ is a positive-definite kernel function (Definition 3.2). Thus, a more appropriate name for the function κ is the *reproducing kernel* of H.

Example 5.3 (Reproducing kernel for H^1)**.** We return to the case of $\mathsf{H}^1([0, 1])$, introduced in Example 5.2. Since $\mathsf{H}^1([0, 1])$ is an RKHS, it has a reproducing kernel $\kappa : [0, 1]^2 \to \mathbb{K}$ satisfying the property that

$$f(x) = \int_0^1 \left(\overline{\kappa(\cdot, a)} f(a) + \overline{\partial_a \kappa(\cdot, a)} f'(a) \right) \mathrm{d}a \tag{5.3}$$
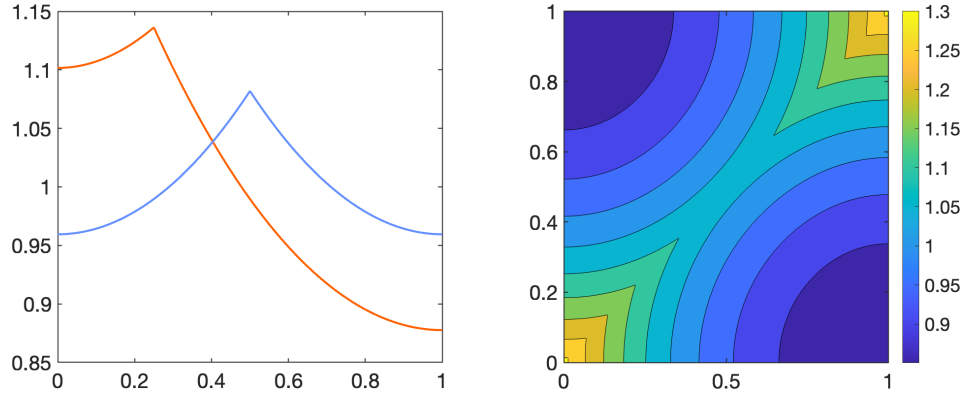
Figure 5.1: *Left:* Univariate restrictions $\kappa(\cdot, a)$ of Sobolev kernel (5.4) for $a = 0.25$ and $a = 0.5$. *Right:* Contour plot of Sobolev kernel (5.4).

for any $f \in \mathsf{H}^1([0, 1])$. We can recover the value of f at any point just by integrating against it and its derivative. Neat!

A formula for κ can be found by integrating the second term of (5.3) by parts and solving an ODE boundary value problem [Sch07, §2.11]. The resulting formula is

$$\kappa(x, x') = \frac{\cosh(\min(x, x')) \cosh(1 - \max(x, x'))}{\sinh(1)}. \tag{5.4}$$

The property (5.3) can be verified analytically or checked for example functions f and values x using a symbolic computing environment like Mathematica. An illustration of this kernel is provided in Fig. 5.1. ⋄

Spaces from kernels

So far, we have defined the notion of an RKHS and showed that every RKHS H has a reproducing kernel κ (that is, a positive-definite kernel function satisfying the reproducing property (5.1)). Often in practice, we would like to in the reverse direction: Begin with a set X, endow it with a positive-definite kernel function κ, and obtain an RKHS H for which κ is the kernel. This construction is possible in view of the Moore–Aronszajn theorem [Aro50, p. 3.44]:

Fact 5.4 (RKHSs from kernels)**.** *Every positive-definite kernel function κ on X gives rise to a unique H over which κ is the reproducing kernel.*

Proof sketch. Notice that the span of univariate restrictions of the kernel $\kappa(\cdot, x)$ form an inner product space, where the inner product is defined by the relation (5.2) and extended to linear combinations by sesquilinearity. This space may then be upgraded to a Hilbert space by taking the closure. □

This result speaks to how kernels are *used* in practice to learn from data. One begins with a dataset D belonging to some ambient space X, typically $\mathsf{X} = \mathbb{R}^d$. Then one selects a kernel for this data, typically from some standard functional form, such as the square-exponential kernel

$$\kappa_{\text{se}}(\boldsymbol{x}, \boldsymbol{x}' \mid \sigma) = \exp\left(-\frac{\|\|\boldsymbol{x} - \boldsymbol{x}'\|\|^2}{2\sigma^2}\right) \tag{5.5}$$

or the Laplace kernel

$$\kappa_{\text{Lap}}(\boldsymbol{x}, \boldsymbol{x}' \mid \sigma) = \exp\left(-\frac{\|\|\boldsymbol{x} - \boldsymbol{x}'\|\|}{\sigma}\right). \tag{5.6}$$

Here, $\|\|\cdot\|\|$ is a norm, typically the ℓ_1 or ℓ_2 norm. Hyperparameters for these kernel families, such as the bandwidth σ in (5.5) and (5.6), are picked either by ad-hoc techniques like the median heuristic [GJK18] or by systematic procedures like cross validation [LLJD+20]. One then uses this selected kernel κ to perform some data analysis task like interpolation (Section 5.2) or regression (Section 5.4). Fact 5.4 guarantees the existence of an RKHS H for which κ is the kernel, and the fine details of the Hilbert space H are typically not needed.

The kernel as a nonlinear inner product

There is another interpretation of kernel functions and RKHSs that can be useful. The kernel κ can be seen as defining a sort of nonlinear inner product on a general space X. More precisely, there exists a *feature map* $\phi : \mathsf{X} \to \mathsf{K}$ mapping X into a Hilbert space K over which κ coincides with the inner product

$$\kappa(x, x') = \langle \phi(x), \phi(x') \rangle_{\mathsf{K}}.$$

The space K is called the *feature space*. The feature map and the feature space are not unique, but a natural choice of feature space is given by the RKHS H itself. With this choice, the mapping $\phi : x \mapsto \kappa(\cdot, x)$ constitutes a feature map, in view of (5.2).

5.2 Kernel interpolation

Having established the RKHS formalism, let us see how it can be used to learn from data. Consider the task of learning a functional relation $g : \mathsf{X} \to \mathbb{K}$ from input–output pairs $(x_1, y_{x_1}), \ldots, (x_n, y_{x_n}) \in \mathsf{X} \times \mathbb{K}$. For convenience, we may package the inputs into a multiset $\mathsf{D} = \{x_1, \ldots, x_n\}$ and the outputs into a vector $\boldsymbol{y} = (y_x : x \in \mathsf{D}) \in \mathbb{K}^{\mathsf{D}}$. (In the case where D has repeated elements $x_i = x_j$ for $i \neq j$, we abuse notation and permit y_{x_i} and y_{x_j} to be different values.)

If we assume that the output values y_x are provided to us without noise, it makes sense to seek a model $g : \mathsf{X} \to \mathbb{K}$ that *interpolates* the data at the provided points:

$$g(x) = y_x \quad \text{for all } x \in \mathsf{D}.$$

The interpolation condition may be written more concisely as $g(\mathsf{D}) = \boldsymbol{y}$.

There are infinitely many functions $g \in \mathsf{H}$ interpolating the data. Of these, it is natural to select the interpolating function g of *minimum norm*. Indeed, if we think of the RKHS norm as a measure of the smoothness (as in Example 5.2) or complexity of a function, seeking the minimum-norm interpolant can be thought of as finding the interpolating function of minimum complexity (cf. the Occam's razor quote at the beginning of the section).

Finding the interpolating function of minimum norm is an optimization problem over the typically infinite dimensional space H:

$$g = \underset{\substack{g \in \mathsf{H} \\ g(\mathsf{D})=\boldsymbol{y}}}{\operatorname{argmin}} \|g\|_{\mathsf{H}}. \tag{5.7}$$

Remarkably, this optimization problem has a closed form solution:

Theorem 5.5 (Kernel interpolation: Solution formula)**.** *Let* D *be a finite set of points and let* $\boldsymbol{y} \in \mathbb{K}^{\mathsf{D}}$ *be output values. Assume* $\kappa(\mathsf{D}, \mathsf{D})$ *is nonsingular. Then the optimization problem* (5.7) *has the following unique solution*

$$g = \sum_{x \in \mathsf{D}} \kappa(\cdot, x)\beta_x \quad \text{where } \boldsymbol{\beta} = \kappa(\mathsf{D}, \mathsf{D})^{-1}\boldsymbol{y}.$$

Theorems of this type, which show that an infinite-dimensional optimization problem over an RKHS has a finitely parametrized solution consisting of a linear combination of kernel functions, are known as *representer theorems* [SS02, §4.2]. For completeness, and because the proof is beautiful and revealing, we shall provide a proof of this representer theorem in the rest of this section. Code for kernel interpolation is provided in Program 5.1.

Remark 5.6 (Interface for kernel methods). For the programs in this thesis, we use a common interface to implement kernel methods. Sets of n data points $\mathsf{D} \subseteq \mathbb{K}^d$ are collected as rows of an $n \times d$ matrix D. A univariate functions g can evaluated on a set of inputs $g(\mathsf{D})$ as $\mathsf{g(D)}$. Similarly, the evaluation $\kappa(\mathsf{D}, \mathsf{E})$ of a bivariate function κ can be evaluated as $\mathtt{kappa(D,E)}$. ◇

Program 5.1 `kernel_interp.m`. Code to compute the kernel interpolant through data y.

```
function [g,beta] = kernel_interp(y,kernel,D)
% Input:  Outputs y, kernel function kernel(x,x'), and inputs D
% Output: Kernel interpolant g and coefficients beta

beta = kernel(D,D) \ y;      % Get interpolation coefficients
g = @(X) kernel(X,D) * beta; % Define interpolant

end
```

Underdetermined systems of linear equations

To motivate the proof of Theorem 5.5, we shall begin with a review of the theory of underdetermined systems of linear equations in finite dimensions.

The linear least-squares problem

$$x = \underset{x \in \mathbb{K}^n}{\operatorname{argmin}} \|Bx - c\| \quad \text{for } B \in \mathbb{K}^{m \times n}, \, c \in \mathbb{K}^m$$

is well-known, as are the normal equations characterizing its solution:

$$B^* Bx = B^* c.$$

There is a parallel theory, usually not covered in introductory linear algebra classes, for the minimum norm solution to an underdetermined system of equations

$$x_\star = \underset{\substack{x \in \mathbb{K}^m \\ B^* x = c}}{\operatorname{argmin}} \|x\| \quad \text{for } B \in \mathbb{K}^{m \times n}, c \in \mathbb{K}^n. \tag{5.8}$$

The result is this:

Theorem 5.7 (Minimum-norm solution to an underdetermined system). *Assume* B *has full column rank. Then* $x = B(B^* B)^{-1} c = (B^\dagger)^* c$ *is the unique solution to* (5.8).

Proof. Decompose any solution x to $B^* x = c$ as an orthogonal sum $x = x_\star + x_\perp$ of a component $x_\star \in \operatorname{range}(B)$ and a component $x_\perp \in \operatorname{range}(B)^\perp$. The orthogonal complement of $\operatorname{range}(B)$ is the nullspace of B^*, so

$$c = B^* x = B^*(x_\star + x_\perp) = B^* x_\star + 0 = B^* x_\star.$$

Consequently, we see that $x_\star$ is also a solution of $B^* x = c$, and its norm is smaller:

$$\|x\|^2 = \|x_\star\|^2 + \|x_\perp\|^2 \geq \|x_\star\|^2. \tag{5.9}$$

Equality holds in (5.9) if and only if $x_\perp = 0$. Since $x_\star \in \text{range}(B)$, we may write $x_\star = By_\star$, so that

$$c = B^* x_\star = (B^* B) y_\star.$$

Since B is full-rank, the matrix $B^* B$ is invertible, so $y_\star = (B^* B)^{-1} c$. We conclude that $x_\star = B(B^* B)^{-1} c$ is the unique minimal-norm solution to $B^* x = c$. $\quad\square$

Underdetermined systems of linear equations in a Hilbert space

With the appropriate apparatus, the derivation for the minimum norm solution of a finite-dimensional system of linear equations translates effortlessly to infinite dimensions. We make the following definition [Ste98; TT15]:

Definition 5.8 (Quasimatrix)**.** Let K be a Hilbert space. An $\mathsf{K} \times n$ *quasimatrix F is a collection $F = (f_i : 1 \le i \le n)$ of elements $f_i \in \mathsf{K}$. For a matrix $H \in \mathbb{K}^{n \times p}$, the product FH is

$$FH = \left(\sum_{i=1}^{n} f_i h_{ij} : 1 \le j \le p \right).$$

The matrix–vector product Fh is defined analogously. For an $\mathsf{K} \times m$ quasimatrix G, the product $G^* F$ is

$$G^* F = \left(\langle g_i, f_j \rangle_\mathsf{K} : 1 \le i \le m, 1 \le j \le n \right) \in \mathbb{K}^{m \times n}.$$

Similarly, $G^* f = (\langle g_i, f \rangle : 1 \le i \le m)$ for $f \in \mathsf{K}$.

With this definition, Theorem 5.7 holds for underdetermined systems of equations over a Hilbert space with the same proof.

Theorem 5.9 (Minimum-norm solution to an underdetermined system over a Hilbert space)**.** *Let B be an $\mathsf{K} \times n$ quasimatrix and let $c \in \mathbb{K}^n$. Assume that B has linearly dependent "columns" $(b_i : 1 \le i \le n)$ or, equivalently, that $B^* B$ has full rank. Then the system of linear systems $B^* x = c$ has a unique solution $x \in \mathsf{K}$ of minimum norm. This solution is $x = B(B^* B)^{-1} c$.*

This result immediately implies the representer theorem for kernel interpolation (Theorem 5.5). To see this, introduce the quasimatrix

$$\kappa(\cdot, \mathsf{D}) := (\kappa(\cdot, x) : x \in \mathsf{D}) \in \mathbb{K}^{\mathsf{H} \times \mathsf{D}}$$

and its adjoint

$$\kappa(\mathsf{D}, \cdot) := \kappa(\cdot, \mathsf{D})^*. \tag{5.10}$$

Proof of Theorem 5.5. By the reproducing property and (5.10),

$$g(\mathsf{D}) = (g(x) : x \in \mathsf{D}) = (\langle \kappa(\cdot, x), g \rangle_{\mathsf{H}} : x \in \mathsf{D}) = \kappa(\mathsf{D}, \cdot)g.$$

Consequently, the minimum-norm interpolation problem (5.7) is equivalent to finding the minimum norm solution of the linear system $\kappa(\mathsf{D}, \cdot)g = \boldsymbol{y}$. By Theorem 5.9, the solution is

$$g = \kappa(\cdot, \mathsf{D})[\kappa(\mathsf{D}, \cdot)\kappa(\cdot, \mathsf{D})]^{-1}\boldsymbol{y}.$$

To complete the proof, observe that

$$
\begin{aligned}
\kappa(\mathsf{D}, \cdot)\kappa(\cdot, \mathsf{D}) &= (\langle \kappa(x, \cdot), \kappa(x', \cdot) \rangle_{\mathsf{H}} : x, x' \in \mathsf{D}) \\
&= (\kappa(x, x') : x, x' \in \mathsf{D}) = \kappa(\mathsf{D}, \mathsf{D}).
\end{aligned}
\tag{5.11}
$$

We conclude that $g = \kappa(\cdot, \mathsf{D})\kappa(\mathsf{D}, \mathsf{D})^{-1}\boldsymbol{y}$, as desired. $\square$

Remark 5.10 (Rank-deficient kernel matrix). If the kernel matrix κ is rank-deficient, then we have the following alternative version of Theorem 5.5:

Theorem 5.11 (Kernel interpolation: General case for solution formula). *Let* D *be a finite set of points and let* $\boldsymbol{y} \in \mathbb{K}^{\mathsf{D}}$ *be output values. Then there is a unique minimum norm solution* $g \in \mathsf{H}$ *to the least-squares problem*

$$\min_{g \in \mathsf{H}} \|\boldsymbol{y} - g(\mathsf{D})\|,$$

and it is given by

$$g = \kappa(\cdot, \mathsf{D})\kappa(\mathsf{D}, \mathsf{D})^{\dagger}\boldsymbol{y}.
\tag{5.12}$$

Going forward, we shall refer to the function g furnished by this theorem as the kernel interpolant, even when $\kappa(\mathsf{D}, \mathsf{D})$ is rank-deficient. In the rank-deficient case, the function g will not interpolate the data unless $\boldsymbol{y} \in \text{range}(\kappa(\mathsf{D}, \mathsf{D}))$. $\diamond$

Kernel interpolation, Nyström approximation, and error bounds

Kernel interpolation may be applied to any output data $\boldsymbol{y} \in \mathbb{K}^{\mathsf{D}}$, but something special happens when the data $\boldsymbol{y} = f(\mathsf{D})$ are outputs of a function $f \in \mathsf{H}$. In this case, the kernel interpolant, now denoted $\widehat{f_{\mathsf{D}}}$, takes the form

$$\widehat{f_{\mathsf{D}}} = \kappa(\cdot, \mathsf{D})\kappa(\mathsf{D}, \mathsf{D})^{\dagger}f(\mathsf{D}) = \kappa(\cdot, \mathsf{D})\kappa(\mathsf{D}, \mathsf{D})^{\dagger}\kappa(\mathsf{D}, \cdot)f.
\tag{5.13}$$

We have used the version (5.12) of the conclusion of Theorem 5.5. Observe the formal similarity of the right-hand side of (5.13) to the column Nyström approximation

(3.5) of a psd matrix. Drawing out this formal connection between kernel interpolation and Nyström approximation will give us a powerful theoretical framework. We will develop and use this theory here to prove error bounds, and this theory will play a major role when we extend RPCHOLESKY to infinite dimensions in Chapter 7.

Definition 5.12 (Nyström approximation of a kernel function). Let κ be a positive-definite kernel function on X and let $\mathsf{D} \subseteq \mathsf{X}$ be a finite subset. The *Nyström approximation* to κ induced by D is

$$\widehat{\kappa}_{\mathsf{D}}(x, x') \coloneqq \kappa(x, \mathsf{D})\kappa(\mathsf{D}, \mathsf{D})^{\dagger}\kappa(\mathsf{D}, x').$$

The *residual kernel* is $\kappa_{\mathsf{D}} \coloneqq \kappa - \widehat{\kappa}_{\mathsf{D}}$.

Just as a column Nyström approximation to a psd matrix and its residual are psd matrices, the Nyström approximation and its residual are both positive-definite kernels. The reader is reminded that a "positive-definite" kernel function is more analogous to a positive *semi*definite matrix.

Proposition 5.13 (Nyström approximation of kernels). *Let κ be a positive-definite kernel function on X, and let $\mathsf{S} \subseteq \mathsf{X}$ be subset. The Nyström approximate kernel $\widehat{\kappa}_{\mathsf{S}}$ and its residual $\kappa_{\mathsf{S}} = \kappa - \widehat{\kappa}_{\mathsf{S}}$ are both positive-definite kernels on X.*

Proof. Let $\mathsf{E} \subseteq \mathsf{X}$ and form the matrix $A \coloneqq \kappa(\mathsf{D} \cup \mathsf{E}, \mathsf{D} \cup \mathsf{E}) \in \mathbb{K}^{(\mathsf{D} \cup \mathsf{E}) \times (\mathsf{D} \cup \mathsf{E})}$. Introduce the matrix Nyström approximation $\widehat{A} \coloneqq A\langle\mathsf{S}\rangle$. One readily sees that

$$\widehat{\kappa_{\mathsf{D}}}(\mathsf{E}, \mathsf{E}) = \widehat{A}(\mathsf{E}, \mathsf{E}) \quad \text{and} \quad \kappa_{\mathsf{D}}(\mathsf{E}, \mathsf{E}) = (A - \widehat{A})(\mathsf{E}, \mathsf{E}).$$

By Proposition 2.9, these matrices are both psd. We conclude that $\widehat{\kappa}_{\mathsf{D}}$ and κ_{D} are positive-definite kernels. $\square$

The following result gives a natural representation for the kernel interpolant.

Lemma 5.14 (Reproducing representation for kernel interpolant). *Let H be an RKHS on X, $\mathsf{D} \subseteq \mathsf{X}$ be a finite set of points, and $f \in \mathsf{H}$. The kernel interpolant $\widehat{f}_{\mathsf{D}}$ through f at D admits the reproducing representation*

$$\widehat{f}_{\mathsf{S}}(x) = \langle\widehat{\kappa}_{\mathsf{S}}(\cdot, x), f\rangle_{\mathsf{H}} \quad \text{for each } x \in \mathsf{X}.$$

Proof. Using quasimatrices, this result is easy. We compute

$$\langle\widehat{\kappa}_{\mathsf{S}}(\cdot, x), f\rangle = (\kappa(\cdot, \mathsf{D})\kappa(\mathsf{D}, \mathsf{D})^{\dagger}\kappa(\mathsf{D}, x))^{*}f = \kappa(x, \mathsf{D})\kappa(\mathsf{D}, \mathsf{D})^{\dagger}\kappa(\mathsf{D}, \cdot)f.$$

By (5.13), this expression equals $\widehat{f}_{\mathsf{D}}(x)$. $\square$

We are tantalizingly close to proving error bounds for kernel interpolation. We need one final result.

Lemma 5.15 (Self inner product of residual and Nyström kernels)**.** *With the setting of Lemma 5.14, we have*

$$\langle \widehat{\kappa}_{\mathsf{D}}(\cdot, x), \widehat{\kappa}_{\mathsf{D}}(\cdot, x) \rangle_{\mathsf{H}} = \widehat{\kappa}_{\mathsf{D}}(x, x) \quad and \quad \langle \kappa_{\mathsf{D}}(\cdot, x), \kappa_{\mathsf{D}}(\cdot, x) \rangle_{\mathsf{H}} = \kappa_{\mathsf{D}}(x, x).$$

Proof. To prove the first identity, we employ the quasimatrix formalism:

$$
\begin{aligned}
\langle \widehat{\kappa}_{\mathsf{D}}(\cdot, x), \widehat{\kappa}_{\mathsf{D}}(\cdot, x) \rangle_{\mathsf{H}} &= (\kappa(\cdot, \mathsf{D})\kappa(\mathsf{D}, \mathsf{D})^{\dagger}\kappa(\mathsf{D}, x))^{*}(\kappa(\cdot, \mathsf{D})\kappa(\mathsf{D}, \mathsf{D})^{\dagger}\kappa(\mathsf{D}, x)) \\
&= \kappa(x, \mathsf{D})\kappa(\mathsf{D}, \mathsf{D})^{\dagger}\kappa(\mathsf{D}, \cdot)\kappa(\cdot, \mathsf{D})\kappa(\mathsf{D}, \mathsf{D})^{\dagger}\kappa(\mathsf{D}, x) \\
&= \kappa(x, \mathsf{D})\kappa(\mathsf{D}, \mathsf{D})^{\dagger}\kappa(\mathsf{D}, \mathsf{D})\kappa(\mathsf{D}, \mathsf{D})^{\dagger}\kappa(\mathsf{D}, x) \\
&= \kappa(x, \mathsf{D})\kappa(\mathsf{D}, \mathsf{D})^{\dagger}\kappa(\mathsf{D}, x) = \widehat{\kappa}_{\mathsf{D}}(x, x).
\end{aligned}
$$

The third line is (5.11), and the fourth line is the identity $\boldsymbol{M}^{\dagger}\boldsymbol{M}\boldsymbol{M}^{\dagger} = \boldsymbol{M}^{\dagger}$ for the pseudoinverse. The second identity follows from the first:

$$
\begin{aligned}
\langle \kappa_{\mathsf{D}}(\cdot, x), \kappa_{\mathsf{D}}(\cdot, x) \rangle_{\mathsf{H}} &= \langle \kappa(\cdot, x) - \widehat{\kappa}_{\mathsf{D}}(\cdot, x), \kappa(\cdot, x) - \widehat{\kappa}_{\mathsf{D}}(\cdot, x) \rangle_{\mathsf{H}} \\
&= \kappa(x, x) - 2\widehat{\kappa}_{\mathsf{D}}(x, x) + \langle \widehat{\kappa}_{\mathsf{D}}(\cdot, x), \widehat{\kappa}_{\mathsf{D}}(\cdot, x) \rangle_{\mathsf{H}} = \kappa_{\mathsf{D}}(x, x).
\end{aligned}
$$

The final equality is $\langle \widehat{\kappa}_{\mathsf{D}}(\cdot, x), \widehat{\kappa}_{\mathsf{D}}(\cdot, x) \rangle_{\mathsf{H}} = \widehat{\kappa}_{\mathsf{D}}(x, x)$ and the definition of κ_{D}. $\qquad\square$

With Lemmas 5.14 and 5.15 in hand, error bounds for kernel interpolation follow effortlessly.

Theorem 5.16 (Kernel interpolation: Pointwise error)**.** *Instate the setting and assumptions of Lemma 5.14. Then the kernel interpolant satisfies the error bound*

$$|f(x) - \widehat{f}_{\mathsf{D}}(x)|^{2} \le \kappa_{\mathsf{D}}(x, x) \cdot \|f\|_{\mathsf{H}}^{2}. \tag{5.14}$$

That is, the squared error at x is bounded in terms of the diagonal entry of the residual kernel κ_{D} at x. At each x, the bound is attained.

Proof. By Lemma 5.14, the error at x is

$$f(x) - \widehat{f}_{\mathsf{D}}(x) = \langle f, \kappa_{\mathsf{D}}(\cdot, x) \rangle_{\mathsf{H}}.$$

Take absolute values, and bound via Cauchy–Schwarz:

$$|f(x) - \widehat{f}_{\mathsf{D}}(x)|^{2} \le \langle \kappa_{\mathsf{D}}(\cdot, x), \kappa_{\mathsf{D}}(\cdot, x) \rangle_{\mathsf{H}} \cdot \|f\|_{\mathsf{H}}^{2} = \kappa_{\mathsf{D}}(x, x) \cdot \|f\|_{\mathsf{H}}^{2}.$$

The final equality is Lemma 5.15. The bound is attained by setting $f = \kappa_{\mathsf{D}}(\cdot, x)$. $\qquad\square$

5.3 Gaussian processes

We now discuss the theory of Gaussian processes and their applications to learning from data, which beautifully parallels the theory of RKHSs.

Definition 5.17 (Gaussian process). Let X be a set. A *Gaussian process* $g \sim \mathrm{GP}(m, \kappa)$ on X with *mean function* $m : X \to \mathbb{K}$ and *covariance function* $\kappa : X \times X \to \mathbb{K}$ is a random function g such that, for every finite subset $D \subseteq X$, the function values $g(D)$ obey a Gaussian distribution

$$g(D) \sim \mathrm{NORMAL}_{\mathbb{K}}(m(D), \kappa(D, D)).$$

If m is identically zero, we say that g is *centered* and write $g \sim \mathrm{GP}(\kappa)$.

Naturally, the covariance function κ is required to have the property that $\kappa(D, D)$ is psd for every D; that is, κ must be a positive-definite kernel. The term Gaussian process is frequently abbreviated "GP".

Remark 5.18 (Existence of GPs). The existence of a GP with any specified mean function and any positive-definite kernel as covariance function is ensured by the Kolmogorov extension theorem [Kle13, Thm. 14.36]. Unfortunately, this basic existence result is not sufficient to ensure that statements like "$g \sim \mathrm{GP}(m, \kappa)$ is continuous" can be assigned definite probabilities (i.e., the event "g is continuous" may not be measurable). Fortunately, in most cases of practical significance, there are results that are powerful enough both to ensure existence of a GP and that global properties like continuity and differentiability have definite probabilities. We will not discuss these more nuanced issues of GP theory in this thesis. ◇

We can also use Gaussian processes to design algorithms for learning from data. The simplest method is *Gaussian process interpolation*. Suppose we are given a (finite) data set $D \subseteq X$ and corresponding labels $y \in \mathbb{K}^D$, and we want to learn a functional relationship $g : D \to \mathbb{K}$. We adopt a Bayesian perspective. Begin with a *prior* that the "true" functional relationship $p : D \to \mathbb{K}$ is a draw from a Gaussian process, $p \sim \mathrm{GP}(\kappa)$; we use a centered prior for simplicity. We then model the data y as measurements of this function $y = p(D)$. We assume, for now, that the measurements are obtained without noise. Under this model, the *conditional distribution* of p is given by the following result.

Theorem 5.19 (Conditioning a Gaussian process). *Let $p \sim \mathrm{GP}(\kappa)$ be a GP on a base space X, let $D \subseteq X$ be a finite subset, and assume $\kappa(D, D)$ is nonsingular. Then*

$$p \mid \{p(D) = y\} \sim \mathrm{GP}(g, \kappa_D) \quad \text{for } g = \kappa(\cdot, D)\kappa(D, D)^{-1}y. \tag{5.15}$$
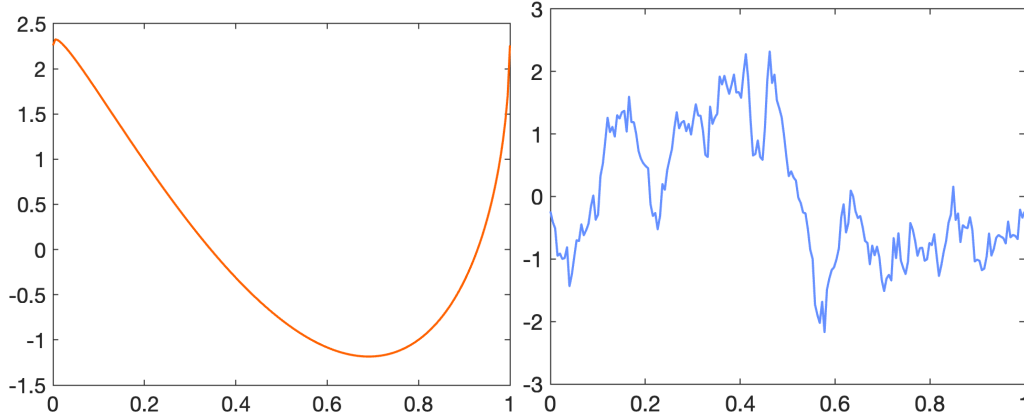
Figure 5.2: RKHS function (*left*) and single draw of a Gaussian process (*right*) for the same positive-definite kernel κ. The draw from the Gaussian process is observed to be much "rougher" than the RKHS function.

In particular, the mean function is the kernel interpolant (5.12) and the covariance function is the residual kernel (Definition 5.12).

Equation (5.15) characterizes the *posterior* distribution of the Gaussian process conditional on observing the data, $p(\mathsf{D}) = \boldsymbol{y}$. The mean $g : \mathsf{X} \to \mathbb{K}$ of the posterior provides a model of the functional relationship $\mathsf{X} \to \mathbb{K}$ that interpolates the data (x, y_x), and the covariance function κ_D captures the remaining uncertainty. In particular, the posterior variance is

$$\mathrm{Var}(p(x) \mid p(\mathsf{D}) = \boldsymbol{y}) = \kappa_\mathsf{D}(x, x). \tag{5.16}$$

Observe that this posterior variance agrees with the pointwise error bound (5.14) when $\|f\|_\mathsf{H} = 1$. We see that the diagonal entries of the residual kernel capture the uncertainty in both the kernel interpolation and Gaussian process interpolation settings. A notable feature of GPs is that the posterior variance (5.14) depends only on the location of the input data D, not the values of the output data $\boldsymbol{y}$.

The proof of Theorem 5.19 is standard, so we omit it; see [Tro23, Ch. 21] for an introduction to conditioning results for Gaussian random variables.

Remark 5.20 (Gaussian processes don't lie in the RKHS). Under fairly general conditions, draws $g \sim \mathrm{GP}(\kappa)$ *do not* belong to the associated RKHS H for which κ is the kernel (with 100% probability)! A visual illustration is provided in Fig. 5.2, which shows a function f from the *periodic* Sobolev space $\mathsf{H}^1_{\mathrm{per}}([0, 1])$ and a draw $g \sim \mathrm{GP}(\kappa)$ from a GP whose covariance function κ is the kernel for $\mathsf{H}^1_{\mathrm{per}}([0, 1])$. The GP realization $g \notin \mathsf{H}^1_{\mathrm{per}}([0, 1])$ is observed to be much rougher than the RKHS function $f \in \mathsf{H}^1_{\mathrm{per}}([0, 1])$. $\diamond$

5.4 Kernel ridge regression and Gaussian process regression

Gaussian process interpolation and kernel interpolation are two names and two interpretations for the same methodology. As the names suggest, these methods interpolate the data. Conventional wisdom from early machine learning practice and statistical learning theory suggest interpolation can lead to *overfitting*, yielding a model that fits the data but fails to generalize [Bis06, §§1.1, 3.2, & 5.5]. Recent machine learning practice has challenged this conventional wisdom, and many theoretical explanations for this phenomenon of *benign overfitting* have been proposed (e.g., [BLLT20; LR20; Bel21; CCBG22]).

Still, we may still have reasons to want to *regularize* kernel or GP fitting methods. First, for problems in lower dimensions, overfitting with kernel and Gaussian process methods may be a serious issue. Second, our data may be provided to us with noise, and including regularization at the level of the noise may help mitigate the effects of noise. Third, regularizing the problem makes it better conditioned and easier to solve with preconditioned iterative methods; see Section 6.1. Finally, even if one wants to interpolate the data, the kernel matrix $\kappa(\mathsf{D}, \mathsf{D})$ can sometimes be rank-deficient, at least up to the resolution of floating-point errors. For such a rank-deficient problems, adding even a small amount of regularization (say, at the level of the machine precision) may be necessary to obtain meaningful results.

Just as before, one can develop regularized data fitting methods in either the RKHS or GP formalisms. The resulting method will be called *kernel ridge regression* (KRR) or *Gaussian process regression* (GPR). As a change of pace, we will begin with the GP approach and then develop the corresponding RKHS perspective.

Gaussian process regression

To incorporate regularization into a GP data-fitting pipeline, we assume the following model. We begin from the same assumption the true underlying relationship is drawn from a *prior distribution*

$$p \sim \mathrm{GP}(\kappa) \tag{5.17a}$$

We now assume that the data $\boldsymbol{y} \in \mathbb{K}^{\mathsf{D}}$ is provided to us corrupted by noise:

$$\boldsymbol{y} = p(\mathsf{D}) + \boldsymbol{\varepsilon} \quad \text{where } \boldsymbol{\varepsilon} \sim \mathrm{NORMAL}_{\mathbb{K}}(0, \lambda \, \mathbf{I}). \tag{5.17b}$$

The parameter $\lambda \geq 0$ sets the variance of the noise $\boldsymbol{\varepsilon} \in \mathbb{K}^{\mathsf{D}}$, which is assumed to have iid entries $(\varepsilon_x : x \in \mathsf{D})$, independent of the GP g. Observe that $\lambda = 0$ recovers the data model for GP interpolation. The posterior of p given the data is as follows:

Theorem 5.21 (Gaussian process regression). *Let $p \sim \mathrm{GP}(\kappa)$ be a GP on a base space* X*, let* $\mathsf{D} \subseteq \mathsf{X}$ *be a finite subset, and assume* $\lambda > 0$*. Under the noise model* (5.17b)*, the posterior is*

$$p \mid \boldsymbol{y} \sim \mathrm{GP}(g, \kappa_\mathsf{D}^\lambda),$$

where

$$g = \kappa(\cdot, \mathsf{D})[\kappa(\mathsf{D}, \mathsf{D}) + \lambda \mathbf{I}]^{-1}\boldsymbol{y}$$

and

$$\kappa_\mathsf{D}^\lambda(x, x') = \kappa(x, x') - \kappa(x, \mathsf{D})[\kappa(\mathsf{D}, \mathsf{D}) + \lambda \mathbf{I}]^{-1}\kappa(\mathsf{D}, x') \quad \textit{for } x, x' \in \mathsf{X}.$$

Proof. We can employ a formal device to reduce the analysis of GPR to the result Theorem 5.19 we already know for Gaussian process interpolation. Begin by introducing a copy D' of the set D and defining $\mathsf{X}' := \mathsf{X} \cup \mathsf{D}'$. Now, package the data $(p, \boldsymbol{y})$ into a Gaussian process p' on X' by setting $p' = p$ on X and $p' = \boldsymbol{y}$ on D'. This GP is centered and its covariance function κ' is characterized by the formula

$$\kappa'(\mathsf{D}' \cup \mathsf{E}, \mathsf{D}' \cup \mathsf{E}) = \begin{bmatrix} \kappa(\mathsf{D}, \mathsf{D}) + \lambda\mathbf{I} & \kappa(\mathsf{D}, \mathsf{E}) \\ \kappa(\mathsf{E}, \mathsf{D}) & \kappa(\mathsf{E}, \mathsf{E}) \end{bmatrix} \quad \text{for } \mathsf{E} \subseteq \mathsf{X} \text{ finite.} \tag{5.18}$$

Let us justify this statement block-by-block. The $(1, 1)$-block is the covariance of the output data $\boldsymbol{y}$, and it equals $\kappa(\mathsf{D}, \mathsf{D}) + \lambda\mathbf{I}$ by (5.17b). The $(1, 2)$- and $(2, 1)$-blocks are the cross-covariances between $\boldsymbol{y}$ and $p(\mathsf{E})$, which equal the cross-covariances between $p(\mathsf{D})$ and $p(\mathsf{E})$ since the noise $\boldsymbol{\varepsilon}$ is independent of p. The $(2, 2)$-block is the covariance of $p(\mathsf{E})$, which is $\kappa(\mathsf{E}, \mathsf{E})$ by definition of the covariance function.

The Gaussian process regression posterior conditions p on the output values $\boldsymbol{y}$. Since p and p' agree on X, conditioning p on $\boldsymbol{y}$ is equivalent to conditioning p' on the event $p'(\mathsf{D}') = \boldsymbol{y}$, i.e., interpolating the values $\boldsymbol{y}$ on the copied space D'. In this sense, the Gaussian process *regression* prior on D is equivalent to the Gaussian process *interpolation* prior on D'. Therefore, by Theorem 5.19, the posterior for Gaussian process regression is

$$p(\mathsf{E}) \mid \boldsymbol{y} \sim \mathrm{GP}(\kappa'(\mathsf{E}, \mathsf{D}')\kappa'(\mathsf{D}', \mathsf{D}')^{-1}\boldsymbol{y}, \kappa_{\mathsf{D}'}'(\mathsf{E}, \mathsf{E})) \quad \text{for } \mathsf{E} \subseteq \mathsf{X} \text{ finite.}$$

Invoking the formula (5.18) for κ' completes the proof. $\qquad\square$

For low-dimensional fitting tasks with noisy data, choosing the correct regularization λ is critical to the success of GPR. An illustration is provided in Fig. 5.3, which applies GPR to fitting the ages of Nobel laureates in Physics versus the year of

$$\lambda = 10^{-13} \qquad\qquad \lambda = 1 \qquad\qquad \lambda = 100$$
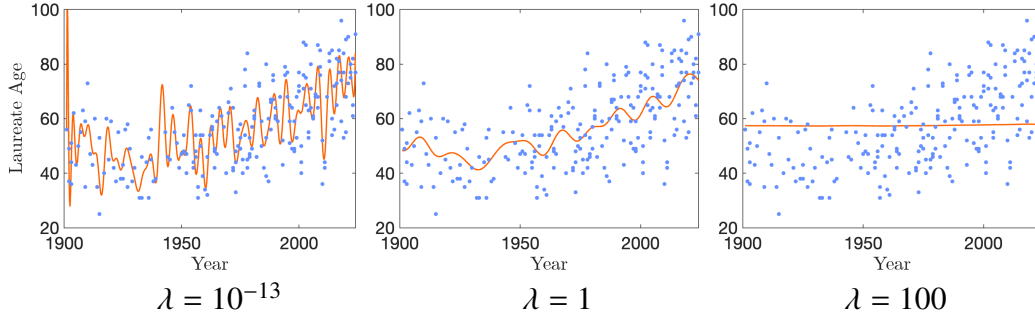
Figure 5.3: Fitting of Nobel laureate ages by prize year using GPR for three values of the regularization $\lambda = 10^{-13}$ (*left*), $\lambda = 1$ (*middle*), and $\lambda = 100$ (*right*); further details are in the text.

their prize. To apply GPR, we first center the ages in the dataset to have mean zero and employ a square-exponential kernel (5.5) with bandwidth $\sigma = 5$. The left panel uses almost no regularization, and the resulting model is wildly oscillatory. The right panel uses too much regularization, and the resulting model is nearly constant and thus uninformative. The middle panel sits between these extremes and produces an interesting model that captures trends in the data. In particular, this model demonstrates that the average age of Nobel laureates was relatively stable in the twentieth century and has since been rising. Even this goldilocks models is perhaps more "wiggly" then one would like.

Remark 5.22 (Parameter selection). The Gaussian process framework also provides a natural prescription for picking hyperparameters such as the regularization parameter λ and the bandwidth σ for a covariance function such as (5.5) or (5.6). In general, assume the kernel $\kappa = \kappa_\theta$ is parametrized by hyperparameters θ. *Under the data generation model* (5.17), the likelihood (probability density) of observing the data y is

$$L(y;\theta,\lambda) = \frac{1}{(2\pi)^{d/2}\det(\kappa_\theta(\mathsf{D},\mathsf{D})+\lambda\mathbf{I})^{1/2}}\exp\left(-\frac{y^*[\kappa_\theta(\mathsf{D},\mathsf{D})+\lambda\mathbf{I}]^{-1}y}{2}\right).$$

Up to an affine transformation, the log-likelihood is

$$\ell(y;\theta,\lambda) = -y^*[\kappa_\theta(\mathsf{D},\mathsf{D})+\lambda\mathbf{I}]^{-1}y - \log\det[\kappa_\theta(\mathsf{D},\mathsf{D})+\lambda\mathbf{I}].$$

One can then pick the hyperparameters θ and λ to maximize the log-likelihood by an iterative procedure like gradient descent. The computation of the log-determinant and its gradients can be a nontrivial problem, for which algorithms based on randomized trace estimation can be employed; see Part II. ◇

Kernel ridge regression

The GPR method can also be derived within the RKHS formalism. The kernel interpretation of GPR is referred to as *kernel ridge regression* (KRR).

To derive the KRR problem, we begin by formulating a least-squares regression problem over the RKHS H. Given data $\boldsymbol{y} \in \mathbb{K}^{\mathsf{D}}$, the least-squares fit

$$\min_{g \in \mathsf{H}} \|\boldsymbol{y} - g(\mathsf{D})\|^2.$$

has infinitely many solutions, all of which interpolate the data (at least as well as possible; see Theorem 5.11). To make the procedure more resilient to noise, we can reformulate this problem by adding a ridge penalty term:

$$\widehat{g} = \underset{g \in \mathsf{H}}{\operatorname{argmin}} \|\boldsymbol{y} - g(\mathsf{D})\|^2 + \lambda \|g\|_{\mathsf{H}}^2 \quad \text{for } \lambda > 0. \tag{5.19}$$

The ridge penalty $\lambda \|g\|_{\mathsf{H}}^2$ serves to penalize "roughness" or "complexity", as measured by the RKHS norm. The (unique) solution $\widehat{g}$ is the KRR regression model, which provides a model of the input output relation $\mathsf{X} \to \mathbb{K}$.

Theorem 5.23 (Kernel ridge regression: Solution formula)**.** *Let H be an RKHS and let $\lambda > 0$. The KRR problem* (5.19) *has a unique solution, which satisfies*

$$g = \kappa(\cdot, \mathsf{D})[\kappa(\mathsf{D}, \mathsf{D}) + \lambda \mathbf{I}]^{-1}\boldsymbol{y}.$$

Equivalently,

$$g = \sum_{x \in \mathsf{D}} \kappa(\cdot, x)\beta_x \quad \textit{where } \boldsymbol{\beta} = \underset{\boldsymbol{\beta} \in \mathbb{K}^{\mathsf{D}}}{\operatorname{argmin}} \|\kappa(\mathsf{D}, \mathsf{D})\boldsymbol{\beta} - \boldsymbol{y}\|^2 + \lambda \boldsymbol{\beta}^* \kappa(\mathsf{D}, \mathsf{D})\boldsymbol{\beta}.$$

Proof sketch. First, argue that the solution g lies in the column span of the quasi-matrix $\kappa(\cdot, \mathsf{D})$, arguing similarly to Theorem 5.7. Then, use the ansatz $g = \kappa(\cdot, \mathsf{D})\boldsymbol{\beta}$ and solve for $\boldsymbol{\beta}$. $\qquad\square$

Code for kernel ridge regression appears in Program 5.2.

Program 5.2 `krr.m`. Kernel ridge regression for data fitting.

```matlab
function [fhat,beta] = krr(y,kernel,D,lamb)
% Input:  Data y, kernel function kernel(x,x'), data D, and
%         regularization lamb >= 0
% Output: Kernel interpolant fhat and coefficients beta

A = kernel(D,D) + lamb*eye(size(D,1)); % kappa(D,D) + lamb*I
beta = A \ y;                          % Get coefficients
fhat = @(X) kernel(X,D) * beta;        % Define regression function

end
```

Chapter 6

ACCELERATING KERNEL AND GAUSSIAN PROCESS METHODS BY SUBSET SELECTION AND COLUMN NYSTRÖM APPROXIMATION

> *Fortunately, there is a path forward. To implement kernel methods, we simply need to approximate the kernel matrix. . . Even a poor approximation of the kernel can suffice to achieve near-optimal performance, both in theory and in practice.*
>
> Per-Gunnar Martinsson and Joel A. Tropp, *Randomized numerical linear algebra: Foundations and algorithms* [MT20]

In last chapter, we saw how to use the theories of reproducing kernel Hilbert spaces (RKHSs) and Gaussian processes (GPs) to develop methods for learning from data. Throughout this discussion, we had little to say about the *computational cost* of these methods. When run on a dataset D of size $|\mathsf{D}| = n$, direct implementation of all the methods in the previous chapter require forming, storing, and factorizing the $n \times n$ kernel matrix $\kappa(\mathsf{D}, \mathsf{D})$. This incurs a heavy computational cost, sometimes called the *curse of kernelization* [WCV12]:

> **Curse of kernelization.** The cost of implementing kernel interpolation, kernel ridge regression, and many other kernel methods on n data points *using standard direct linear algebra methods* requires $O(n^2)$ storage and $O(n^3)$ operations.

In this chapter, we will see how RPCHOLESKY and other psd column subset selection methods can be used to accelerate kernel methods, resulting in faster algorithms.

We say up front that there is no free lunch here. RPCHOLESKY and other psd low-rank approximation are effective for kernel problems when the kernel matrix $\kappa(\mathsf{D}, \mathsf{D})$ is well-approximated by a low-rank matrix. Fortunately, many kernel matrices possess this property, so the approaches described in this chapter have wide—though not universal—applicability.

Sources. Section 6.1 is adapted from the paper

Mateo Díaz et al. *Robust, Randomized Preconditioning for Kernel Ridge Regression.* July 2024. arXiv: 2304.12465v5

Section 6.2 is adapted from the original RPCHOLESKY paper

Yifan Chen, Ethan N. Epperly, Joel A. Tropp, and Robert J. Webber. "Randomly Pivoted Cholesky: Practical Approximation of a Kernel Matrix with Few Entry Evaluations". In: *Communications on Pure and Applied Mathematics* 78.5 (2025), pp. 995–1041. DOI: 10.1002/cpa.22234.

The material in Section 6.3 is new.

Outline. Sections 6.1 to 6.3 present three approaches to accelerating KRR (equivalently, GPR) using RPCHOLESKY or other psd column subset selection algorithms; these sections are ordered from most expensive and most accurate to least expensive and most approximate. Section 6.1 discusses using RPCHOLESKY to precondition the KRR linear systems; this approach leads to full accuracy, but requires $O(n^2)$ operations even under favorable conditions. Section 6.2 discusses the restricted KRR problem, a cheaper approximate version of KRR with a reduced cost of $O(k^2 n)$, where k is a user-tunable subset size. Section 6.3 discusses using RPCHOLESKY for KRR in the setting of *active learning problems*.

6.1 Column Nyström preconditioning

Our first way of using psd low-rank approximation methods to accelerate kernel methods is via *preconditioning*. Let us focus on KRR. Per Theorem 5.23, the optimal coefficients $\boldsymbol{\beta} \in \mathbb{K}^n$ for KRR are the solution to a linear system

$$(A + \lambda \mathbf{I})\boldsymbol{\beta} = y \quad \text{for } A := \kappa(\mathsf{D}, \mathsf{D}). \tag{6.1}$$

When n is large (say, so large that one cannot fit the entire matrix A in memory at once), it is natural to use an iterative method like conjugate gradient [Saa03, §6.7] to solve (6.1). However, if the matrix $A + \lambda \mathbf{I}$ is ill-conditioned, the convergence of iterative methods will be slow. To improve convergence, we can form a column Nyström approximation $\widehat{A} \approx A$ and use this approximation to *precondition* the linear system, resulting in faster convergence.

Let $\widehat{A} = \boldsymbol{F} \boldsymbol{F}^*$ be a column Nyström approximation computed by any method; we recommend RPCHOLESKY (or, more precisely, a fast implementation of RPCHOLESKY, see Chapter 8) for use in practice. Define the Nyström preconditioner $\boldsymbol{P} := \widehat{A} + \lambda \mathbf{I}$. In practice, it is grossly inefficient to store $\boldsymbol{P}$ directly. Instead, we compute an

Program 6.1 `rpcholesky_precon.m`. RPCHOLESKY-preconditioned conjugate gradient for solving KRR problems. Subroutine `mypcg` is provided in Program F.5.

```
function [g,Betas] = rpcholesky_precon(y,kernel,D,lamb,k,iter)
% Input:  Outputs y, kernel function kernel(x,x'), inputs D,
%         regularization lamb, rank k, and number of iterations iter
% Output: Kernel interpolant g and sequence Beta of coefficients
%         produced by PCG, stacked columnwise

A = kernel(D,D);                    % Generate full kernel matrix
Acol = @(S) A(:,S);                 % Column generation subroutine
Asub = @(S) A(S,S);                 % Submatrix generation subroutine
d = diag(A);                        % Diagonal of kernel matrix
b = ceil(k/2);                      % Block size for accelerated RPC
F = acc_rpcholesky(Acol,Asub,d,k,b);% LRA by accelerated RPCholesky
[U,S,~] = svd(F,"econ");            % Economy-size SVD

% Define matrix-vector product, preconditioner
matvec = @(z) A*z + lamb*z;
pre = @(z) U*((S^2 + lamb*eye(k))\(U'*z)) + (z - U*(U'*z)) / lamb;

Betas = mypcg(matvec,pre,y,iter);   % Coefficients by PCG
g = @(X) kernel(X,D) * Betas(:,end);% Define interpolant

end
```

(economy-size) SVD $F = U\Sigma V^*$, from which we can apply the action of P and its inverse via the formulas

$$Pz = U(\Sigma^2(U^*z)) + \lambda z,$$
$$P^{-1}z = U((\Sigma^2 + \lambda I)^{-1}(U^*z)) + \lambda^{-1}(z - U(U^*z)). \tag{6.2}$$

To solve (6.1), we run preconditioned conjugate gradient (PCG) with preconditioner P. The matrix $A + \lambda I$ is applied via the formula $(A + \lambda I)z = Az + \lambda z$, and the inverse-preconditioner is applied via the formula (6.2). See Program 6.1 for an implementation. (Note that we use the faster *accelerated* version of RPCHOLESKY algorithm for our implementation; see Section 8.4.)

Computational cost. The computational cost of KRR with column Nyström preconditioning consists of generating the entries of the kernel matrix $A = \kappa(\mathsf{D}, \mathsf{D})$, forming the preconditioner P, and performing iterations with preconditioned conjugate gradient. We analyze the cost of two variants, a high-memory version where the kernel matrix A is formed once and stored, and a low-memory version where the kernel matrix is regenerated each PCG iteration. We denote by "niter" the number

Implementation	High-Memory	Low-Memory
Storage	$O(n^2)$	$O(kn)$
Runtime	$O((t_\kappa + \text{niter})n^2 + k^2n)$	$O(\text{niter} \cdot t_\kappa n^2 + k^2n)$

Table 6.1: Storage and runtime costs for low-memory and high-memory column Nyström preconditioned kernel ridge regression. Here, niter is the number of PCG iterations and $t_\kappa \geq 1$ the number of operations required for a single kernel function evaluation.

of PCG iterations and $t_\kappa \geq 1$ the number of operations required for a single kernel function evaluation.

First, suppose that we generate and store the whole kernel matrix once, requiring $O(n^2)$ memory and $O(t_\kappa n^2)$ time. Afterwards, computing and factorizing $F = U\Sigma V^*$ using most column Nyström methods (RPCHOLESKY, RLS sampling via the RRLS algorithm, uniform sampling, greedy selection) requires $O(k^2n)$ operations. Each PCG iteration consists of one matvec with $A + \lambda \mathbf{I}$, costing $O(n^2)$ operations, and one invocation of the primitive (6.2), costing $O(kn)$ operations. The total cost is thus $O((t_\kappa + \text{niter})n^2 + k^2n)$ operations.

Second, suppose we regenerate entries of kernel matrix on an as-needed basis. The storage costs are now dominated by storing the factor U needed for the inverse-preconditioner operation (6.2), requring $O(kn)$ memory. Forming the preconditioner requires $O(t_\kappa kn + k^2n)$ operations, and each PCG iteration requires $O(t_\kappa n^2 + kn)$ operations, since we must regenerate the matrix at each iteration. The total cost is thus $O(\text{niter} \cdot t_\kappa n^2 + k^2n)$ operations.

Table 6.1 compares both implementations. These implementations represent a classic time–space tradeoff. Regenerating the kernel matrix makes each PCG iteration significantly more expensive but also cuts the storage costs dramatically.

Analysis. How many PCG iterations are required for RPCHOLESKY-preconditioned KRR to converge? The following result provides a partial answer:

Theorem 6.1 (RPCHOLESKY preconditioning). *Fix $\lambda > 0$ and psd matrix $A \in \mathbb{K}^{n \times n}$. Introduce the* tail rank

$$d_{\text{tail}}(\lambda) := \min \left\{ r \geq 0 : \sum_{i=r+1}^{n} \lambda_i(A) \leq \lambda \right\},$$

execute RPCHOLESKY for $k \geq d_{\text{tail}}(\lambda)(1 + \log(\text{tr}(A)/\lambda))$ steps to produce low-rank approximation $\widehat{A}$, and instantiate the RPCHOLESKY preconditioner $P := \widehat{A} + \lambda \mathbf{I}$. With

90% probability, the preconditioned condition number is controlled as

$$\text{cond}(\boldsymbol{P}^{-1/2}(\boldsymbol{A} + \lambda \mathbf{I})\boldsymbol{P}^{-1/2}) \leq 30.$$

Consequently, PCG produces a solution $\boldsymbol{\beta}^{(\text{niter})}$ satisfying the guarantee

$$\|\boldsymbol{\beta}^{(\text{niter})} - \boldsymbol{\beta}\|_{A+\lambda\mathbf{I}} \leq \varepsilon \cdot \|\boldsymbol{\beta}\|_{A+\lambda\mathbf{I}}$$

after at most $\text{niter} \leq \lceil 6 \log(2/\varepsilon) \rceil$ *steps. Here, $\|z\|_M := (z^* \boldsymbol{M} z)^{1/2}$ denotes the norm associated to a positive definite matrix $\boldsymbol{M}$.*

A slight strengthening of this result and its proof appear in [DEFT+24, Thm. 2.2].

We conjecture that Theorem 6.1 holds with the tail dimension $d_{\text{tail}}(\lambda)$ replaced by the effective dimension $d_{\text{eff}}(\lambda)$, defined in Definition 3.16. To get a sense of the difference between $d_{\text{tail}}(\lambda)$ and $d_{\text{eff}}(\lambda)$, suppose the dimension n is very large, and consider a matrix with polynomially decaying spectrum $\lambda_i(\boldsymbol{A}) = i^{-p}$, where $p > 1$ is *fixed*. The tail rank and effective dimensions are $d_{\text{tail}}(\lambda) = \Theta(\lambda^{-1/(p-1)})$ and $d_{\text{eff}}(\lambda) = \Theta(\lambda^{-1/p})$. For a small power, say $p = 2$, these parameters are $d_{\text{tail}}(\lambda) = \Theta(\lambda^{-1})$ and $d_{\text{tail}}(\lambda) = \Theta(\lambda^{-1/2})$, leading to a dramatic difference when $\lambda \ll 1$. For large p (or high λ), the distinction between tail rank and effective dimension is less significant. Ultimately, the "missing link" that would allow us to replace $d_{\text{tail}}(\lambda)$ in Theorem 6.1 by $d_{\text{eff}}(\lambda)$ is better bounds for RPCHOLESKY in the spectral norm; see Section 11.1 for discussion on the types of bounds I conjecture.

Parameter selection. There are many approaches to choosing parameters for RP-CHOLESKY preconditioning. One approach, useful in memory-constrained settings, is to just choose the parameter k as large as memory will allow. A second rule of thumb, guided more by runtime considerations, is to set $k \sim n^{1/2}$, so that the $O(k^2 n)$ cost of forming the preconditioner is comparable to the $O(n^2)$ cost of a single conjugate gradient step. Since many CG steps are typically required, I would recommend values of $k \approx 3\sqrt{n}$ or $k \approx 10\sqrt{n}$. Last, one can use the residual trace to set the approximation rank, running RPCHOLESKY until the trace of the residual matrix falls below a tolerance. The parameter λ provides a natural guide for the scale of the tolerance.

Experiment. Here, we report an experiment performed by myself and coauthors in [DEFT+24, Fig. 6]. Here, we apply several column Nyström preconditioners to predict the highest occupied molecule orbital energy for $n = 5 \times 10^4$ data points
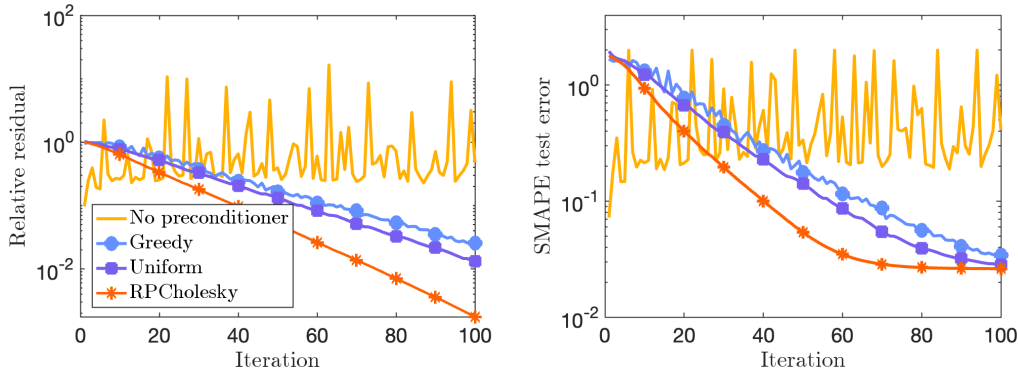
Figure 6.1: Relative residual (*left*) and SMAPE test error (*right*) for column Nyström-preconditioned conjugate gradient with greedy (blue circles), uniform (purple squares), RPCHOLESKY (orange asterisks), and no preconditioning (yellow).

from the QM9 dataset [RvBR12; RDRv14]. We set the approximation rank to be $k := 10^3$. For each choice of KRR coefficients $\boldsymbol{\beta}$, we report the relative residual

$$\|(\kappa(\mathsf{D}, \mathsf{D}) + \lambda \mathbf{I})\boldsymbol{\beta} - \boldsymbol{y}\| / \|\boldsymbol{y}\|$$

and the symmetric mean average percentage error

$$\mathrm{SMAPE}(\boldsymbol{y}^{(\mathrm{test})}, \widehat{\boldsymbol{y}}^{(\mathrm{test})}) = \frac{1}{m} \sum_{i=1}^{m} \frac{|y_i^{(\mathrm{test})} - \widehat{y}_i^{(\mathrm{test})}|}{(|y_i^{(\mathrm{test})}| + |\widehat{y}_i^{(\mathrm{test})}|)/2},$$

computed between the test data $\boldsymbol{y}^{(\mathrm{test})}$ and the predicted values $\widehat{\boldsymbol{y}}^{(\mathrm{test})} = \kappa(\mathsf{D}^{(\mathrm{test})}, \mathsf{D})\boldsymbol{\beta}$ for the test data.

Results are shown in Fig. 6.1. We see that, on this example, RPCHOLESKY outperforms the other two column Nyström preconditioning strategies, achieving the lowest possible test error in about 60 iterations versus 100 iterations for the other two methods. More than anything, this example illustrates that *some* preconditioner is absolutely necessary to solve this problem using an iterative method, as the method fails to converge at all without preconditioning.

Figure 6.1 shows just a single example of the success of Nyström preconditioning. See [DEFT+24, §2.1] for many more experiments, including performance plots which demonstrate that RPCHOLESKY preconditioning is generally the most effective column Nyström preconditioning method among available strategies on a testbed of examples.

6.2   Restricted kernel ridge regression

Even in the most optimistic setting, column Nyström preconditioning for KRR still requires work *quadratic* in the data size *n*. In this section, we will develop methods with a runtime that is, in principle, *linear* in size of the data. More precisely, we will describe *restricted kernel ridge regression*, an approximate form of KRR that requires at most $O(kn)$ storage and $O(k^2n)$ time, where $1 \leq k \leq n$ is a tunable parameter. Larger values of k typically leads to more accurate results, at the cost of being more expensive.

Description of restricted KRR method

We have largely described kernel and GP fitting algorithms in abstract terms, as minimum norm interpolants in a Hilbert space, solutions to infinite-dimensional regularized least-squares problems, or as conditional expectations of Gaussian processes. More prosaically, kernel interpolation is just interpolation of scattered data (x, y_x) by a linear combination of functions $\kappa(\cdot, x)$ for $x \in \mathsf{D}$:

$$g = \sum_{x \in \mathsf{D}} \kappa(\cdot, x)\beta_x. \tag{6.3a}$$

For KRR, we fit the data by a function of this form that minimizes a regularized least-squares loss. More precisely, following Theorem 5.23, $\boldsymbol{\beta}$ is chosen to satisfy

$$\boldsymbol{\beta} = \underset{\boldsymbol{\beta} \in \mathbb{K}^{\mathsf{D}}}{\operatorname{argmin}} \|\kappa(\mathsf{D}, \mathsf{D})\boldsymbol{\beta} - \boldsymbol{y}\|^2 + \lambda \boldsymbol{\beta}^* \kappa(\mathsf{D}, \mathsf{D})\boldsymbol{\beta}. \tag{6.3b}$$

To reduce computational cost and obtain a model $\widehat{g} : \mathsf{D} \to \mathbb{K}$ with fewer parameters, it is natural to consider a restricted version of the full optimization problem (6.3) where $\boldsymbol{\beta} \in \mathbb{K}^{\mathsf{D}}$ is only permitted to be nonzero in k selected positions $\mathsf{S} \subseteq \mathsf{D}$. Denoting $\widehat{\boldsymbol{\beta}} = \boldsymbol{\beta}(\mathsf{S})$, we have the following *restricted* version of the KRR problem:

$$\widehat{g} = \sum_{x \in \mathsf{S}} \kappa(\cdot, x)\widehat{\beta}_x \tag{6.4a}$$

with

$$\widehat{\boldsymbol{\beta}} = \underset{\boldsymbol{\beta} \in \mathbb{K}^{\mathsf{S}}}{\operatorname{argmin}} \|\kappa(\mathsf{D}, \mathsf{S})\widehat{\boldsymbol{\beta}} - \boldsymbol{y}\|^2 + \lambda \widehat{\boldsymbol{\beta}}^* \kappa(\mathsf{S}, \mathsf{S})\widehat{\boldsymbol{\beta}}. \tag{6.4b}$$

We call (6.4) the *restricted KRR method*. Its solution satisfies the *normal equations*

$$[\kappa(\mathsf{S}, \mathsf{D})\kappa(\mathsf{D}, \mathsf{S}) + \lambda\kappa(\mathsf{S}, \mathsf{S})]\widehat{\boldsymbol{\beta}} = \kappa(\mathsf{S}, \mathsf{D})\boldsymbol{y}. \tag{6.5}$$

The landmark set S is fixed during restricted KRR, though it can (and should!) be adaptively selected based on the data D using a procedure such as RPCHOLESKY. The

landmarks S can be chosen by a psd column selection algorithm like RPCHOLESKY. To facilitate comparison, we call (6.3) the *full-data KRR method*.

Observe that the restricted KRR problem (6.4b) is a regularized least-squares problem of dimension $n \times k$, where $k = |\mathsf{S}|$. Its solution defines a regression function $\widehat{g}$ that can evaluated at a point using only k kernel function evaluations; compare to the n kernel function evaluations required to evaluate the full KRR model (6.3a).

Remark 6.2 (History and terminology). Methods for simplifying a regression problem by using only a subset of basis functions date back at least to work on scattered data interpolation in the 1980s [Wah90, Ch. 7]. The modern literature on restricted KRR began in the GP community with the work of Smola and Bartlett [SB00], who coined the name *sparse Gaussian process regression* (SGPR). The method reemerged in the kernel literature in the work of Rudi, Camoriano, and Rosasco [RCR15] as the *Nyström method*.

I find the terms "sparse Gaussian process" and "Nyström method" both to be potentially misleading. In sparse Gaussian process regression, it is the *coefficient vector β* that is sparse, not the kernel matrix or the target solution $\widehat{g}$. Confusingly, methodologies where one designs a kernel with compact support [Wen04, Ch. 9] or zero out small entries in the kernel matrix, resulting in a sparse kernel matrix, are also common [FGN06]. As we have seen already and will continue to see in this thesis, there are several non-equivalent ways of using Nyström approximation to accelerate the solution of KRR problems, so the term "Nyström method" can also be ambiguous. For these reasons. I use the term "restricted KRR" for (6.4), which my co-authors and I introduced in [DEFT+24]. ◇

Restricted KRR as Nyström approximation of the kernel

So far, we have presented restricted KRR as an ad hoc solution to reduce computational costs. We can also give interpretations of this method in both the RKHS and GP formalisms. We look at the RKHS formalism first.

Recall that a subset $\mathsf{S} \subseteq \mathsf{D}$ induces a Nyström approximation

$$\kappa(\mathsf{D}, \mathsf{S})\kappa(\mathsf{S}, \mathsf{S})^{\dagger}\kappa(\mathsf{S}, \mathsf{D})$$

to the kernel matrix $\kappa(\mathsf{D}, \mathsf{D})$ *and* a Nyström approximation of the entire kernel function κ:

$$\widehat{\kappa}_{\mathsf{S}}(x, x') = \kappa(x, \mathsf{S})\kappa(\mathsf{S}, \mathsf{S})^{\dagger}\kappa(\mathsf{S}, x') \quad \text{for } x, x' \in \mathsf{X}.$$

This observation leads to an interpretation of restricted KRR, which was suggested to me by Yifan Chen:

Theorem 6.3 (Restricted KRR as Nyström approximation of the kernel). *Let $\mathsf{S} \subseteq \mathsf{D} \subseteq \mathsf{X}$ be nested finite subsets of a base space X, and let $\boldsymbol{y} \in \mathbb{K}^{\mathsf{D}}$ be data. Assume the regularization parameter $\lambda > 0$ is* <mark>positive</mark> *and the kernel matrix $\kappa(\mathsf{S}, \mathsf{S})$ is* <mark>invertible</mark>*. The restricted KRR function $\widehat{g}$ given by* (6.4a) *is the output of the full-data KRR method applied to the data $\boldsymbol{y} \in \mathbb{K}^{\mathsf{D}}$ with the Nyström approximate kernel:*

$$\widehat{g} = \widehat{\kappa}_{\mathsf{S}}(\cdot, \mathsf{D})[\widehat{\kappa}_{\mathsf{S}}(\mathsf{D}, \mathsf{D}) + \lambda \mathbf{I}]^{-1}\boldsymbol{y}. \tag{6.6}$$

To prove this result, we will employ the following lemma:

Lemma 6.4 (Nonsymmetric ridge regression identity). *Let $\boldsymbol{B}, \boldsymbol{C} \in \mathbb{K}^{n \times k}$ be matrices, and choose $\lambda \in \mathbb{K}$ to ensure that the matrix inverses below are well-defined. Then*

$$(\boldsymbol{B}^*\boldsymbol{C} + \lambda\mathbf{I})^{-1}\boldsymbol{B}^* = \boldsymbol{B}^*(\boldsymbol{C}\boldsymbol{B}^* + \lambda\mathbf{I})^{-1}.$$

The case $\boldsymbol{B} = \boldsymbol{C}$, yields the identity $(\boldsymbol{B}^*\boldsymbol{B} + \lambda\mathbf{I})^{-1}\boldsymbol{B}^* = \boldsymbol{B}^*(\boldsymbol{B}\boldsymbol{B}^* + \lambda\mathbf{I})$, useful in the analysis of ridge-regularized linear regression, which may be called the (symmetric) ridge regression identity. The formula is easily checked by multiplying to clear the inverses and checking that the left- and right-hand sides agree. It may also be derived using the Sherman–Morrison–Woodbury formula.

Proof of Theorem 6.3. Let h denote the right-hand side of (6.6). Our goal is to show $\widehat{g} = h$. We prove the case $\lambda > 0$.

Using the definition of the Nyström-approximate kernel, we may write

$$h = \kappa(\cdot, \mathsf{S})\kappa(\mathsf{S}, \mathsf{S})^{-1}\kappa(\mathsf{S}, \mathsf{D})[\kappa(\mathsf{D}, \mathsf{S})\kappa(\mathsf{S}, \mathsf{S})^{-1}\kappa(\mathsf{S}, \mathsf{D}) + \lambda\mathbf{I}]^{\dagger}\boldsymbol{y}.$$

Introduce $\boldsymbol{B} = \kappa(\mathsf{D}, \mathsf{S})\kappa(\mathsf{S}, \mathsf{S})^{-1}$ and $\boldsymbol{C} = \kappa(\mathsf{D}, \mathsf{S})$, and invoke the nonsymmetric ridge regression identity to obtain

$$\begin{aligned} h &= \kappa(\cdot, \mathsf{S})[\kappa(\mathsf{S}, \mathsf{S})^{-1}\kappa(\mathsf{S}, \mathsf{D})\kappa(\mathsf{D}, \mathsf{S}) + \lambda\mathbf{I}]^{-1}\kappa(\mathsf{S}, \mathsf{S})^{-1}\kappa(\mathsf{D}, \mathsf{S})\boldsymbol{y} \\ &= \kappa(\cdot, \mathsf{S})[\kappa(\mathsf{S}, \mathsf{D})\kappa(\mathsf{D}, \mathsf{S}) + \lambda\kappa(\mathsf{S}, \mathsf{S})]^{-1}\kappa(\mathsf{D}, \mathsf{S})\boldsymbol{y}. \end{aligned}$$

Utilize the normal equations (6.5) to complete the proof. $\qquad\square$

Sparse Gaussian process regression

The GP framework provides alternative perspectives on the restricted KRR method. In their original work, Smola and Bartlett [SB00] introduce SGPR as a way of restricting the GPR problem to a subset of basis functions to increase computational efficiency, similar to our original derivation of restricted KRR above.

One may also develop probabilistic interpretations of restricted KRR. The easiest interpretation is that restricted KRR is equivalent to ordinary Gaussian process regression on the *restricted prior*

$$\widehat{p} := \mathbb{E}[p \mid p(\mathsf{S})].$$

The restricted prior $\widehat{p}$ averages out all the randomness in the initial prior p except at the landmark points S. The equivalence of this approach with restricted KRR follows from Theorems 3.13 and 6.3. There is also a more sophisticated probabilistic interpretation due to Titsias [Tit09].

Experiment

Experiments for restricted KRR with RPCHOLESKY and other column Nyström methods on a scientific dataset are provided in [CETW25, §4.2]. Here, we provide a more conceptual experiment to illustrate the benefits of RPCHOLESKY as a method for selecting the subset S for use with restricted KRR.

We consider the task of fitting a function $f : \mathbb{R}^2 \to \mathbb{R}$ from data $\mathsf{D} \subseteq \mathbb{R}^2$ and $y :=$ $f(\mathsf{D}) \in \mathbb{R}^\mathsf{D}$. For concreteness, we choose the function to be $f(x) = \sin((x_1+x_2)/10)$ and D to consist of $n = 10^4$ points in $\mathbb{R}^2$ forming a smiley face with radius 10; see the right panel of Fig. 6.2 for illustration. We use the Matérn-5/2 kernel with bandwidth $\sigma = 40$, and we set the regularization to $\lambda := 0$.

Results for restricted KRR with both RPCHOLESKY-selected points and uniformly selected data points are shown in the right panel of Fig. 6.2. We see that the error, measured as the largest difference between the true function f and the model g on the dataset D,

$$\text{error} := \max_{x \in \mathsf{D}} |f(x) - g(x)|, \tag{6.7}$$

is substantially lower for RPCHOLESKY restricted KRR than uniform restricted KRR. An explanation for why can be seen in the left panel of Fig. 6.2, which shows $k = 100$ points selected either by RPCHOLESKY or uniformly at random. Each eye comprises just $n^{1/2} \ll n$ data points, which are easy for uniform sampling to miss.
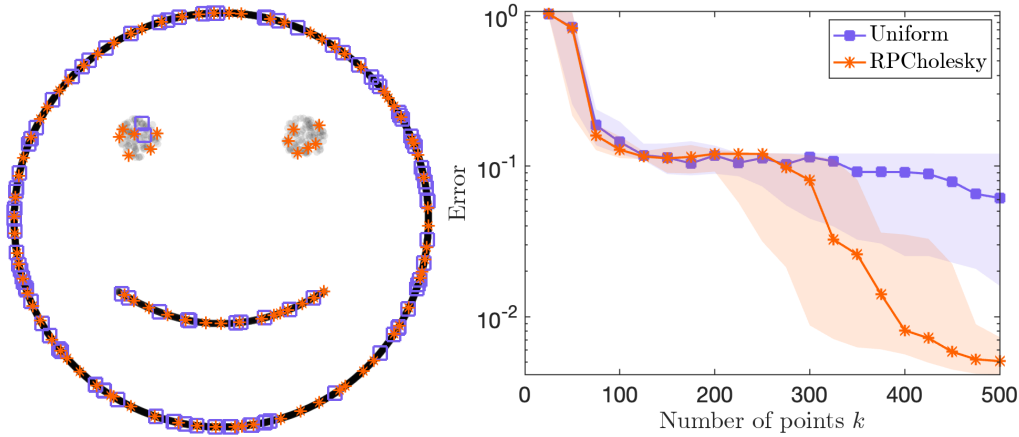
Figure 6.2: *Left*: Error (6.7) for restricted KRR with RPCHOLESKY-selected points (orange asterisks) and uniformly selected points (purple squares). Lines show the median of 100 trials, and shaded regions show the 10% and 90% quantiles. *Right:* Sets of $k = 100$ points selected either by RPCHOLESKY or uniformly at random, overlayed on the data D (black translucent circles). RPCHOLESKY is seen to produce an even coverage of the data, whereas uniform sampling misses the right eye.

RPCHOLESKY, by contrast, selects points that are well-distributed across the whole data set, including both eyes.

6.3 Active learning for kernel interpolation and ridge regression

So far, we have seen two approaches to that use RPCHOLESKY and other subset selection algorithms to improve the efficiency of KRR. First, in Section 6.1, we employed column Nyström approximations to precondition the full-data KRR problem. Under ideal conditions, the cost of this approach was as low as $O(n^2)$ operations. Next, in Section 6.2, we used subset selection algorithms to *restrict* the KRR problem to a set of k selected centers, at a cost of $O(k^2 n)$ operations. In both of these sections, we assumed access to a dataset of n data points $\mathsf{D} \subseteq \mathsf{X}$ *with labels* $\boldsymbol{y} \in \mathbb{K}^n$. In this section, we will consider an *active learning setting* where we are given just the unlabeled training data $\mathsf{D} \subseteq \mathsf{X}$ and must choose what data to label.

For now, we suppose the set of data points D to potentially label is *finite*. Unlike our other settings, the assumption that D is finite is non-vacuous; one can imagine applications where there is an infinite set of points that we could label. We will return to this infinite setting in Chapter 7. Given the set D, we can select k points to label by running RPCHOLESKY (or another subset selection algorithm). After we have a subset $\mathsf{S} \subseteq \mathsf{D}$, we are free to fit whatever model we want to the data S and labels $\boldsymbol{y} \in \mathbb{K}^\mathsf{S}$. In this framework, it is most natural to use a kernel or GP

Program 6.2 rpcholesky_active_krr.m. Active kernel ridge regression with data points selected by RPCHOLESKY. Subroutines krr and acc_rpcholesky are provided in Programs 5.2 and 8.4.

```
function [g,beta,S] = rpcholesky_active_krr(y,kernel,D,lamb,k)
% Input:  Outputs y, kernel function kernel(x,x'), inputs D,
%         regularization lamb, and number of points k
% Output: Kernel interpolant g, coefficients beta, and subset S

Acol = @(S) kernel(D,D(S,:));            % Column subroutine
Asub = @(S) kernel(D(S,:),D(S,:));       % Submatrix subroutine
d = ones(size(D,1),1);                   % Diagonal of kernel matrix
b = ceil(k/2);                           % Block size for acc. RPC
[~,S] = acc_rpcholesky(Acol,Asub,d,k,b); % Subset selection by RPC

[g,beta] = krr(y,kernel,D(S,:),lamb);    % Apply KRR

end
```

method, like kernel interpolation or KRR, to fit the data. One could also combine kernel-based subset selection with any type of machine learning method such as an artificial neural network. Code for active KRR with RPCHOLESKY is provided in Program 6.2, and theoretical guarantees are provided later in Corollary 7.6.

Remark 6.5 (Is this active learning?). The term "active learning" is contested in the literature. Miller and Calder suggest drawing a distinction between *coreset methods*, which "leverage the geometry of the underlying dataset... but not the set of labels observed at labeled points during the active learning process", and *active learning methods*, which use both the geometry of the data set D *and* the observed labels y_x at queried points $x \in D$ [MC23, §3.1]. Other researchers do not make this distinction and refer to any machine learning that chooses points to label as an active learning method [CP19; MMWY22]. We shall adopt the second definition in this work, though our methods would be described as a "coreset method" in Miller and Calder's taxonomy. ◇

To test RPCHOLESKY as a method for active learning, we apply active kernel interpolation to $n = 10^5$ randomly selected points from the SUSY dataset [BSW14]; we hold out an additional n randomly selected points as a test set. Data is standardized. We compare active kernel interpolation with subsets selected uniformly at random to those selected by RPCHOLESKY. As the kernel, we use a Laplace kernel with ℓ_2 distances and bandwidth $\sigma = 4$. As a baseline, we also report the errors with
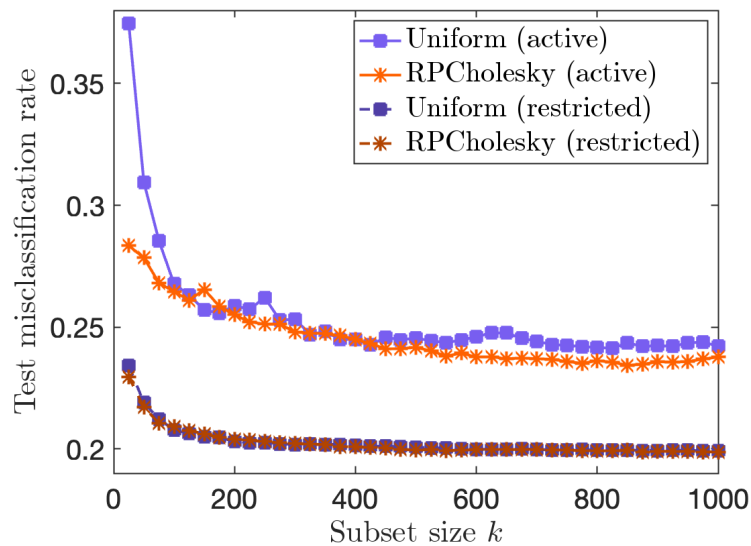
Figure 6.3: Comparison of active (line solid lines) and restricted (dark dashed lines) kernel interpolation with subset S selected by RPCHOLESKY (orange asterisks) or uniformly at random (purple squares).

restricted KRR (with regularization parameter $\lambda = 0$) using the same subsets S.

Results are shown in Fig. 6.3. We see that RPCHOLESKY active kernel interpolation outperforms active kernel interpolation with uniformly selected points for most values of k; the benefits are pronounced when the number of labeled points is small, say, $k < 100$. The baselines results for restricted KRR show much smaller errors than the active learning methods, demonstrating that it is valuable to use label information y_x at each point $x \in D$ if it is available.

C h a p t e r 7

TO INFINITE DIMENSIONS

One should never try to prove anything that is not almost obvious.

A sentiment attributed to Alexandre Grothendieck by Allyn Jackson in
As if summoned from the void, the life of Alexandre Grothendieck [Jac04]

We have seen the RPCHOLESKY algorithm is effective at producing low-rank approximations to a psd matrix. In the previous chapter, we applied this algorithm to the kernel matrix associated with a finite set of data points, and we used the resulting low-rank approximations and subsets of landmark points to accelerate kernel machine learning algorithms.

In Chapter 5, we worked with kernels $\kappa : \mathsf{X} \times \mathsf{X} \to \mathbb{K}$ defined over an infinite base space X, but we only applied RPCHOLESKY to kernel matrices $\kappa(\mathsf{D}, \mathsf{D})$ associated with *finite* subsets $\mathsf{D} \subseteq \mathsf{X}$. It is natural to ask: *Is there an infinite-dimensional version of RPCHOLESKY that we can use to construct low-rank approximations to the infinite kernel function κ and to identify landmark points $\mathsf{S} \subseteq \mathsf{X}$ for the infinite set X?* This chapter answers this question in the affirmative.

Sources. This chapter is based on the following paper:

Ethan N. Epperly and Elvira Moreno. "Kernel Quadrature with Randomly Pivoted Cholesky". In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 65850–65868. URL: https://dl.acm.org/doi/10.5555/3666122.36689 97.

The material has been significantly expanded, including a much lengthier introduction to the functional analysis setting, the new Theorem 7.1 (generalizing Theorem 4.1), and a new application to active kernel interpolation in Section 7.4.

Outline. Section 7.1 sets the stage by introducing appropriate operators and function spaces for infinite-dimensional low-rank approximation. Section 7.2 presents the infinite-dimensional RPCHOLESKY algorithm, and Section 7.3 discusses implementations of this procedure using rejection sampling. Sections 7.4 and 7.5 provide applications to active learning and quadrature.

7.1 Mathematical setting

In this section, we develop a functional analysis setting for low-rank approximation of kernels and subset selections on general, possibly infinite sets, following the treatments given in [HB04; Bac17; BBC19; EM23]. Inspired by the Grothendieck quote at the start of this chapter, our goal will be to develop a function space setting where the infinite dimensional extension of RPCHOLESKY and its analysis is a direct translation of the ordinary matrix setting.

Function spaces. Our setting will be a topological space X endowed with a Borel measure μ, upon which we define an RKHS H of functions $f : X \to \mathbb{K}$. We let $\kappa : X \times X \to \mathbb{K}$ denote the reproducing kernel of H, which we assume is continuous.

This set X supports two spaces of functions, the RKHS H and the space of square integrable functions $L^2(\mu)$. The relation between these spaces will be crucial to our development. To ensure these two spaces place nicely with each other, we make the assumption that the kernel κ is integrable along the diagonal:

$$\int_X \kappa(x, x)\, d\mu(x) < +\infty$$

and that H is dense in $L^2(\mu)$. These assumptions imply that H is compactly embedded in $L^2(\mu)$ [HB04, Prop. 2].

Discovering the integral operator. Since H is compactly embedded in $L^2(\mu)$, the identity mapping $\iota f := f$ defines a compact linear map $\iota : H \to L^2(\mu)$. The maps ι "forgets" that a function f belongs to H and treats it as a function in $L^2(\mu)$. Introduce the symbol $A := \iota^*$ for its adjoint. The adjoint $A : L^2(\mu) \to H$ maps a function $u \in L^2(\mu)$ (which, in general, can be quite rough) to a "smooth" function $Au \in H$. What could this mysterious operator A be?

We shall discover the correct answer by a formal calculation. Let $u \in L^2(\mu)$ and $f \in H$ be arbitrary, and compute the inner product

$$\langle \iota f, u \rangle_{L^2(\mu)} = \int_X \overline{f(x)} u(x)\, d\mu(x).$$

Recall that the adjoint is defined via the relation $\langle \iota f, u \rangle_{L^2(\mu)} = \langle f, \iota^* u \rangle_H$, so we must find a way of introducing the H inner product. To do so, employ the reproducing property $\langle f, \kappa(\cdot, x) \rangle_H = \overline{f(x)}$ to obtain

$$\langle \iota f, u \rangle_{L^2(\mu)} = \int_X \langle f, \kappa(\cdot, x) \rangle_H u(x)\, d\mu(x).$$

Pull the integral over x into the inner product, yielding

$$\langle \iota f, u \rangle_{\mathsf{L}^2(\mu)} = \left\langle f, \int_{\mathsf{X}} \kappa(\cdot, x) u(x) \, \mathrm{d}\mu(x) \right\rangle_{\mathsf{H}} =: \langle f, \iota^* u \rangle_{\mathsf{H}}.$$

We conclude that the adjoint $A = \iota^*$ is the integral operator

$$Au = \int_{\mathsf{X}} \kappa(\cdot, x) u(x) \, \mathrm{d}\mu(x). \tag{7.1}$$

A rigorous version of this argument is provided in [HB04, Prop. 2].

Properties of the integral operator. The integral operator A, defined by (7.1), can also be seen as an operator on $\mathsf{L}^2(\mu)$, which we denote $A_{\mathsf{L}^2(\mu) \to \mathsf{L}^2(\mu)}$. Using the "forgetting" map $\iota : \mathsf{H} \to \mathsf{L}^2(\mu)$, this operator can be written

$$A_{\mathsf{L}^2(\mu) \to \mathsf{L}^2(\mu)} = A_{\mathsf{L}^2(\mu) \to \mathsf{H}} \cdot \iota = \iota^*.$$

Thus, since the inclusion operator ι is compact (since H is compactly embedded in $\mathsf{L}^2(\mu)$), we conclude that $A_{\mathsf{L}^2(\mu) \to \mathsf{L}^2(\mu)}$ is compact and psd.

To this point, we have been very careful about domains and codomains of linear mappings; we will now permit ourselves to be more lax and will use A to refer to the transformation (7.1), whatever its domain and codomain are. Since A is a compact psd operator on $\mathsf{L}^2(\mu)$, it admits a spectral decomposition

$$A = \sum_{i=1}^{\infty} \lambda_i e_i e_i^*, \tag{7.2}$$

where $\lambda_1 \geq \lambda_2 \geq \cdots \geq 0$ are the eigenvalues of A and $\{e_i\}$ form an orthonormal basis of $\mathsf{L}^2(\mu)$. (We will tacitly assume that $\mathsf{L}^2(\mu)$ is infinite-dimensional for this chapter, though the formalism all carries through for finite-dimensional spaces as well.) Here, we let

$$u^* := \langle u, \cdot \rangle_{\mathsf{L}^2(\mu)} : \mathsf{L}^2(\mu) \to \mathbb{K}$$

denote the adjoint of a function $u \in \mathsf{L}^2(\mu)$, identified with the linear transformation $\alpha \mapsto \alpha u$ from $\mathbb{K} \to \mathsf{L}^2(\mu)$.

In addition to being compact, the operator A on $\mathsf{L}^2(\mu)$ is trace-class. Its trace is

$$\mathrm{tr}(A) = \int_{\mathsf{X}} \kappa(x, x) \, \mathrm{d}\mu. \tag{7.3}$$

This conclusion may be derived formally as follows. Observe that an arbitrary function in $\mathsf{L}^2(\mu)$ can be represented a limit $u = \lim_{n \to \infty} (\sum_{i=1}^{n} e_i e_i^*) u$. Therefore,

we compute

$$\int_X \kappa(x,x)\,d\mu = \int_X \sum_{i=1}^{\infty} e_i(x)(e_i^*\kappa(\cdot,x))\,d\mu(x)$$

$$= \int_X \sum_{i=1}^{\infty} \left(\int_X \kappa(x',x)\overline{e_i(x')}\,d\mu(x') \right) e_i(x)\,d\mu(x)$$

$$= \sum_{i=1}^{\infty} \int_X \left(\int_X \kappa(x',x)e_i(x)\,d\mu(x) \right) \overline{e_i(x')}\,d\mu(x')$$

$$= \sum_{i=1}^{\infty} \langle Ae_i, e_i \rangle_{L^2(\mu)} = \sum_{i=1}^{\infty} \lambda_i = \operatorname{tr}(A).$$

A rigorous proof of (7.3) is provided in [SS12, Thm. 3.10].

From the eigendecomposition (7.2), we can define the operator square root

$$A^{1/2} := \sum_{i=1}^{\infty} \lambda_i^{1/2} e_i e_i^* : L^2(\mu) \to L^2(\mu).$$

In fact, the range of the linear transformation $A^{1/2}$ is H and the map $A^{1/2} : L^2(\mu) \to H$ is an isometric embedding. To see this, first take $u \in H$ and compute

$$\|u\|_{L^2(\mu)}^2 = \langle u, u \rangle_{L^2(\mu)} = \langle Au, u \rangle_H = \langle A^{1/2}u, A^{1/2}u \rangle_H = \|A^{1/2}u\|_H^2.$$

The second equality is an invocation of the adjoint relation $\langle u, f \rangle_{L^2(\mu)} = \langle Au, f \rangle_H$. Since H is dense in $L^2(\mu)$, we conclude that $A^{1/2} : L^2(\mu) \to H$ is an isometric embedding.

Since $A^{1/2} : L^2(\mu) \to H$ is an isometric embedding, it follows that $\{A^{1/2}e_i\}_{i=1}^{\infty} = \{\lambda_i^{1/2} e_i\}_{i=1}^{\infty}$ is an orthonormal system in H. In fact, it is an orthonormal basis, and $A^{1/2}$ is an isometry [SS12, Thm. 3.1].

The expression (7.2) constitutes an eigenvalue decomposition for the integral operator A associated with the kernel κ, but what about κ itself? Does it have such an eigendecomposition? Proceeding formally again, we compute

$$\int_X \kappa(x,x')u(x')\,d\mu(x') = Au(x) = \sum_{i=1}^{\infty} \lambda_i e_i(x) \langle e_i, u \rangle_{L^2(\mu)}$$

$$= \sum_{i=1}^{\infty} \int_X \lambda_i e_i(x)\overline{e_i(x')}u(x')\,d\mu(x') = \int_X \left(\sum_{i=1}^{\infty} \lambda_i e_i(x)\overline{e_i(x')} \right) u(x')\,d\mu(x').$$

Since this equation holds for all x and all u, it is natural to conjecture that

$$\kappa(x,x') = \sum_{i=1}^{\infty} \lambda_i e_i(x)\overline{e_i(x')}.$$

Indeed, this *Mercer decomposition* holds true, and the convergence holds pointwise [SS12, Thm. 3.1].

7.2 Infinite-dimensional psd low-rank approximation and RPCHOLESKY

Having established the proper infinite-dimensional setting, we can now formulate the appropriate low-rank approximation problem and devise an infinite-dimensional version of randomly pivoted Cholesky to solve it.

As we saw in Section 5.2, a subset $\mathsf{S} \subseteq \mathsf{X}$ induces a rank-k approximation

$$\widehat{\kappa}_{\mathsf{S}}(x, x') := \kappa(x, \mathsf{S}) \kappa(\mathsf{S}, \mathsf{S})^{\dagger} \kappa(\mathsf{S}, x')$$

to the kernel function κ. It also defines the residual

$$\kappa_{\mathsf{S}} = \kappa - \widehat{\kappa}_{\mathsf{S}}.$$

These objects suggest an infinite-dimensional version of the psd low-rank approximation and column subset selection problems:

Infinite-dimensional column subset selection problem. Find a subset $\mathsf{S} \subseteq \mathsf{X}$ of size k that approximately minimizes the trace error

$$\int_{\mathsf{X}} \kappa_{\mathsf{S}}(x, x) \, \mathrm{d}\mu(x) = \int_{\mathsf{X}} \left[\kappa(x, x) - \kappa(x, \mathsf{S}) \kappa(\mathsf{S}, \mathsf{S})^{\dagger} \kappa(\mathsf{S}, x) \right] \, \mathrm{d}\mu(x).$$

Using the formalism developed in the previous section, we can recast this problem in terms of integral operators. Indeed, introduce the Nyström-approximate operator

$$\widehat{A}_{\mathsf{S}} u := \int_{\mathsf{X}} \widehat{\kappa}_{\mathsf{S}}(\cdot, x) u(x) \, \mathrm{d}\mu(x). \tag{7.4}$$

Since the residual kernel $\kappa_{\mathsf{S}}(x, x) = \kappa(x, x) - \widehat{\kappa}_{\mathsf{S}}(x, x)$ is psd and integrable along the diagonal (Proposition 5.13), the residual operator $A - \widehat{A}_{\mathsf{S}}$ is psd, and the trace error is

$$\mathrm{tr}(A - \widehat{A}_{\mathsf{S}}) = \int_{\mathsf{X}} \kappa_{\mathsf{S}}(x, x) \, \mathrm{d}\mu(x). \tag{7.5}$$

Thus, choosing a subset S to control the integrated diagonal of the residual κ_{S} is equivalent to construction a Nyström approximation of the integral operator A with a small trace error $\mathrm{tr}(A - \widehat{A}_{\mathsf{S}})$.

Given this setup, the infinite-dimensional version of RPCHOLESKY is natural.
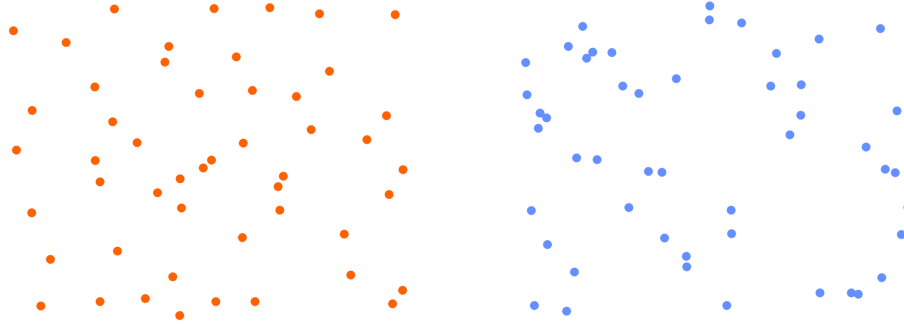
Figure 7.1: 50 random points in the unit square generated by RPCHOLESKY (*left*) and uniform sampling (*right*). Points sampled using RPCHOLESKY are more spread out, demonstrating the repulsive nature of the RPCHOLESKY sampling process. RPCHOLESKY is implemented using the uniform measure $\mu = \text{UNIF}([0, 1]^2)$ and the Laplace kernel with bandwidth 10.

RPCHOLESKY on a general space. Begin from the empty subset $\mathsf{S}_0 := \emptyset$, iterate for $i = 0, 1, 2 \ldots, k - 1$:

1. **Sample** a random pivot

$$s_{i+1} \sim \kappa_{\mathsf{S}_i}(x, x) \, \mathrm{d}\mu(x) \qquad (7.6)$$

   from the diagonal of the residual kernel $\kappa_{\mathsf{S}_i}(x, x)$. Here, $s \sim \nu$ to denotes a random element sampled from an *unnormalized* finite measure ν. That is, $\mathbb{P}\{s \in \mathsf{B}\} = \nu(\mathsf{B})/\nu(\mathsf{X})$ for every measurable set B.

2. **Induct** the sampled pivot into the landmark set, $\mathsf{S}_{i+1} := \mathsf{S}_i \cup \{s_{i+1}\}$.

This procedure is easy to state, but it is unclear how to implement the sampling step (7.6). We will turn to the issue of implementation in Section 7.3.

An illustration of the landmarks selected by RPCHOLESKY on a continuous space is given in Fig. 7.1. We see that the points produced by RPCHOLESKY are more well-distributed across the region, whereas points generated by uniform sampling are more clustered. This example demonstrates that the RPCHOLESKY sampling process is repulsive; once a landmark is selected, it is less likely that nearby points will be selected as landmarks.

The error bounds from Section 4.3 (Theorems 4.1 and 4.3) generalize to infinite-dimensional RPCHOLESKY in the natural way. The proofs are the same, modulo

appropriate details to handle the infinite-dimensional setting; we omit the details.

Theorem 7.1 (Continuous RPCʜᴏʟᴇsᴋʏ: Trace error). *Instate the prevailing notation and assumptions, let $r \geq 1$ and $\varepsilon > 0$, and introduce the relative error of the best rank-r approximation*

$$\eta := \frac{\sum_{i=r+1}^{\infty} \lambda_i}{\sum_{i=1}^{\infty} \lambda_i}.$$

The continuous RPCʜᴏʟᴇsᴋʏ method is guaranteed to produce a pivot set S achieving the guarantee

$$\mathbb{E}\operatorname{tr}(A - \widehat{A}_{\mathsf{S}}) \leq (1 + \varepsilon) \sum_{i=r+1}^{\infty} \lambda_i$$

after running for any number of steps k satisfying

$$k \geq \frac{r}{\varepsilon} + r \log\left(\frac{1}{\varepsilon \eta}\right).$$

Theorem 7.2 (Continuous RPCʜᴏʟᴇsᴋʏ: Spectral norm of expected error). *Instate the prevailing notation and assumptions, let $r \geq 1$ and $\varepsilon > 0$, and introduce the relative error of the best rank-r approximation*

$$\eta := \frac{\sum_{i=r+1}^{\infty} \lambda_i}{\sum_{i=1}^{\infty} \lambda_i}.$$

The continuous RPCʜᴏʟᴇsᴋʏ method is guaranteed to produce a pivot set S achieving the guarantee

$$\lambda_{\max}(\mathbb{E}(A - \widehat{A}_{\mathsf{S}})) \leq b + \varepsilon \sum_{i=r+1}^{\infty} \lambda_i$$

after running for any number of steps k satisfying

$$k \geq \frac{1}{\varepsilon} + r \log\left(\frac{\lambda_1}{b}\right).$$

Here, $\lambda_{\max}(\cdot)$ denotes the largest eigenvalue of a Hermitian operator on $\mathsf{L}^2(\mu)$. In particular, for such k and any function $g \in \mathsf{L}^2(\mu)$,

$$\mathbb{E}\langle g, (A - \widehat{A}_{\mathsf{S}})g\rangle_{\mathsf{L}^2(\mu)} \leq \left(b + \varepsilon \sum_{i=r+1}^{\infty} \lambda_i\right) \|g\|_{\mathsf{L}^2(\mu)}^2.$$

Theorem 7.2 originally appeared as [EM23, Thm. 7]. The infinite-dimensional trace error bound Theorem 7.1 has not appeared in print before; it is an immediate transplant of the finite-dimensional result Theorem 4.1, originally proven in [CETW25, Thm. 1.1], to the infinite setting.

Program 7.1 `rejection_sample.m`. General implementation of rejection sampling from a target distribution $\tau(x)\,\mathrm{d}\mu(x)$ using a proposal distribution $\pi(x)\,\mathrm{d}\mu(x)$.

```matlab
function s = rejection_sample(sample_proposal,pi,tau)
% Input:  Function sample_proposal() generating a sample from
%         proposal distribution, functions pi(p) and tau(p) defining
%         (unnormalized) density functions for proposal and target
% Output: Sample s from proposal_distribution

while true
    s = sample_proposal();     % Sample from proposal
    if rand() < tau(s) / pi(s) % Accept with probability tau(s)/pi(s)
        break
    end
end

end
```

7.3 Implementing RPCHOLESKY using rejection sampling

How could we implement the RPCHOLESKY method in infinite dimensions? Nominally, storing even a single "column" $\kappa(\cdot, s)$ of the "infinite kernel matrix" κ requires infinite memory, which makes a direct implementation of infinite RPCHOLESKY similar to Program 4.1 impossible in general. We circumvent this limitation by implementing RPCHOLESKY in infinite dimensions using an alternative approach based on *rejection sampling*.

Rejection sampling in general

Rejection sampling is a standard technique in probabilistic computing, originally due to von Neumann [von51]. See [Liu04, §2.2] for a contemporary introduction to rejection sampling. Suppose that we are interested in sampling $s \sim \tau(x)\,\mathrm{d}\mu(x)$ from an (unnormalized) target distribution $\tau(x)\,\mathrm{d}\mu(x)$. Assume the target distribution is difficult to sample from directly, but that we have easy access to samples from an (unnormalized) *proposal distribution* $p \sim \pi(x)\,\mathrm{d}\mu(x)$. Suppose additionally that we know that the proposal distribution bounds the target distribution pointwise, $0 \le \tau \le \pi$. With these preliminaries, the rejection sampling procedure is as follows.

> **Rejection sampling.** Repeat until termination:
>
> 1. **_Generate_** a sample $p \sim \pi(x) \, \mathrm{d}\mu(x)$ from the proposal distribution.
>
> 2. **_Accept/reject?_** With probability $\tau(p)/\pi(p)$, *accept* by setting $s := p$ and exiting this loop. Otherwise, *reject* the sample and repeat step 1.

A program implementing rejection sampling is given as Program 7.1.

The correctness of the rejection sampling procedure is established by the following standard result, in essence due to von Neumann [von51].

Fact 7.3 (Rejection sampling). *The rejection sampling procedure works: Provided $\tau > 0$ on a set of positive measure, the procedure terminates with probability 1, and it outputs a sample $s \sim \tau(x) \, \mathrm{d}\mu(x)$.*

The proof is short and simple; see [Liu04, §2.2] for a proof when μ is the Lebesgue measure on $\mathbb{R}^n$. (The proof is no different for general probability spaces.)

Rejection sampling for RPCHOLESKY

We can use rejection sampling to perform the diagonal sampling step for RP-CHOLESKY. We assume access to two primitive operations:

1. **_Kernel evaluations._** We can evaluate the $\kappa(x, x')$ at any pair of points $x, x' \in \mathsf{X}$.

2. **_Sampling from the diagonal._** We have access to an efficient procedure for sampling $p \sim \kappa(x, x) \, \mathrm{d}\mu(x)$.

In typical machine learning applications, these primitives are easily accessible, as (1) the kernel function κ is often given by an explicit functional form and (2) the kernel function often satisfies $\kappa(x, x) = 1$ for all x, so sampling from the diagonal amounts to uniform sampling from μ. As we will see, access to these two weak primitives will suffice to implement RPCHOLESKY in infinite dimensions.

Consider step i of RPCHOLESKY. We seek a sample $s_{i+1} \sim \kappa_{\mathsf{S}_i}(x, x) \, \mathrm{d}\mu(x)$ from the target distribution $\kappa_{\mathsf{S}_i}(x, x) \, \mathrm{d}\mu(x)$. To do so, we will employ rejection sampling with $\kappa(x, x) \, \mathrm{d}\mu(x)$ as the proposal distribution. These distributions form a valid pair for rejection sampling in view of the inequality

$$0 \le \kappa_{\mathsf{S}_i}(x, x) \le \kappa(x, x) \quad \text{for every } x \in \mathsf{X},$$

Program 7.2 `rejection_rpcholesky.m`. Rejection sampling-based implementation of RPCHOLESKY on general spaces.

```matlab
function [S,R] = rejection_rpcholesky(kernel,sample_diagonal,k,d)
% Input:  Kernel function kernel(S,T) giving m*n pairwise evaluations
%         of the kernel function on the rows of an m*d matrix S and
%         n*d matrix T, function sample_diagonal() giving a sample
%         from kernel(x,x) dmu(x) outputted as a row vector,
%         number of landmarks k, dimension d of data
% Output: Set S of k selected landmarks stored as a k*d array,
%         Cholesky factor R = chol(kernel(S,S))

S = zeros(k,d);                          % Landmark set
R = zeros(k,k);                          % Cholesky factor
for i = 0:(k-1)
    while true
        S(i+1,:) = sample_diagonal(); % Proposal

        % Quantities needed to evaluate ratio
        c = R(1:i,1:i)' \ kernel(S(1:i,:),S(i+1,:));
        kss = kernel(S(i+1,:),S(i+1,:));
        d = kss - norm(c)^2;
        if rand() < d / kss        % Accept or reject
            % Update Cholesky factor upon acceptance
            R(1:i,i+1) = c;
            R(i+1,i+1) = sqrt(d);
            break                   % Exit loop and sample next point
        end
    end
end
```

which holds due to Proposition 5.13.

Since we have efficient access to proposals $p \sim \kappa(x,x)\,\mathrm{d}\mu(x)$, implementing this procedure just requires the ability to evaluate the ratio

$$\mathrm{ratio}_i(x) := \frac{\kappa_{\mathsf{S}_i}(x,x)}{\kappa(x,x)} = \frac{\kappa(x,x) - \widehat{\kappa}_{\mathsf{S}_i}(x,x)}{\kappa(x,x)}.$$

To do so, we maintain a Cholesky decomposition $\kappa(\mathsf{S}_i, \mathsf{S}_i) = \boldsymbol{R}_i^* \boldsymbol{R}_i$ of the currently selected kernel submatrix and evaluate $\mathrm{ratio}_i(x)$ using the formula

$$\mathrm{ratio}_i(x) = \frac{\kappa(x,x) - \|\boldsymbol{c}_i(x)\|^2}{\kappa(x,x)} \quad \text{where } \boldsymbol{c}_i(x) := \boldsymbol{R}_i^{-*}\kappa(\mathsf{S}_i,x). \qquad (7.7)$$

Once a new pivot s_{i+1} is accepted, the Cholesky factor can be updated using the

formula

$$\boldsymbol{R}_{i+1} := \begin{bmatrix} \boldsymbol{R}_i & \boldsymbol{c}_i(s_{i+1}) \\ \boldsymbol{0} & (\kappa(s_{i+1}, s_{i+1}) - \|\boldsymbol{c}_i(s_{i+1})\|^2)^{1/2} \end{bmatrix}.$$

This rejection sampling-based implementation of RPCHOLESKY is called REJEC-TIONRPCHOLESKY. An implementation is given in Program 7.2.

Computational cost

The computational cost of REJECTIONRPCHOLESKY can be characterized in multiple ways. Each execution of the rejection sampling loop requires at most k kernel function evaluations and $O(k^2)$ operations to perform the triangular solve in (7.7). Thus, b proposals of REJECTIONRPCHOLESKY expends at most bk kernel function evaluations and $O(bk^2)$ additional arithmetic operations, where $k \leq b$ denotes the number of accepted proposals. An end-to-end cost estimate is provided by the following result:

Proposition 7.4 (REJECTIONRPCHOLESKY). *Program 7.2 is correct: It produces exact samples from the continuous RPCHOLESKY algorithm. To measure the computational cost, introduce the relative trace error of the best rank-$(i-1)$ approximation:*

$$\eta_i := \frac{\sum_{j=i}^{\infty} \lambda_i}{\sum_{j=1}^{\infty} \lambda_i}.$$

The expected computational cost of Program 7.2 is at most

$$O\left(\sum_{i=1}^{k} \frac{1}{\eta_i} \right) \leq O\left(\frac{k}{\eta_k} \right) \qquad \text{rejection sampling steps,}$$

$$O\left(\sum_{i=1}^{k} \frac{i}{\eta_i} \right) \leq O\left(\frac{k^2}{\eta_k} \right) \qquad \text{kernel function evaluations,}$$

$$O\left(\sum_{i=1}^{k} \frac{i^2}{\eta_i} \right) \leq O\left(\frac{k^3}{\eta_k} \right) \qquad \text{additional arithmetic operations.}$$

This result is [EM23, Thm. 4]. The main idea is that, to sample s_{i+1}, the probability of each proposal being accepted is $\mathrm{tr}(A - \widehat{A}_{\{s_1,\dots,s_i\}})/\mathrm{tr}(A) \geq \eta_i$. Ergo, the expected number of proposals is a geometric random variable with expectation $1/\eta_i$, and the result follows.

This result demonstrates that rejection sampling-based RPCHOLESKY suffers from a *curse of smoothness*: the runtime of the algorithm is higher when the eigenvalues λ_i decrease at a rapid rate. The reason for this curse is simple. When the eigenvalues

decay rapidly, the Nyström approximations $\widehat{\kappa}_{S_i}$ become increasingly close to the kernel κ. As such, the acceptance probability (7.7) goes down, requiring more proposals to achieve each acceptance.

Elvira Moreno and I proposed a potentially costly fix for the curse of smoothness in [EM23, Alg. 4]. Here is the idea: When the algorithm begins rejecting proposals at a high rate (say, 10 rejections in a row), solve a global minimization problem

$$\alpha \leftarrow \max_{x \in X} \mathrm{ratio}_i(x),$$

where $\mathrm{ratio}_i(x)$ was defined in (7.7). Then, run rejection sampling with the rescaled proposal distribution $\alpha^{-1}\kappa(x,x)\,\mathrm{d}\mu(x)$. This rescaling does not effect the distribution of proposals $p \sim \alpha^{-1}\kappa(x,x)\,\mathrm{d}\mu(x)$, but the acceptance probabilities are scaled by a factor α^{-1}. Since $\alpha \leq 1$, this has the effect of boosting the acceptance probabilities. This procedure can be very effective; our paper [EM23] reports speedups of 39× over the non-accelerated version. However, the accelerated REJECTIONRPC-HOLESKY procedure requires solving a global nonconvex optimization problem over the domain X, which can be computationally costly. The development of tractable rejection sampling schemes for RPCHOLESKY that avoid nonconvex optimization is an interesting direction for future research.

7.4 Application: Active kernel interpolation

Consider the active regression task of approximating a function $f \in H$ whose value we may be query at k locations $S = \{s_1, \ldots, s_k\} \subseteq X$, which we are free to choose. A natural approach to this problem is to select the locations S using RPCHOLESKY, then interpolate the function values at those points using kernel interpolation. Throughout this section, we let $\widehat{f}_S$ denote the kernel interpolant through the points S.

The success of this active kernel interpolation strategy can be analyzed by combining the following known result for kernel interpolation with the trace error bound Theorem 7.1 for RPCHOLESKY:

Proposition 7.5 (Kernel interpolation: mean-squared error)**.** *Let $f \in H$ be a function and let $\widehat{f}_S$ be its kernel interpolant through the points S. We have the error bound*

$$\|f - \widehat{f}_S\|_{L^2(\mu)}^2 \leq \|f\|_H^2 \cdot \mathrm{tr}(A - \widehat{A}_S).$$

Proof. The bound follows by integrating the pointwise bound Theorem 5.16 and using the identity (7.5) for $\mathrm{tr}(A - \widehat{A}_S)$. $\square$
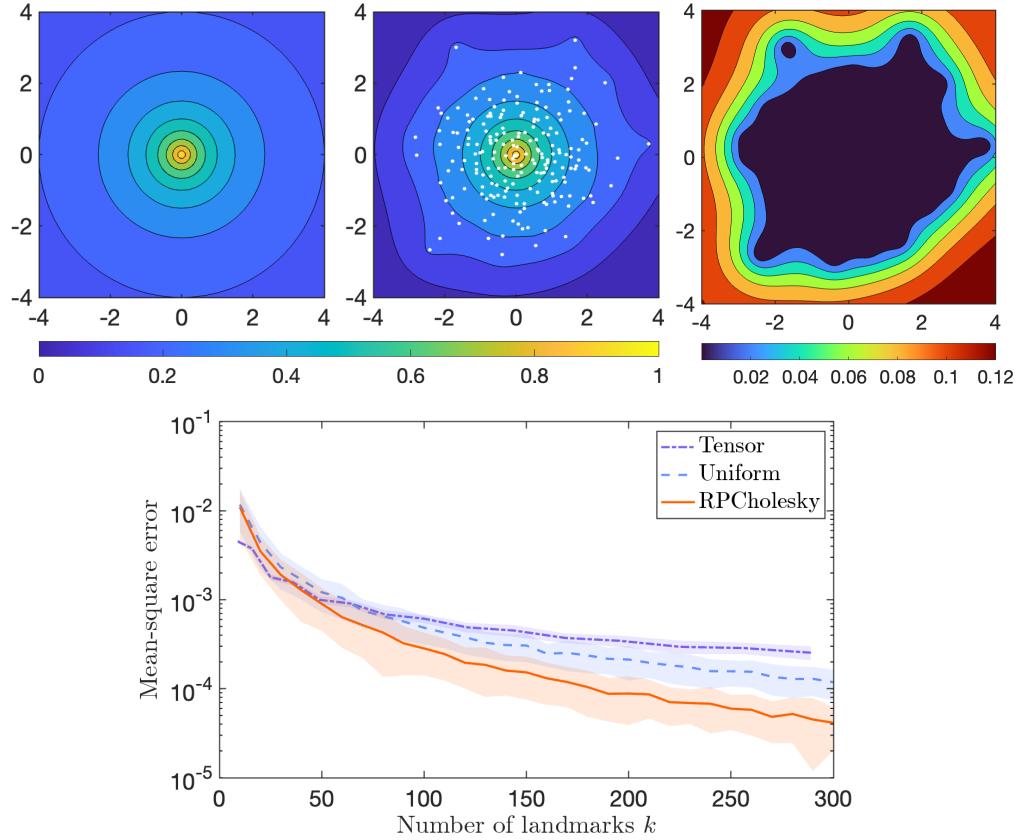
Figure 7.2: *Top:* Function f (*left*), kernel interpolant $\widehat{f}_{\mathsf{S}}$ with RPCHOLESKY (*middle*), and error $|f - \widehat{f}_{\mathsf{S}}|$ (*right*). Landmarks selected by RPCHOLESKY are overlaid as white dots in the middle plot. Function f, kernel κ, and measure μ are described in the text. *Bottom:* mean-squared error of kernel interpolation through three different sets of nodes. Lines show a mean of 100 trials, and shaded regions show one standard deviation.

Combining with Theorem 7.1 yields the following immediate corollary:

Corollary 7.6 (Kernel interpolation with RPCHOLESKY). *With the setting and value of k from Theorem 7.1, a set S picked by k steps of RPCHOLESKY satisfies the bound*

$$\mathbb{E}\|f - \widehat{f}_{\mathsf{S}}\|^2_{\mathsf{L}^2(\mu)} \le \|f\|^2_{\mathsf{H}} \cdot (1+\varepsilon) \sum_{i=r+1}^{\infty} \lambda_i.$$

Moreover, we have an estimate that is uniform over all functions in the RKHS:

$$\mathbb{E} \sup \left\{ \frac{\|f - \widehat{f}_{\mathsf{S}}\|^2_{\mathsf{L}^2(\mu)}}{\|f\|^2_{\mathsf{H}}} : f \in \mathsf{H} \setminus \{0\} \right\} \le (1+\varepsilon) \sum_{i=r+1}^{\infty} \lambda_i.$$

An example of the active interpolation methdology is shown in Fig. 7.2. Here, we apply kernel interpolation to approximate the function

$$f(\boldsymbol{x}) = (1 + \|\boldsymbol{x}\|)^{-1}$$

on the domain $\mathsf{X} \coloneqq \mathbb{R}^2$. We use the standard Gaussian measure $\mu = \text{Normal}_{\mathbb{R}}(\mathbf{0}, \mathbf{I}_2)$ and set κ to be the Laplace kernel with bandwidth 1.

The top panels in Fig. 7.2 illustrate the approximation produced by active kernel interpolation with RPCholesky-selected landmarks. The left panel plots the target function f over to the finite window $[-4, 4]^2$. The middle panel shows the kernel interpolant $\widehat{f}_{\mathsf{S}}$ computing by interpolating through 200 points selected by RPCholesky, shown as white dots. The right panel shows the absolute error $|f - \widehat{f}_{\mathsf{S}}|$. RPCholesky kernel interpolation produces a small error near the origin, with growing errors farther away. This behavior is expected, as the measure μ places most of its mass near the center, leading most of the landmarks to be placed there. Fortunately, higher error in regions of small measure is usually acceptable in applications.

The bottom panel of Fig. 7.2 compares the mean-squared error of three different sets of kernel interpolation through three sets of landmarks: uniformly random points $s_1, \ldots, s_k \overset{\text{iid}}{\sim} \mu$, points selected by RPCholesky, and points from a tensor product grid $\mathsf{S} = \mathsf{E} \times \mathsf{E}$ where $\mathsf{E} = \{\Phi^{-1}((2i - 1)/2\sqrt{k}) : i = 1, \ldots, k\}$. Here, Φ denotes the cdf of the standard Gaussian distribution $\text{Normal}_{\mathbb{R}}(0, 1)$. We estimate the mean-squared error $\|f - \widehat{f}_{\mathsf{S}}\|^2_{\mathsf{L}^2(\mu)}$ using the Monte Carlo estimator $s^{-1} \sum_{i=1}^{s} (f(x_i) - \widehat{f}_{\mathsf{S}}(x_i))^2$ where $x_1, \ldots, x_s \overset{\text{iid}}{\sim} \mu$; we use $s = 10^3$ for these experiments. For each value of k, we display the mean of 100 trials; error bars show one standard deviation. The tensor product nodes are deterministic, so the error bars capture only the Monte Carlo fluctuations in the estimated mean-squared error, which are observed to be small. The RPCholesky method achieves the lowest error of these three point sets.

Remark 7.7 (Beyond kernel interpolation). The continuous RPCholesky method provides a general strategy for selecting points to label during active learning. While we have used kernel interpolation as the learning method in this section, one can also combine RPCholesky with any other functional regression technique. Kernel ridge regression is one natural choice, but one can also use nonlinear approximation methods such as artificial neural networks. ◇

7.5 Application: Kernel quadrature

Another task we can apply RPCholesky to is numerical quadrature. Given a weight function $u \in L^2(\mu)$, the task is to devise weights $w \in \mathbb{K}^k$ and nodes $s_1, \ldots, s_k \in X$ so that

$$\int_X \overline{u(x)} f(x) \, d\mu(x) \approx \sum_{i=1}^{k} \overline{w_i} f(s_i) \quad \text{for all } f \in H.$$

To simplify notation in later parts of this section, I have made the unorthodox choice to complex conjugate both the weight function u and the quadrature weights w_i in this expression. The quality of a quadrature scheme (i.e., both weights and nodes) for a given function f is quantified by the error:

$$\text{Err}(S, w, f) := \left| \int_X \overline{u(x)} f(x) \, d\mu(x) - \sum_{i=1}^{k} \overline{w_i} f(s_i) \right|. \tag{7.8}$$

We will use RPCholesky to pick the nodes $S = \{s_1, \ldots, s_k\}$, and there is a standard procedure for picking the weights w. We begin by reviewing these ideal weights and then prove error bounds for optimally weighted RPCholesky quadrature.

The ideal weights. There are five equivalent ways of characterizing the ideal weights, which I summarized in [Epp23a]. We quickly derive the ideal weights and review these interpretations, as each sheds different light on the kernel quadrature problem.

Characterization #1: Minimizing the worst-case error. Under this interpretation, we choose w to minimize the worst-case value of the error (7.8) over all $f \in H$ with $\|f\|_H \le 1$. To derive the worst-case error, use the integral operator A and the reproducing property to rewrite the error (7.8) as

$$\text{Err}(S, w, f) = \left| \langle u, f \rangle_{L^2(\mu)} - \sum_{i=1}^{k} \overline{w_i} \langle \kappa(\cdot, s_i), f \rangle_H \right| = |\langle Au - \kappa(\cdot, S)w, f \rangle_H|. \tag{7.9}$$

By the Cauchy–Schwarz inequality, the maximum value of this quantity over all f with $\|f\|_H \le 1$ is

$$\text{Err}(S, w) := \max_{\|f\|_H \le 1} \text{Err}(S, w, f) = \|Au - \kappa(\cdot, S)w\|_H. \tag{7.10}$$

Minimizing this quantity over w is a linear least-squares problem. The normal equations characterizing the minimizer $w_\star$ are

$$\kappa(S, S)w_\star = Au(S) = \left(\int_X \kappa(s_i, x)u(x) \, d\mu(x) : 1 \le i \le k \right). \tag{7.11}$$

We call the solution to this system the *ideal weights* $w_\star$.

Characterization #2: Exactness. A classical way of deriving a quadrature rule on k nodes is to require that it integrate k chosen functions exactly. Here, the natural choice for k functions are the kernel functions $f_i := \kappa(\cdot, s_i)$. Enforcing exactness $\mathrm{Err}(\mathsf{S}, w, f_i) = 0$ on these f_i yields the same system (7.11) characterizing the ideal weights.

Characterization #3: Interpolate and integrate. Another classical approach to deriving quadrature rules is to interpolate the function at the nodes, then integrate the interpolant. Using kernel interpolation as our method of interpolation, this approach yields ideally weighted kernel quadrature:

$$\sum_{i=1}^{k} \overline{w_\star(i)} f(s_i) = \int_{\mathsf{X}} \overline{u(x)} \widehat{f_\mathsf{S}}(x) \, \mathrm{d}\mu(x).$$

Characterization #4: Conditional expectation. Our last two interpretations will drawn on a Gaussian process frame. Imagine the function $f \sim \mathrm{GP}(\kappa)$ is drawn from a Gaussian process with covariance function κ. Under this model, the integral $\int_{\mathsf{X}} f(x) \overline{u(x)} \, \mathrm{d}\mu(x)$ is a random quantity, and we can obtain a quadrature rule by evaluating the conditional expectation of this random quantity conditional on the value $f(s_i)$ of the function at queried positions s_i. Evaluating this conditional expectation also yields kernel quadrature with the ideal weights:

$$\sum_{i=1}^{k} \overline{w_\star(i)} f(s_i) = \mathbb{E}\left[\int_{\mathsf{X}} \overline{u(x)} f(x) \, \mathrm{d}\mu(x) \,\middle|\, f(\mathsf{S}) \right].$$

Based on this interpretation, ideally weighted kernel quadrature is sometimes called *Bayesian quadrature*.

Characterization #5: Minimizing the mean-squared error. As a last notion of how the "ideal weights" could be designed, we could also choose the weights to minimize the *mean-squared error* of the integral for a function $f \sim \mathrm{GP}(\kappa)$ drawn from the Gaussian process $\mathrm{GP}(\kappa)$:

$$\mathrm{MSE}(\mathsf{S}, w) := \mathbb{E}\left| \int_{\mathsf{X}} \overline{u(x)} f(x) \, \mathrm{d}\mu(x) - \sum_{i=1}^{k} \overline{w_i} f(s_i) \right|^2.$$

We compute this mean-squared error formally. Begin by re-expressing the mean-squared error using the H-inner product, following (7.9):

$$\mathrm{MSE}(\mathsf{S}, w) = |\langle Au - \kappa(\cdot, \mathsf{S}) w, f \rangle_\mathsf{H}|^2$$

Now, employ a Karhunen-Loève expansion $f = \sum_{i=1}^{\infty} z_i(\lambda_i^{1/2} e_i)$. The functions $\lambda_i^{1/2} e_i$ (defined above in (7.2)) form an orthonormal basis for H, and the random variables $z_1, z_2, \ldots \overset{\text{iid}}{\sim} \text{NORMAL}_{\mathbb{K}}(0, 1)$ are iid and Gaussian. Similarly, we may decompose $Au - \kappa(\cdot, \mathsf{S})w = \sum_{i=1}^{\infty} b_i(\lambda_i^{1/2} e_i)$ in this orthonormal basis is well. The sequence $(b_i)_{i=1}^{\infty}$ is square summable, and it admits the Pythagorean identity $\sum_{i=1}^{\infty} |b_i|^2 = \|Au - \kappa(\cdot, \mathsf{S})w\|_{\mathsf{H}}^2$. By orthonormality of $\{\lambda_i^{1/2} e_i\}_{i=1}^{\infty}$, we compute

$$\text{MSE}(\mathsf{S}, w) = \mathbb{E} \, |\langle Au - \kappa(\cdot, \mathsf{S})w, f\rangle_{\mathsf{H}}|^2 = \mathbb{E} \left| \sum_{i=1}^{\infty} z_i b_i \right|^2$$

$$= \sum_{i=1}^{\infty} |b_i|^2 = \|Au - \kappa(\cdot, \mathsf{S})w\|_{\mathsf{H}}^2.$$

The mean-squared error $\text{MSE}(\mathsf{S}, w)$ for a function $f \sim \text{GP}(\kappa)$ is equal to square the worst-case error $\text{Err}(\mathsf{S}, w)$ over all $f \in \mathsf{H}$ with $\|f\|_{\mathsf{H}} \leq 1$, in view of (7.10). Thus, the same set of ideal weights minimizes both the mean-squared error and worst-case error.

Error bounds for ideally weighted kernel quadrature. The fact that five different definitions of the ideal weights all coincide is a powerful demonstration of the robustness of the RKHS and Gaussian process formalisms. Having thoroughly convinced ourselves that the ideal weights given by (7.11) are, in five senses, the natural way of weighting quadrature rules on RKHSs (or with respect to Gaussian processes), let us now analysis the error for ideally weighted kernel quadrature.

Introduce the error

$$\text{Err}(\mathsf{S}) := \text{Err}(\mathsf{S}, w_{\star})$$

for ideally weighted kernel quadrature. This quantity expresses both the worst-case quadrature error over a function in the unit ball of H and the root-mean-squared error of a function drawn from the corresponding Gaussian process (see interpretations #1 and #5 above). Combing formulas (7.10) and (7.11), this quantity may be written

$$\text{Err}(\mathsf{S})^2 = \left\| Au - \kappa(\cdot, \mathsf{S})\kappa(\mathsf{S}, \mathsf{S})^{-1} Au(\mathsf{S}) \right\|_{\mathsf{H}}^2. \tag{7.12}$$

(We assume here and throughout this section that $\kappa(\mathsf{S}, \mathsf{S})$ is nonsingular.) Introducing the definition of A, we obtain

$$\text{Err}(\mathsf{S})^2 = \left\| \int_{\mathsf{X}} \kappa(\cdot, x)u(x) \, \mathrm{d}\mu(x) - \int_{\mathsf{X}} \kappa(\cdot, \mathsf{S})\kappa(\mathsf{S}, \mathsf{S})^{-1}\kappa(\mathsf{S}, x)u(x) \, \mathrm{d}\mu(x) \right\|_{\mathsf{H}}^2$$

$$= \left\| (A - \widehat{A}_{\mathsf{S}})u \right\|_{\mathsf{H}}^2.$$

In the second line, we recognize the Nyström approximate integral operator $\widehat{A}_S$, defined above in (7.4). In Section 7.1, we saw that $A^{1/2}$ is an isometry and a bijection between $L^2(\mu)$ and H. As such, it has an inverse, which we will denote $A^{-1/2}$, that is an isometry and bijection from H to $L^2(\mu)$. This transformations acts according to the rule

$$A^{-1/2}f = \sum_{i=1}^{\infty} \lambda_i^{-1/2} e_i(e_i^* f) \quad \text{for } f \in H.$$

Additionally, for $f \in H$ for which the sequence $\{\lambda_i^{-1}(e_i^* f)\}$ is square-summable we may also define a map A^{-1} by the formula

$$A^{-1}f = \sum_{i=1}^{\infty} \lambda_i^{-1} e_i(e_i^* f).$$

Using these linear maps, we may bound

$$\begin{aligned}
\text{Err}(S)^2 &= \left\| A^{-1/2}(A - \widehat{A}_S)u \right\|_{L^2(\mu)}^2 \\
&= \langle (A - \widehat{A}_S)A^{-1}(A - \widehat{A}_S)u, u \rangle_{L^2(\mu)} \leq \langle (A - \widehat{A}_S)u, u \rangle_{L^2(\mu)}.
\end{aligned}$$

The last inequality is valid because $A - \widehat{A}_S$ and $\widehat{A}_S$ are both psd operators. We have established the following property:

Proposition 7.8 (Kernel quadrature error). *The worst-case quadrature error* $\text{Err}(S)$ *defined in* (7.12) *admits the bound*

$$\text{Err}(S)^2 \leq \langle (A - \widehat{A}_S)u, u \rangle_{L^2(\mu)}.$$

Combining with Theorem 7.2 immediately yields a result for RPCHOLESKY kernel quadrature with ideal weights.

Corollary 7.9 (Kernel quadrature with RPCHOLESKY). *Instate the prevailing notation and assumptions, and let* $r \geq 0$ *and* $\varepsilon > 0$. *Then*

$$\mathbb{E}\,\text{Err}(S)^2 \leq \varepsilon \cdot \|u\|_{L^2(\mu)}^2 \cdot \sum_{r+1}^{\infty} \lambda_r$$

where S *is selected by running* k *steps of RPCHOLESKY where*

$$k \geq r \log \left(\frac{2\lambda_1}{\varepsilon \sum_{i=r+1}^{\infty} \lambda_i} \right) + \frac{2}{\varepsilon}.$$

Program 7.3 `kernel_quad_wts.m`. Program to compute the ideal kernel quadrature weights for computing integrals $\int_X f(x)\overline{u(x)}\,d\mu(x)$ of functions f drawn from an RKHS.

```
function [w,integrator] = kernel_quad_wts(kernel,S,Au)
% Input:  Kernel function kernel(x,x'), landmark set S, and integrals
%         Au = int kernel(S,x) u(x) dx
% Output: Ideal weights w and function integrator(f) computing the
%         ideally weighted quadrature approximation w'*f(S)

w = kernel(S,S) \ Au;
integrator = @(f) w'*f(S);

end
```

Computational considerations. Implementing ideally weighted kernel quadrature requires evaluating the integrals

$$\int_X \kappa(s_i,x)u(x)\,d\mu(x) \quad \text{for } i = 1,2,\ldots,k \tag{7.13}$$

that compose the right-hand side of the system (7.11). Many schemes for quadrature on RKHSs have been proposed that avoid the computation of these exact integrals [DM22; SDM22; HOL22], but the rate of convergence is reduced. There is no obvious way to obtain *spectrally accurate* kernel quadrature schemes without high-accuracy evaluation of the integrals (7.13).

Example. As an illustration, we consider a simple problem in two dimensions. The measure is $\mu = \text{NORMAL}(\mathbf{0},\mathbf{I}_2)$, the kernel is square-exponential with bandwidth 1, the weight function is $u \equiv 1$, and the function is $f(x) = \cos(\|x\|)$. The integral is known exactly:

$$\int_X f(x)\,d\mu(x) = 1 - \sqrt{2}F(1/\sqrt{2}) = 0.27522154099292\ldots$$

Here, F is the Dawson integral (`DawsonF` in Mathematica). The values of Au are also known exactly:

$$Au(x) = \frac{1}{2}\exp\left(-\frac{\|x\|^2}{4}\right).$$

Figure 7.3 shows the results. The left panel shows $k = 30$ nodes selected by RPCHOLESKY, with the shading indicating the diagonal values $\kappa_S(x,x)$ of the residual kernel. The right panel shows the relative error for the outputs of ideally weighted kernel quadrature with $5 \leq k \leq 60$ nodes selected with RPCHOLESKY
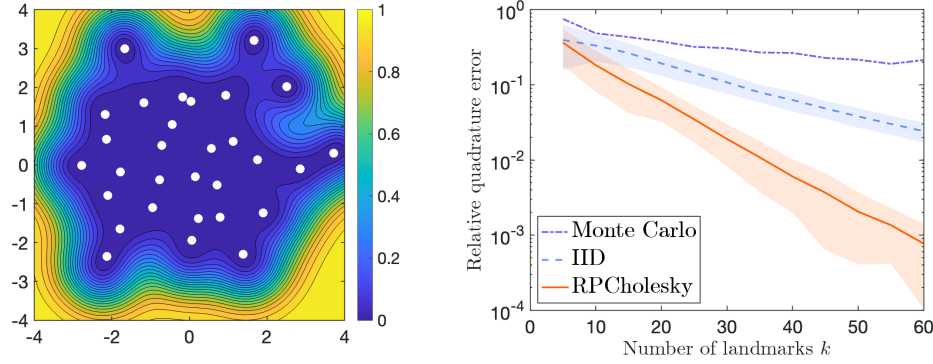
Figure 7.3: *Left:* Quadrature nodes selected by RPCHOLESKY ($k = 30$, white). Shading shows the diagonal elements of the residual kernel $\kappa_S(\boldsymbol{x}, \boldsymbol{x})$. *Right:* Error in evaluating integral $\int_X f(\boldsymbol{x}) \, d\mu(\boldsymbol{x})$ using kernel quadrature for different numbers of nodes k, selected by RPCHOLESKY or by iid sampling. We also show the error of the simple Monte Carlo estimate $\int_X f(\boldsymbol{x}) \, d\mu(\boldsymbol{x}) \approx k^{-1} \sum_{i=1}^{k} f(\boldsymbol{x}_i)$ for $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k \overset{\text{iid}}{\sim} \mu$. The function f, kernel κ, and measure μ are described in the text. Lines show a mean of 100 trials, and shaded regions show one standard deviation.

and by iid sampling. We also compare the simple Monte Carlo integral estimate $\int_X f(\boldsymbol{x}) \, d\mu(\boldsymbol{x}) \approx k^{-1} \sum_{i=1}^{k} f(\boldsymbol{x}_i)$ for $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k \overset{\text{iid}}{\sim} \mu$. The rate of convergence for RPCHOLESKY kernel quadrature is faster than both other methods.

See [EM23] for more experiments with RPCHOLESKY quadrature, accuracy and timing comparisons of RPCHOLESKY quadrature to more methods for kernel quadrature [BBC20; DM22; SDM22; HOL22], and an application to chemistry.

C h a p t e r   8

BLOCKED ALGORITHMS

> *Since all machines have stores of finite size often divided up into high*
> *speed and auxiliary sections, storage considerations often have a vitally*
> *important part to play.*
>
> Jim Wilkinson, *The use of iterative methods for finding the latent roots*
> *and vectors of matrices* [Wil55]

When designing algorithms, we often operate under the convenient fiction that all memory elements can be accessed equally quickly and that the runtime of an algorithm is roughly proportional to the total number of arithmetic operations. Unfortunately, the performance of algorithms in practice is more complicated than this simple model suggests. In particular, matrix algorithms tend to be more performant when data is processed in *blocks* rather than column-by-column or entry-by-entry. This chapter responds to this reality by discussing improved variants of the RPCHOLESKY algorithm that are more hardware-efficient and utilize block matrix computations.

Sources. This chapter is based on the paper

Ethan N. Epperly, Joel A. Tropp, and Robert J. Webber. *Embrace Rejection: Kernel Matrix Approximation by Accelerated Randomly Pivoted Cholesky*. Apr. 2025. arXiv: `2410.03969v3`,

though some of the content appears in the original RPCHOLESKY paper

Yifan Chen, Ethan N. Epperly, Joel A. Tropp, and Robert J. Webber. "Randomly Pivoted Cholesky: Practical Approximation of a Kernel Matrix with Few Entry Evaluations". In: *Communications on Pure and Applied Mathematics* 78.5 (2025), pp. 995–1041. DOI: `10.1002/cpa.22234`.

It expands on these references by providing a more detailed discussion of the benefits of block matrix computations (Section 8.1) and a more thorough discussion of the RBRP Cholesky method of [DCMP24] (Section 8.3).

Notation. Throughout this chapter, we will be interested in computing a pivoted partial Cholesky decomposition of a psd matrix $A \in \mathbb{K}^{n \times n}$ of size n. The pivots will

be denoted $s_1, \ldots, s_k$, and $A^{(i)}$ denotes the residual of the decomposition after the first i elimination steps:

$$A^{(i)} = A - A\langle\{s_1, \ldots, s_i\}\rangle. \tag{8.1}$$

The matrix $F \in \mathbb{K}^{n \times k}$ denotes the factor matrix computed by the pivoted partial Cholesky algorithm (Program 3.1), which satisfies $A\langle\{s_1, \ldots, s_i\}\rangle = F(:, 1 : i)F(:, 1 : i)^*$ for every i.

Outline. Section 8.1 discusses the benefits of blocking for matrix computations with a focus on kernel computations in particular. The next three sections present three blocked versions of the RPCHOLESKY algorithm, referred to as *block RPCHOLESKY* (Section 8.2), *RBRP Cholesky* (Section 8.3), and *accelerated RPCHOLESKY* (Section 8.4). A comparison of these methods appears in Section 8.5.

8.1 Why blocking?

Why are blocked algorithms faster than unblocked algorithms? The answer boils down to *memory transfers*. Applied mathematicians and computer scientists tend to analyze numerical algorithms by counting the number of arithmetic operations. However, on modern computing architectures, movement of data between random access memory and the cache can become the rate-limiting factor in the speed of algorithms, exceeding the cost of floating point operations [DFFG+03, pp. 13, 581–582].

A simplified model for kernel matrix computations

To get a sense of why memory transfer costs can make blocked algorithms faster than unblocked methods, consider the following simplified scenario. Suppose we wish to generate columns of a kernel matrix $A = \kappa(\mathsf{D}, \mathsf{D})$, where $\mathsf{D} \subseteq \mathsf{X}$ is a set of n data points taking values in some set X. We consider an extremely simplified computing machine with a memory hierarchy consisting of two layers; the data elements $x \in \mathsf{D}$ are stored in *memory*. In order to perform computations on data elements, they must be moved from memory into the *cache*. However, the cache has a limit size and is only able to store $1 \ll t \ll n$ elements of X at once.

First, suppose we generate columns of A one at a time, as is done in the standard RPCHOLESKY algorithm. To generate each column $\kappa(\mathsf{D}, x)$ requires bringing each entry of $x' \in \mathsf{D}$ into memory one at a time and evaluating $\kappa(x', x)$. Since the cache has $t \ll n$ entries, most entries of D must be brought back from memory to the cache

for each column generation. Thus, generating k columns requires $O(kn)$ memory transfers.

Now contrast this with a strategy where we generate columns in *blocks* B of size $|\mathsf{B}| = b \le t - 1$. To evaluate a block of columns $\kappa(\mathsf{D}, \mathsf{B})$, we can now generate the matrix row-wise $\kappa(x', \mathsf{B})$, moving each $x' \in \mathsf{D}$ from memory to the cache in sequence. The consequence is we can generate *all b* columns at once using just n memory transfers. Generating $k \ge b$ columns in blocks of size b requires $O(nk/b)$ memory transfers, a factor b smaller than the one-column-at-a-time approach. Thus, for an algorithm whose runtime is dominated by memory movement, the blocked algorithm can be significantly faster.

Remark 8.1 (Cache and memory: Models and reality). The model we have used in this section is a simplified version of the *ideal cache model* [FLPR99]. The memory architecture for a modern computer is significantly more complicated than the basic description given here. In particular, memory hierarchies are divided into many more than two levels and data is moved between levels in chunks of fixed size called *cache lines*. Given this complexity, we advise against using the model described above to make quantitative predictions about the runtime of algorithms (i.e., it is not typically true that generating columns in blocks of size b is precisely b times faster than generating one-at-a-time). Rather, our point should be taken as qualitative and conceptual. ⋄

The submatrix access model

In view of the discussion in the previous subsection, we see the entry access model described in Section 3.1 may be too simplistic an abstraction to describe the behavior of kernel matrix algorithms. A better abstraction, introduced in [ETW25], is the *submatrix access model*:

> **Submatrix access model.** We are given a matrix $\boldsymbol{B} \in \mathbb{K}^{m \times n}$ that may be efficiently accessed by submatrices $\boldsymbol{B}(\mathsf{S}, \mathsf{T})$ of small size $|\mathsf{S}| \cdot |\mathsf{T}| \ll mn$. The access is most efficient when both S and T have size $\gg 1$.

The goal of this section will be to develop versions of RPCHOLESKY that are efficient in the submatrix access model.

Matrix–matrix arithmetic

Memory transfers also help explain why block matrix operations like matrix–matrix multiplication are fast on modern computers. The core idea is that, with careful implementation to manage data movement, computing a matrix–matrix product $\boldsymbol{BC}$ all at once requires fewer memory transfers than computing the product one column $\boldsymbol{Bc}_i$ at a time.

Achieving high performance for block matrix arithmetic on modern computer architectures demands significant programming effort. Fortunately, much of this work has already been done, with hardware vendors providing optimized implementations of matrix multiplication as part of the Level-3 Basic Linear Algebra Subprograms (BLAS3). Modern linear algebra libraries such as LAPACK are built to exploit these routines, computing matrix factorizations (LU, Cholesky, QR, etc.) through sequences of matrix multiplications implemented via BLAS3. For the algorithm designer or numerical programmer, the takeaway is clear: To maximize performance, it is advantageous to reorganize computations to rely on matrix–matrix operations rather than on matrix–vector or vector–vector operations.

8.2 Algorithm 1: Block RPCHOLESKY

Given currently selected pivots $s_1, \ldots, s_i$, the standard RPCHOLESKY algorithm draws a single next pivot at each iteration using the sampling rule

$$s_{i+1} \sim \mathbf{diag}(\boldsymbol{A}^{(i)}),$$

and it computes the next column $\boldsymbol{f}_{i+1}$ of the factor matrix $\boldsymbol{F}$ by the formula

$$\boldsymbol{f}_{i+1} := \frac{\boldsymbol{a}_{s_{i+1}} - \boldsymbol{F}(:, 1:i)\boldsymbol{F}(s_{i+1}, 1:i)^*}{(a_{s_{i+1}s_{i+1}} - \boldsymbol{F}(s_{i+1}, 1:i)\boldsymbol{F}(s_{i+1}, 1:i)^*)^{1/2}}. \tag{8.2}$$

(Recall the definition of $\boldsymbol{A}^{(i)}$ in (8.1).) Each iteration exposes a single column $\boldsymbol{a}_{s_i}$ of the matrix $\boldsymbol{A}$ and evaluates the matrix–vector product $\boldsymbol{F}(:, 1:i)\boldsymbol{F}(s_{i+1}, 1:i)^*$.

To create a block variant of this algorithm, we can draw multiple random pivots at once

$$s_{i+1}, \ldots, s_{i+b} \stackrel{\text{iid}}{\sim} \mathbf{diag}(\boldsymbol{A}^{(i)}), \tag{8.3a}$$

where $b \geq 1$ is a user-specified block size. Setting $\mathsf{S}' := \{s_{i+1}, \ldots, s_{i+b}\}$, a block of b new columns of the factor matrix can be generating using the formula

$$\boldsymbol{F}(:, i+1:i+b) := (\boldsymbol{A}(:, \mathsf{S}') - \boldsymbol{F}(:, 1:i)\boldsymbol{F}(\mathsf{S}', 1:i)^*)\boldsymbol{R}^{-1}, \tag{8.3b}$$

Program 8.1 `block_rpcholesky.m`. Block RPCHOLESKY algorithm for psd low-rank approximation and column subset selection. Subroutine `sqrownorms` is defined in Program F.3.

```matlab
function [F,S] = block_rpcholesky(Acol,d,k,b)
% Input:  Function Acol for producing columns Acol(I) = A(:,I) of A,
%         diagonal d of A, rank k, block size b
% Output: Factor F defining a rank-k approximation Ahat = F*F', pivot
%         set S

n = length(d);                  % Matrix size
F = zeros(n,k);                 % To store output
S = zeros(k,1);                 % To store pivots
shift = 4*max(d)*eps;           % Shift to ensure positive definiteness
i = 0;                          % Index to store current position
while i < k
    b = min(b,k-i);                             % At most k pivots
    % Random sample using current diagonal as sampling weights
    Snew = datasample(1:n,b,"Weights",d,"Replace",false);
    S(i+1:i+b) = Snew;                          % Update pivots
    G = Acol(Snew) - F(:,1:i)*F(Snew,1:i)'; % Columns of residual
    R = chol(G(Snew,:) + shift*eye(b));      % Shift for stability
    F(:,i+1:i+b) = G / R;                       % Update factor
    d = d - sqrownorms(F(:,i+1:i+b));        % Update diagonal
    d = max(d,0); % Ensure nonnegative diagonal in floating point
    i = i + b;                                  % Update index
end

end
```

where

$$R^*R = A(\mathsf{S}', \mathsf{S}') - F(\mathsf{S}', 1:i)F(\mathsf{S}', 1:i)^* = A^{(i)}(\mathsf{S}', \mathsf{S}'). \qquad (8.3c)$$

is an upper-triangular Cholesky factorization. Equation (8.3b) is the block Cholesky analog of (8.2). The update steps (8.3) generate b columns of the kernel matrix, and they employ block matrix–matrix operations to update the factor matrix F. We call the procedure (8.3) the *block RPCHOLESKY algorithm*. Code is provided in Program 8.1.

The redundant pivot problem. The block RPCHOLESKY algorithm is simple and performant. However, it can suffer from the *redundant pivot problem*: The selected pivots $\mathsf{S}' = \{s_{i+1}, \ldots, s_{i+b}\}$ can contain similar information to each other, resulting in a submatrix $A^{(i)}(\mathsf{S}', \mathsf{S}')$ that is ill-conditioned. Redundant pivots can hurt the performance of the algorithm in two ways:

1. **Cholesky decomposition failure.** On some examples, the matrix $A^{(i)}(\mathsf{S}', \mathsf{S}')$ can be (numerically) rank-deficient, causing the Cholesky decomposition (8.3c) to fail.

2. **Lower quality approximation/pivot set.** For a fixed rank k, a low-rank approximation with redundant pivots is less accurate than an approximation where each pivot is distinct. Additionally, for subset selection problems, the computational task is to select a *diverse* set of representatives for a data set; selecting redundant pivots is contrary to this goal.

The first issue can be addressed by adding a shift on the order of the machine precision to the matrix $A^{(i)}(\mathsf{S}', \mathsf{S}')$ in (8.3c) or applying RPCHOLESKY to a shifted version $A + \mu\mathbf{I}$ of the matrix. My coauthors and I have experimented with different values for the shift parameter across different papers and different versions of our software, with the goal of finding the minimum possible shift that guarantees convergence of the algorithm. Our current recommendation is to use a shift of $4\max\{a_{ii}\}u$, where u is the unit roundoff ($u \approx 10^{-16}$ in double precision).

The second issue may or may not be serious depending on the application. We emphasize that redundant pivots do not *hurt* the quality of the approximation (at least in exact arithmetic); by Proposition 2.9(c), the quality of a low-rank approximation $A\langle\mathsf{S} \cup \mathsf{S}_{\text{redundant}}\rangle$ with redundant pivots is never worse than the low-rank approximation $A\langle\mathsf{S}\rangle$ without redundant pivots. Rather, each redundant pivot is a wasted opportunity to find a better pivot that better represents the data set. The next two sections outline improved versions of block RPCHOLESKY that *subsample* the set of iid pivots (8.3a) to remove redundant pivots.

Analysis. The block RPCHOLESKY bound satisfies the following guarantee, which we take from [ETW25, Thm. 4.1].

Theorem 8.2 (Block randomly pivoted Cholesky). *Let $r \geq 1$ be an integer, $\varepsilon > 0$ be a real number, and $A \in \mathbb{K}^{n \times n}$ be a psd matrix. Introduce the relative error of the best rank-r approximation:*

$$\eta := \operatorname{tr}(A - [\![A]\!]_r)/\operatorname{tr}(A).$$

Fix a block size b that divides the rank k. Block randomly pivoted Cholesky produces an (r, ε)-approximation ((3.2)) provided

$$k \geq \frac{r}{\varepsilon} + (r + b)\log\left(\frac{1}{\varepsilon\eta}\right). \tag{8.4}$$

The guarantee (8.4) for block RPCHOLESKY is very similar to the guarantee (4.1) for standard RPCHOLESKY; the only difference between the bounds is that the $(r + b)$ factor in the block RPCHOLESKY bound is replaced by r in the standard RPCHOLESKY bound. The implication is that the *worst-case guarantees* for standard and block RPCHOLESKY are similar for any block size $1 \leq b \leq O(r)$. However, while the theoretical bounds for standard and block RPCHOLESKY are different, the performance of these methods differs greatly in practice. On some instances, standard RPCHOLESKY produce far smaller errors than block RPCHOLESKY. (On the other hand, block RPCHOLESKY can be much faster than standard RPCHOLESKY, so it can accommodate a larger value of k given a fixed runtime budget.)

We refer the interested reader to [ETW25, Thm. 4.1] for a proof of this result, which proceeds by a comparison with the accelerated RPCHOLESKY algorithm (Section 8.4). See also the discussion around [ETW25, Thm. 4.3] for a discussion of the error bounds for block RPCHOLESKY that can be inferred from the earlier works of Deshpande, Rademacher, Vempala, and Wang [DRVW06].

8.3 Algorithm 2: RBRP Cholesky

The first approach to improving on block RPCHOLESKY is *robust blockwise random pivoting* (RBRP), which was proposed by Dong, Chen, Martinsson, and Pearce [DCMP24]. This approach was developed concurrently with and released prior to the accelerated RPCHOLESKY algorithm, which we describe in the next section. The RBRP strategy was originally developed for use with the randomly pivoted QR algorithm (Chapter 9), but the idea works equally well when combined with RPCHOLESKY. We describe the RPCHOLESKY variant here.

The RBRP algorithm is delightfully simple. Just like ordinary block RPCHOLESKY, we begin by drawing a set of iid random pivots

$$\mathsf{S}' = \{s'_{i+1}, \ldots, s'_{i+b}\} \quad \text{with} \quad s'_1, \ldots, s'_b \overset{\text{iid}}{\sim} \mathbf{diag}(\boldsymbol{A}^{(i)}).$$

Then, form the submatrix

$$\boldsymbol{H} \coloneqq \boldsymbol{A}^{(i)}(\mathsf{S}', \mathsf{S}') = \boldsymbol{A}(\mathsf{S}', \mathsf{S}') - \boldsymbol{F}(\mathsf{S}', 1:i)\boldsymbol{F}(\mathsf{S}', 1:i)^*.$$

To filter any redundant pivots, we apply a step Dong et al. call *robust blockwise filtering*, which runs Cholesky with greedy pivoting (Program 3.2) on $\boldsymbol{H}$ until the trace has been reduced by a factor $\tau \in (0, 1)$. (Dong et al. suggest $\tau = 1/b$.) Letting $\mathsf{T} \subseteq \mathsf{S}'$ denote the set of pivots selected by the greedy method, we induct

Program 8.2 `robust_block_filter.m`. Implementation of robust blockwise filtering, a subroutine for the RBRP Cholesky algorithm (Program 8.3).

```
function [T,L] = robust_block_filter(H,tau,lmax)
% Input:  Psd matrix H, tolerance tau, maximum number of pivots to
%         accept bmax
% Output: Set of pivots T, Cholesky factor L = chol(H(T,T), "lower")

b = size(H,1);           % Matrix dimension
F = zeros(size(H));      % Store Cholesky factor
T = zeros(b,1);          % Store pivots
d = diag(H);             % Diagonal of H
orig_trace = sum(d);     % Original trace of H

for i = 1:min(b,lmax)  % Don't exceed lmax pivots
    [~,T(i)] = max(d);                                % Largest diag entry
    hi = H(:,T(i)) - F(:,1:i-1)*F(T(i),1:i-1)';       % ith col of H-F*F'
    F(:,i) = hi / sqrt(hi(T(i)));                     % Rescale
    d = d - abs(F(:,i)).^2;                           % Update diagonal
    if sum(d) <= tau * orig_trace; break; end         % Terminate?
end

T = T(1:i);              % Extract pivots
L = F(T,1:length(T));    % Extract Cholesky factor

end
```

the subselected pivots as new pivots:

$$s_{i+1} = s'_{i_1}, s_{i+2} = s'_{i_2}, \ldots, s_{i+\ell} := s_{i'_\ell} \quad \text{where } \mathsf{T} = \{s'_{i_1}, \ldots, s'_{i_\ell}\}.$$

Observe that this procedure inducts a random number of pivots $1 \leq \ell := |\mathsf{T}| \leq b$ at each step. Code for the robust blockwise filtering and RBRP Cholesky procedures appear in Programs 8.2 and 8.3. At present, theoretical analysis for RBRP Cholesky is unavailable.

8.4 Algorithm 3: Accelerated RPCHOLESKY

Accelerated RPCHOLESKY is another blocked variant of RPCHOLESKY that solves the redundant pivot problem [ETW25]. It is conceptually similar to the RBRP Cholesky algorithm but uses rejection sampling in place of robust blockwise filtering. Compared to block RPCHOLESKY and RBRP Cholesky, accelerated RPCHOLESKY has the virtue that it produces *the same random output as standard RPCHOLESKY* (Program 4.1). Thus, it inherits RPCHOLESKY's theoretical guarantees (Section 4.3), and it can be used for applications where the precise distribution of the RPCHOLESKY

Program 8.3 `rbrp_chol.m`. RBRP Cholesky algorithm for psd low-rank approximation and column subset selection. Subroutines `robust_block_filter` and `sqrownorms` ared defined in Programs 8.2 and F.3.

```
function [F,S] = rbrp_chol(Acol,Asub,d,k,b)
% Input:  Function Acol for producing columns Acol(I) = A(:,I) of A,
%         function Asub for producing submatrices Asub(I) = A(I,I) of
%         A, diagonal d of A, rank k, block size b
% Output: Factor F defining a rank-k approximation Ahat = F*F', pivot
%         set S

n = length(d);                  % Matrix size
F = zeros(n,k);                 % To store output
S = zeros(k,1);                 % To store pivots
i = 0;                          % Index to store current position
while i < k
    % Random sample using current diagonal as sampling weights
    Sp = datasample(1:n,b,"Weights",d,"Replace",false);
    H = Asub(Sp) - F(Sp,1:i)*F(Sp,1:i)';    % Residual submatrix
    [T,L] = robust_block_filter(H,1/b,k-i); % Block filtering
    T = Sp(T);                              % Get selected pivots
    l = length(T);                          % Number of pivots
    S(i+1:i+l) = T;                         % Update pivots
    G = Acol(T) - F(:,1:i)*F(T,1:i)';       % Columns of residual
    F(:,i+1:i+l) = G / L';                  % Update factor
    d = d - sqrownorms(F(:,i+1:i+l));       % Update diagonal
    d = max(d,0); % Ensure nonnegative diagonal in floating point
    i = i + l;                              % Update index
end

end
```

pivots is important, such as fixed-size DPP sampling (Section 4.5). We will discuss the relative strengths of the three algorithms in Sections 8.5 and 8.6.

Recall from Section 7.3 that rejection sampling allows one to sample from a (potentially complicated) *target distribution* using samples from a simpler *proposal distribution*. In that section, we used the diagonal of a kernel function κ as a proposal distribution, with the RPCHOLESKY sampling distribution as the target. Here, we will use the same principles in a somewhat different way.

Suppose we have sampled an initial batch of pivots $s_1, \ldots, s_i$, giving rise to the residual matrix $A^{(i)}$. We are interested in sampling a new set of pivots $s_{i+1}, \ldots, s_{i+\ell}$,

Program 8.4 `acc_rpcholesky.m`.   Accelerated RPCHOLESKY method for psd low-rank approximation and column subset selection.   Subroutine `rejection_sample_submatrix` is defined in Program 8.5.

```matlab
function [F,S] = acc_rpcholesky(Acol,Asub,d,k,b)
% Input:  Function Acol for producing columns Acol(I) = A(:,I) of A,
%         function Asub for producing submatrices Asub(I) = A(I,I) of
%         A, diagonal d of A, rank k, block size b
% Output: Factor F defining a rank-k approximation Ahat = F*F', pivot
%         set S

n = length(d);                  % Matrix size
F = zeros(n,k);                 % To store output
S = zeros(k,1);                 % To store pivots
i = 0;                          % Index to store current position
while i < k
    % Random sample using current diagonal as sampling weights
    Sp = datasample(1:n,b,"Weights",d,"Replace",true);
    H = Asub(Sp) - F(Sp,1:i)*F(Sp,1:i)';    % Residual submatrix
    [T,L] = rejection_sample_submatrix(H,diag(H),k-i);
    T = Sp(T);                           % Get selected pivots
    l = length(T);                       % Number of pivots
    S(i+1:i+l) = T;                      % Update pivots
    G = Acol(T) - F(:,1:i)*F(T,1:i)';    % Columns of residual
    F(:,i+1:i+l) = G / L';               % Update factor
    d = d - sqrownorms(F(:,i+1:i+l));    % Update diagonal
    d = max(d,0); % Ensure nonnegative diagonal in floating point
    i = i + l;                           % Update index
end

end
```

each of which has distribution

$$s_{i+j+1} \sim \mathbf{diag}(A^{(i+j)}) \quad \text{for } j = 0, \ldots, \ell - 1.$$

We shall sample these distributions using rejection sampling, with $\mathbf{diag}(A^{(i)})$ serving as the sampling distribution.

More precisely, we do the following. As in the other algorithms, we draw a block of proposal pivots $S' = \{s'_1, \ldots, s'_b\}$ iid from the diagonal of the residual matrix:

$$s'_1, \ldots, s'_b \overset{\text{iid}}{\sim} \mathbf{diag}(A^{(i)}).$$

We emphasize that the sampling must be performed *with replacement*. (The other algorithms can be implemented with or without replacement.) We now use this

block of pivots to perform b proposals of rejection sampling. Beginning with $\ell = 0$, we perform the following step for each $j = 1, \ldots, b$:

With probability $\dfrac{A^{(i+\ell)}(s'_j, s'_j)}{A^{(i)}(s'_j, s'_j)}$, *accept* by setting $\ell \leftarrow \ell + 1$ and $s_{i+\ell} \leftarrow s'_j$.

The output of this rejection sampling loop is a collection of new pivots $s_{i+1}, \ldots, s_{i+\ell}$ of random size $1 \le \ell \le b$. (We have $\ell \ge 1$ since the first pivot is always accepted.) Having selected new pivots, we then generate columns $F(:, i + 1 : i + \ell)$ using the block update formula (8.3b). In the next round, we use the diagonal of $A^{(i+\ell)}$ as the new proposal distribution for selecting new pivots. The accelerated RPCHOLESKY method is shown in Program 8.4.

The accelerated RPCHOLESKY method uses rejection sampling to simulate the performance of the original RPCHOLESKY method while taking advantage of blockwise matrix operations. We emphasize that the output of accelerated RPCHOLESKY has the *same random distribution* as the output of the original RPCHOLESKY algorithm.

REJECTIONSAMPLESUBMATRIX subroutine. For computational efficiency and extensibility, the rejection sampling steps in the accelerated RPCHOLESKY program are encapsulated in a subroutine called REJECTIONSAMPLESUBMATRIX, shown in Program 8.5. This subroutine takes as input a submatrix $H = A^{(i)}(S', S') \in \mathbb{K}^{S' \times S'}$ and proposal distribution subvector $u \ge \mathbf{diag}(H) \in \mathbb{R}_+^{S'}$. The vector u stores the entries of the proposal distribution for the proposed pivots S'. In accelerated RPCHOLESKY, the proposal weights are $\mathbf{diag}(A^{(i)}(S', S'))$. One can also use REJECTIONSAMPLESUBMATRIX to develop a block version of the infinite-dimensional RPCHOLESKY sampler (Program 7.2), in which case the proposal distribution subvector is $u = \mathbf{diag}(\kappa(S', S'))$.

This subroutine works by performing a Cholesky decomposition of H in place, with a probabilistic decision made at each step whether to eliminate entry j or ignore it entirely. For $j = 1, \ldots, b$, do the following:

1. **Accept/reject.** With probability $H(s'_j, s'_j)/u(s'_j)$ accept and induct s'_j as a new pivot $\mathsf{T} \leftarrow \mathsf{T} \cup \{s'_j\}$ and go to step 2. Otherwise, reject and skip step 2.

2. **Update and eliminate.** Perform a step of Cholesky decomposition to eliminate s'_j. Specifically, set

$$H(\{s'_j, \ldots, s'_b\}, s'_j) \leftarrow H(\{s'_j, \ldots, s'_b\}, s'_j)/H(s'_j, s'_j)^{1/2},$$

Program 8.5 `rejection_sample_submatrix.m`. Sample a set of RPCholesky pivots using rejection sampling. This subroutine is used in Program 8.4.

```
function [T,L] = rejection_sample_submatrix(H,u,lmax)
% Input:  Psd matrix H and proposal distribution u
% Output: Indices T of selected pivots, Cholesky factor L
%         = chol(H(T,T), "lower")

b = size(H,1);                          % Matrix dimension
T = zeros(b,1);                         % Buffer for accepted pivots
num_accepts = 0;                        % Counter for acceptances

for j = 1:b
    if u(j) * rand() > H(j,j)           % Rejection sampling
        continue                        % If reject, continue
    end
    num_accepts = num_accepts + 1;      % Increment counter
    T(num_accepts) = j;                 % Accept i as pivot
    H(j:b,j) = H(j:b,j) / sqrt(H(j,j)); % Overwrite H with Cholesky
    % Update residual (Schur complement)
    H(j+1:b,j+1:b) = H(j+1:b,j+1:b) - H(j+1:b,j)*H(j+1:b,j)';
    if num_accepts == lmax; break; end  % At most lmax pivots
end

T = T(1:num_accepts);                   % Fix buffer size
L = tril(H(T,T));                       % Extract Cholesky factor

end
```

then update

$$\boldsymbol{H}(\mathsf{S}'_{j+1:b}, \mathsf{S}'_{j+1:b}) \leftarrow \boldsymbol{H}(\mathsf{S}'_{j+1:b}, \mathsf{S}'_{j+1:b}) - \boldsymbol{H}(\mathsf{S}'_{j+1:b}, s_j)\boldsymbol{H}(\mathsf{S}'_{j+1:b}, s_j)^*$$

with $\mathsf{S}'_{j+1:b} := \{s'_{j+1}, \ldots, s'_b\}$.

This procedure terminates with T storing the set of accepted pivots and the lower triangular portion $\boldsymbol{L}$ of $\boldsymbol{H}(\mathsf{T}, \mathsf{T})$ containing a Cholesky factorization of the T-submatrix of the (unmodified) input matrix $\boldsymbol{H}(\mathsf{T}, \mathsf{T})$. Code is provided in Program 8.5.

Comparison of REJECTIONRPCHOLESKY and accelerated RPCHOLESKY. We now have two different strategies for implementing RPCHOLESKY using rejection sampling, REJECTIONRPCHOLESKY and accelerated RPCHOLESKY. A comparison is warranted.

We originally introduced REJECTIONRPCHOLESKY for sampling from an infinite space X endowed with a positive-definite kernel function κ, but the algorithm also

works for implementing RPCHOLESKY on a finite psd matrix A. REJECTIONRPC-HOLESKY uses the same proposal distribution $\mathbf{diag}(A)$ throughout the algorithm, and it forms only the pivot set S, not the factor matrix F. Assuming $\mathbf{diag}(A) = \mathbf{1}$ is the vector of ones, the REJECTIONSAMPLESUBMATRIX runs in roughly

$$O(k^3/\eta_k) \text{ operations,}$$

where $\eta_k = \text{tr}(A - [\![A]\!]_{k-1})/\text{tr}(A)$ is the relative error of the best rank-$(k-1)$ approximation. (Here, we assume that a random integer between $1 \le i \le n$ can be generated in $O(1)$ operations, which is true in the WordRAM model of computation.) The runtime of the algorithm is *independent* of the dimension n, but it suffers from the *curse of smoothness*, with the algorithm slowing when the eigenvalues of A decay rapidly. The reason for the curse of smoothness is that the proposal distribution $\mathbf{diag}(A)$ remains static throughout the algorithm, and the probability of each proposal being accepted becomes smaller and smaller as the diagonal $\mathbf{diag}(A^{(i)})$ of the residual shrinks. An additional weakness of REJECTIONSAMPLESUBMATRIX is that it does not generate the factor matrix F, although it can be assembled in $O(k^2 n)$ operations if desired.

The accelerated RPCHOLESKY defeats the curse of smoothness by updating the proposal distribution after every b proposals. The cost of the algorithm is at most

$$O(k^2 n + kb^2) \text{ operations and } (k+1)n + kb^2 \text{ entry accesses,}$$

only slightly more than standard RPCHOLESKY provided the block size b is not large. Moreover, accelerated RPCHOLESKY is designed so that most of the operations are performed in a block-wise fashion, making it 5× to 40× faster than standard RPCHOLESKY in practice.

Ultimately, REJECTIONSAMPLESUBMATRIX (or a block version of it) is a useful algorithm for certain scenarios, namely for applying RPCHOLESKY to matrices with slow spectral decay or for extremely large or infinite-dimensional problems. Accelerated RPCHOLESKY, by contrast, is excellent for *general-purpose use*, and I recommend the algorithm for deployment in practice.

Picking the block size. An advantage of the accelerated RPCHOLESKY algorithm over the block RPCHOLESKY algorithm is that it can accommodate a very large block size while maintaining approximation quality, as the rejection sampling step helps filter any redundant pivots. When the number of pivots k is set by the user in

advance, I recommend $b = k/10$ or even $b = k/3$ as a good default value for the block size.

One can also set the block size automatically to avoid additional algorithmic parameters. Here is one procedure I have used in my code. Initialize the block size b at some default value. Each round of the algorithm, time both the rejection step (i.e., sampling proposals S', forming the submatrix $\boldsymbol{A}^{(i)}(\mathsf{S}', \mathsf{S}')$, and running REJECTIONSAMPLESUBMATRIX) and the processing step (reading columns $\boldsymbol{A}(:, \mathsf{T})$ and updating $\boldsymbol{F}$). Let t_{r} and t_{p} be the respective runtimes. We may assume for simplicity that the runtimes of these steps satisfy the proportionality relations

$$t_{\mathrm{r}} \approx c_{\mathrm{r}} b^2 \quad \text{and} \quad t_{\mathrm{p}} \approx c_{\mathrm{p}} b. \tag{8.5}$$

for appropriate constants c_{r} and c_{p}. It is reasonable want the rejection step to comprise a modest fraction of the total runtime, say, 20%. Setting $t_{\mathrm{r}} = 0.25 t_{\mathrm{p}}$ and solving (8.5) suggests choosing the next block size as $b = t_{\mathrm{p}}/(4t_{\mathrm{r}})$. To make this procedure robust, we make sure that the block size does not change too much from iteration to iteration, and weset a maximum block size $b_{\max}$ (e.g., $b_{\max} = k/3$ if k is known in advance). The resulting update rule is

$$b \leftarrow \max \left\{ \min \left\{ \frac{t_{\mathrm{p}}}{4t_{\mathrm{r}}} \cdot b, \lceil 1.5b \rceil, b_{\max} \right\}, \left\lceil \frac{b}{3} \right\rceil \right\}.$$

Analysis. One can show that the accelerated RPCHOLESKY algorithm achieves the (r, ε)-guarantee using a limited number of *proposals*. Specifically, we have the following result [ETW25, Thm. 4.1]:

Theorem 8.3 (Accelerated randomly pivoted Cholesky)**.** *Let $r \geq 1$ be an integer, $\varepsilon > 0$ be a real number, and $\boldsymbol{A} \in \mathbb{K}^{n \times n}$ be a psd matrix. Introduce the relative error of the best rank-r approximation:*

$$\eta := \mathrm{tr}(\boldsymbol{A} - [\![\boldsymbol{A}]\!]_r)/\mathrm{tr}(\boldsymbol{A}).$$

Fix a number of rounds t and a number of proposals per round b. Accelerated randomly pivoted Cholesky produces an (r, ε)-approximation to $\boldsymbol{A}$ provided that

$$bt \geq \frac{r}{\varepsilon} + (r + b) \log \left(\frac{1}{\varepsilon \eta} \right). \tag{8.6}$$

We omit the proof, which is a more involved version of the standard RPCHOLESKY analysis (Theorem 4.1). The similarity between the block RPCHOLESKY guarantee
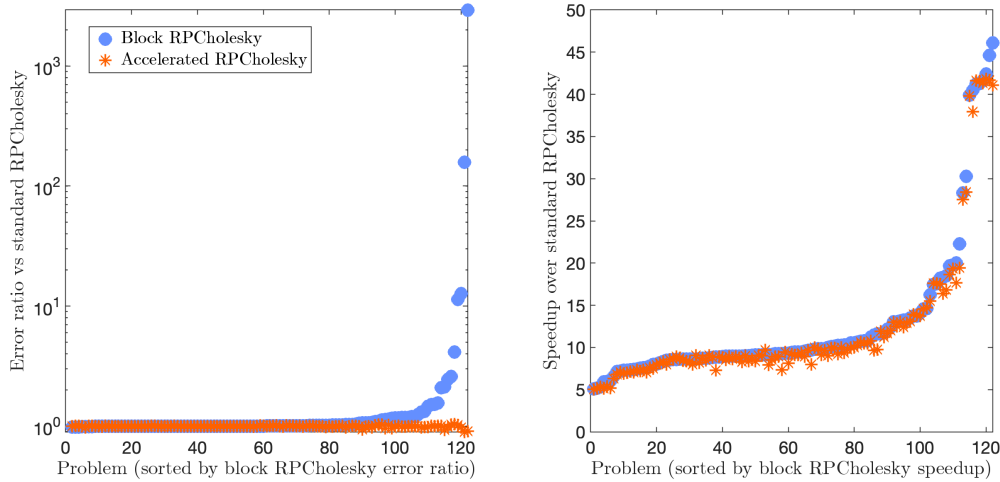
Figure 8.1: Error ratio ((8.7), *left*) and speedup over standard RPCHOLESKY (*right*) for block RPCHOLESKY and accelerated RPCHOLESKY on a testbed of 125 psd kernel matrices. The data is taken from [ETW25, Fig. 3].

(8.4) and the accelerated RPCHOLESKY guarantee (8.6) is no accident. Theorem 8.2 follows as a corollary of Theorem 8.3 under the principle that "Accepting more pivots can only help the approximation quality."

8.5 Experiments

In [ETW25, §3.1], my coauthors and I compared the error and runtime of block RPCHOLESKY and accelerated RPCHOLESKY for rank-1000 approximation of 125 test matrices with sizes between 4×10^4 and 10^5, including both synthetic and real data. I reproduce the data from those experiments here in Fig. 8.1. Standard RPCHOLESKY provides a baseline for both speed and accuracy, and Fig. 8.1 plots the speedup and error ratio

$$\text{error ratio} := \frac{\text{algorithm error}}{\text{standard RPCHOLESKY error}} \tag{8.7}$$

for each matrix in the test suite. The runtimes in this data were computed using the Python code employed in [CETW25; ETW25], not the MATLAB code provided in this thesis. The block size was set to $b = 150$.

We see that accelerated and block RPCHOLESKY achieve comparable speed to each other, with both methods achieving speedups of 5× to 40× over standard RP-CHOLESKY. These experiments demonstrate the large benefits of blocking for the speed of RPCHOLESKY-type low-rank approximation algorithms.

These results show that the accuracy of block RPCHOLESKY and accelerated RP-

	Relative trace error	Runtime (sec)
Block RPCHOLESKY	3.89e-04 ± 1.40e-04	7.00 ± 0.63
Accelerated RPCHOLESKY	4.85e-07 ± 3.60e-08	7.43 ± 1.06
RBRP Cholesky	6.62e-07 ± 1.35e-07	7.53 ± 0.80

Table 8.1: Runtime and relative error for rank-1000 approximation produced by three blocked RPCHOLESKY algorithms for the synthetic test matrix of dimension $n = 10^5$ from [ETW25, Fig. 1]. (All methods use block size $b = 120$.) The table shows the mean and standard deviation computed over 100 trials, and data is taken from [ETW25, Tab. 3].

CHOLESKY differs significantly. On all test matrices, the accelerated RPCHOLESKY method achieves the same error as standard RPCHOLESKY, which makes sense because both methods produce the same random distribution of outputs. On 100 out of 125 examples, the error of block RPCHOLESKY is similar to the other methods, with an error ratio between 1.00 and 1.16. Because of the redundant pivot problem, block RPCHOLESKY suffers significantly higher errors (up to 3000×) than standard RPCHOLESKY.

Our paper [ETW25] does not provide detailed experiments comparing RBRP Cholesky to the other blocked RPCHOLESKY algorithms, but it does compare the algorithms on a single challenging synthetic matrix. We reproduce this table as Table 8.1. We see that the performance of RBRP Cholesky and accelerated RPCHOLESKY are similar, with accelerated RPCHOLESKY being slightly more accurate and faster. (Notably, the output of accelerated RPCHOLESKY is also *less variable* than RBRP Cholesky, which is curious because the accelerated RPCHOLESKY algorithm uses "more randomness" than the RBRP Cholesky does.) The small advantage of accelerated RPCHOLESKY over RBRP Cholesky on this example pales in comparison to the large improvement in accuracy both methods have over block RPCHOLESKY.

8.6 Comparison of three algorithms

All three blocked RPCHOLESKY algorithms have merits for some use cases.

The block RPCHOLESKY is simple and fast, and it performs operations on chunks of data of *fixed* size $n \times b$. As such, block RPCHOLESKY algorithm is ideal for computing hardware like GPUs that are optimized for repetitive operations on data buffers of fixed size. Additionally, the block RPCHOLESKY algorithm may be appropriate for applications where its subpar accuracy can be compensated for by taking a larger approximation rank k.

Both accelerated RPCHOLESKY and RBRP Cholesky algorithms both yield much better low-rank approximations on tough examples than block RPCHOLESKY. At least for the CPU experiments reported in Section 8.5, the cost of using these more-accurate methods is minimal, as these methods achieve nearly the same speedup as block RPCHOLESKY does. For this reason, I would recommend accelerated either RPCHOLESKY or RBRP Cholesky as a natural choice for most applications of RPCHOLESKY. The merits of these methods over block RPCHOLESKY are particularly pronounced when used for subset selection problems, where it is important not to select redundant pivots.

The differences between RBRP Cholesky and accelerated RPCHOLESKY are fairly minor, but there are some reasons to prefer accelerated RPCHOLESKY:

1. Accelerated RPCHOLESKY produces the same random pivot distribution as standard RPCHOLESKY. As such, it inherits RPCHOLESKY's theoretical guarantees (Section 4.3), and it can be used in applications where the precise distribution of pivots is important (e.g., sampling from a projection DPP, Section 4.5).

2. In my computational experience, accelerated RPCHOLESKY sometimes produces modestly better low-rank approximations on some examples (like the one in Table 8.1).

Notwithstanding these differences, accelerated RPCHOLESKY and RBRP Cholesky tend to perform similarly in practice, and both are natural choices for deployment in software.

Chapter 9

RANDOMLY PIVOTED QR: LOW-RANK APPROXIMATION OF GENERAL MATRICES

> *But the idea of using actual columns and rows of the matrix A [for low-rank approximation] can be highly attractive. Those vectors have meaning. They are often sparse and/or nonnegative and they reflect useful properties that we wish to preserve in approximating A.*

> Gilbert Strang and Cleve Moler, *LU and CR Elimination* [SM22, §6]

So far, this thesis has focused on low-rank approximation of psd matrices. Algorithms for this task can achieve remarkable results, producing a near-optimal low-rank approximation to *any* psd matrix after reading a fraction of its entries. The algorithms we have seen accomplish this goal by forming a column Nyström approximation to a matrix A using a judiciously chosen subset of columns. Among the available algorithms for psd low-rank approximation from limited entry evaluations, RPCHOLESKY and its variants are among the fastest and most reliable.

The impossibility result Proposition 3.3 shows that producing accurate approximations to a general matrix is impossible from a small budget of entry evaluations without additional information. Therefore, to produce a low-rank approximation to a general matrix, we are generally interested in algorithms that read the full input matrix, opening up a much larger design space for algorithms. In particular, natural algorithms include the randomized SVD and its variants (Sections 2.3 and 2.4), which multiply the input matrix B by a randomized test matrix Ω (constructed independently from B).

Still, there are reasons in some applications to prefer low-rank approximations to B spanned by a subset of the columns of B. This chapter will review *randomly pivoted QR* algorithms for constructing such low-rank approximations and describe their connection to RPCHOLESKY and other methods for psd low-rank approximation.

Sources. The algorithm we call randomly pivoted QR algorithm was originally proposed as adaptive sampling by Deshpande, Rademacher, Vempala, and Wang [DRVW06; DV06]. Our treatment follows the papers

Yifan Chen, Ethan N. Epperly, Joel A. Tropp, and Robert J. Webber. "Randomly Pivoted Cholesky: Practical Approximation of a Kernel Matrix with Few Entry Evaluations". In: *Communications on Pure and Applied Mathematics* 78.5 (2025), pp. 995–1041. DOI: 10.1002/cpa.22234

and

Ethan N. Epperly, Joel A. Tropp, and Robert J. Webber. *Embrace Rejection: Kernel Matrix Approximation by Accelerated Randomly Pivoted Cholesky*. Apr. 2025. arXiv: 2410.03969v3.

These papers revisited the randomly pivoted QR algorithm, though their focus was more on RPCHOLESKY. This chapter's treatment of randomly pivoted QR is substantially expanded from those papers. It contains new research including discussion of numerical stability issues.

Outline. Section 9.1 introduces column projection approximations for approximating a matrix using a subset of their columns and discusses pivoted partial QR algorithms for computing them. Section 9.2 presents randomly pivoted QR , and Section 9.3 discuss fast implementations using rejection sampling. Related work on sketchy pivoting is discussed in Section 9.4, and numerical experiments are provided in Section 9.5. Finally, Section 9.6 concludes by discussing the history and connections between randomly pivoted QR and randomly pivoted Cholesky.

9.1 Low-rank approximation via column selection

Recall from Chapter 2 that many of the commonly used low-rank approximations for general matrices are *projection approximations*. Given a matrix $\boldsymbol{B} \in \mathbb{K}^{m \times n}$ and a test matrix $\boldsymbol{\Omega}$, the projection approximation to $\boldsymbol{B}$ is

$$\widehat{\boldsymbol{B}} \coloneqq \boldsymbol{\Pi}_{\boldsymbol{B}\boldsymbol{\Omega}} \boldsymbol{B} = \boldsymbol{Q}\boldsymbol{Q}^* \boldsymbol{B} \quad \text{where } \boldsymbol{Q} = \mathbf{Orth}(\boldsymbol{B}\boldsymbol{\Omega}).$$

Throughout this chapter, $\boldsymbol{\Pi}_F$ denotes the orthoprojector onto range($\boldsymbol{F}$).

Just as we obtained column *Nyström* approximations from choosing the test matrix $\boldsymbol{\Omega} = \mathbf{I}(:, \mathsf{S})$ to be a column submatrix of the identity matrix, we may obtain column *projection* approximations in the same way:

Definition 9.1 (Column projection approximation)**.** Let $\boldsymbol{B} \in \mathbb{K}^{m \times n}$ be a matrix and let $\mathsf{S} \subseteq \{1, \ldots, n\}$ be a set of *pivot indices*. The *column projection approximation* to $\boldsymbol{B}$ induced by S is

$$\widehat{\boldsymbol{B}} \coloneqq \boldsymbol{\Pi}_{\boldsymbol{B}(:,\mathsf{S})} \boldsymbol{B}.$$

Naturally, the column projection approximation $\mathbf{\Pi}_{\boldsymbol{B}(:,\mathsf{S})}\boldsymbol{B}$ is the projection approximation (Definition 2.2) to $\boldsymbol{B}$ with test matrix $\boldsymbol{\Omega} = \mathbf{I}(:,\mathsf{S})$. As column projection approximations are projection approximations, they enjoy the properties listed in Proposition 2.3.

Pivoted partial QR decomposition

Just as column Nyström approximations can be computed by pivoted Cholesky decompositions, column projection approximations can be computed using pivoted QR decompositions. We collect the low-rank approximation in a factorization $\widehat{\boldsymbol{B}} = \boldsymbol{Q}\boldsymbol{F}^*$, where $\boldsymbol{Q} \in \mathbb{K}^{m \times k}$ and $\boldsymbol{F} \in \mathbb{K}^{n \times k}$. Beginning from initial residual $\boldsymbol{B}^{(0)} := \boldsymbol{B}$, do the following for $i = 0, 1, 2, \ldots, k-1$:

- **Select a pivot.** Choose a *pivot index* $s_i \in \{1, \ldots, n\}$ associated with a nonzero pivot column $\boldsymbol{b}^{(i)} \neq \boldsymbol{0}$.

- **Update approximation.** Set $\boldsymbol{q}_i := \boldsymbol{b}^{(i)}_{s_i}/\|\boldsymbol{b}^{(i)}_{s_i}\|$ and $\boldsymbol{f}_i := \boldsymbol{B}^*\boldsymbol{q}_i$. Observe that $\boldsymbol{q}_i\boldsymbol{f}_i^* = \boldsymbol{q}_i\boldsymbol{q}_i^*\boldsymbol{B}^{(i)}$ is the orthogonal projection of the residual matrix $\boldsymbol{B}^{(i)}$ onto its s_ith column.

- **Update residual.** Define $\boldsymbol{B}^{(i+1)} := \boldsymbol{B}^{(i)} - \boldsymbol{q}_i\boldsymbol{f}_i^* = (\mathbf{I} - \boldsymbol{q}_i\boldsymbol{q}_i^*)\boldsymbol{B}^{(i)}$.

As written, this procedure is effectively a Gram–Schmidt orthogonalization: We successively select a pivot column and orthonormalize the remaining columns against it. *Neglecting rounding errors*, the matrix $\boldsymbol{Q}$ produced by this procedure has orthonormal columns. See Program 9.1 for an implementation.

Remark 9.2 (Numerical stability of Gram–Schmidt). In fact, the implementation of pivoted partial QR we described above and presented in Program 9.1 is actually the *modified* Gram–Schmidt algorithm. The difference between the modified and classical Gram–Schmidt algorithms in our context is that modified Gram–Schmidt orthogonalizes the entire $\boldsymbol{B}$ matrix against the selected pivot column, whereas classical Gram–Schmidt orthogonalizes only on an as-needed basis, forming columns of $\boldsymbol{B}^{(i)}$ on an as-needed basis using the formula $\boldsymbol{b}^{(i)}_s = \boldsymbol{b}_s - \sum_{j=1}^{i} \boldsymbol{q}_j\boldsymbol{q}_j^*\boldsymbol{b}_s$. The modified Gram–Schmidt method has dramatically better properties in finite precision arithmetic than classical Gram–Schmidt; see [Hig02, §19.8] for discussion and analysis. Even using this modified Gram–Schmidt procedure, the columns of the matrix $\boldsymbol{Q}$ produced by this procedure can lose orthogonality in finite-precision arithmetic. To ensure one obtains a $\boldsymbol{Q}$ matrix with numerically orthonormal columns, one can perform pivoted partial QR using Householder reflectors; see [GV13, §5.1]. ⋄

Program 9.1 `pivpartqr.m`. Pivoted partial QR decomposition based on modified Gram–Schmidt orthogonalization.

```
function [Q,F] = pivpartqr(B,S)
% Input:  Matrix B and set of pivots S
% Output: Factors Q and F defining a rank-k approximation Bhat = Q*F'

[m,n] = size(B);
k = length(S);                      % Rank = number of pivots
Q = zeros(m,k);                     % Left factor (orthonormal cols)
F = zeros(n,k);                     % Right factor

for i = 1:k
    s = S(i);                       % Get selected pivot
    Q(:,i) = B(:,s) / norm(B(:,s)); % Normalize column
    F(:,i) = B' * Q(:,i);           % Column of factor matrix
    B = B - Q(:,i) * F(:,i)';       % Modified Gram-Schmidt
end

end
```

The pivoted partial QR algorithm computes a column projection approximation:

Proposition 9.3 (Pivoted partial QR computes a column projection approximation). *Pivoted partial QR decomposition (Program 9.1) with input matrix $\boldsymbol{B}$ and pivot set S computes a column projection approximation $\boldsymbol{\Pi}_{\boldsymbol{B}(:,\mathsf{S})}\boldsymbol{B} = \boldsymbol{Q}\boldsymbol{F}^*$.*

Two factorizations of a column projection approximation

In applications, there are two common ways of representing a column projection approximation in factored form. The first way is a pivoted partial QR decomposition:

$$\boldsymbol{\Pi}_{\boldsymbol{B}(:,\mathsf{S})}\boldsymbol{B} = \boldsymbol{Q}\boldsymbol{F}^* \quad \text{for } \boldsymbol{Q} = \mathbf{Orth}(\boldsymbol{B}(:,\mathsf{S})) \text{ and } \boldsymbol{F}^* = \boldsymbol{Q}^*\boldsymbol{B}.$$

The pivoted partial QR algorithm (Program 9.1) returns this factorization.

Another convenient way of representing the projection approximation is the *interpolative decomposition* (ID, [MT20, §13]), which takes the form

$$\boldsymbol{\Pi}_{\boldsymbol{B}(:,\mathsf{S})}\boldsymbol{B} = \boldsymbol{B}(:,\mathsf{S})\boldsymbol{W}^* \quad \text{where } \boldsymbol{W}^* = \boldsymbol{B}(:,\mathsf{S})^\dagger\boldsymbol{B}. \tag{9.1}$$

The *interpolation matrix* $\boldsymbol{W}$ approximates the columns of $\boldsymbol{B}$ as linear combinations of the pivot columns $\boldsymbol{B}(:,\mathsf{S})$. The pivot columns are represented perfectly by the interpolative decomposition, and, provided $\boldsymbol{B}(:,\mathsf{S})$ is full rank, the pivot columns

of $\boldsymbol{W}^*$ are $\boldsymbol{W}^*(:, \mathsf{S}) = \mathbf{I}$. The interpolative decomposition is a key ingredient for rank-structured matrix computations [Mar11; Wil21] and tensor network algorithms [OT10; TSL24].

The pivoted partial QR decomposition, as produced by many of the methods in this chapter, can easily be converted to an interpolative decomposition. Indeed, $\boldsymbol{F}^*(:, \mathsf{S}) = \boldsymbol{F}(\mathsf{S}, :)^*$ is an upper-triangular matrix, so

$$\boldsymbol{\Pi}_{\boldsymbol{B}(:,\mathsf{S})}\boldsymbol{B} = \boldsymbol{Q}\boldsymbol{F}^* = \boldsymbol{Q}\boldsymbol{F}(\mathsf{S}, :)^* \boldsymbol{F}(\mathsf{S}, :)^{-*}\boldsymbol{F}^* = \boldsymbol{B}(:, \mathsf{S})[\boldsymbol{F}(\mathsf{S}, :)^{-*}\boldsymbol{F}^*].$$

Ergo, the interpolation matrix is $\boldsymbol{W} = \boldsymbol{F}\boldsymbol{F}(\mathsf{S}, :)^{-1}$, which can be computed in $O(nk^2)$ operations using triangular solves.

To facilitate discussion of other algorithms later in this chapter, let us also introduce the notion of an *inexact* interpolative decomposition.

Definition 9.4 (Exact and inexact interpolative decomposition)**.** An *exact* interpolative decomposition is a decomposition of the form (9.1). An *inexact interpolative decomposition* is any low-rank approximation of the form

$$\widehat{\boldsymbol{B}} = \boldsymbol{B}(:, \mathsf{S})\widehat{\boldsymbol{W}}^*.$$

We say a randomized algorithm for computing an inexact ID is *near-optimal* if it is guaranteed to produce an output satisfying

$$\|\boldsymbol{B} - \boldsymbol{B}(:, \mathsf{S})\widehat{\boldsymbol{W}}^*\|_{\mathrm{F}}^2 \leq \mathrm{const} \cdot \|\boldsymbol{B} - \boldsymbol{\Pi}_{\boldsymbol{B}(:,\mathsf{S})}\boldsymbol{B}\|_{\mathrm{F}}^2$$

with 90% probability for *any* matrix $\boldsymbol{B}$ and pivot set S.

The Gram correspondence: Equivalence between Cholesky and QR

The Gram correspondence (Theorem 2.12) establishes a link between column projection approximation and column Nyström approximation and, consequently, a link between the pivoted partial QR and Cholesky algorithms. The general Gram correspondence theorem (Theorem 2.12) yields the following corollary:

Corollary 9.5 (QR and Cholesky)**.** *Let $\boldsymbol{A}$ be a psd matrix and let $\boldsymbol{B}$ be any Gram square root of $\boldsymbol{A}$ (that is, $\boldsymbol{A} = \boldsymbol{B}^*\boldsymbol{B}$). Let $\boldsymbol{Q}$ and $\boldsymbol{F}$ be factors produced by pivoted partial QR applied to $\boldsymbol{B}$ with pivot set S. The following conclusions hold.*

(a) ***Same factor.*** *The pivoted partial Cholesky on $\boldsymbol{A}$ with pivot set S produces the factor matrix $\boldsymbol{F}$, up to a possible scaling of the columns by elements $\omega \in \mathbb{K}$ of unit modulus.*

(b) ***Connection between low-rank approximations.*** *The low-rank approximation* $\widehat{B} = QF^*$ *produced by pivoted partial* QR *is a Gram square root of the approximation* $\widehat{A} = FF^*$ *produced by pivoted partial Cholesky,*

$$\widehat{A} = \widehat{B}^*\widehat{B},$$

(c) ***Same errors.*** *For any* $p \geq 1$, *the approximation errors are related*

$$\|A - \widehat{A}\|_{S_p} = \|B - \widehat{B}\|_{S_{2p}}^2.$$

Here, $\|\cdot\|_{S_p}$ *denotes the Schatten p-norm.*

(d) ***Connection between residuals.*** *At every step i, the i-step* QR *residual* $B^{(i)} := B - Q(:, 1 : i)F(:, 1 : i)^*$ *is a Gram square root of the i-step Cholesky residual* $A^{(i)} := A - F(:, 1 : i)F(:, 1 : i)^*$.

(e) ***Diagonals and squared column norms.*** *At each step i, the squared column norms of* $B^{(i)}$ *are the diagonal entries of* $A^{(i)}$.

As we shall see, this result allows us to to seamlessly convert Cholesky-based algorithms for psd low-rank approximation to QR-based algorithms for computing column projection approximations to a general matrix. We restate the transference of algorithms principle below:

> **Gram correspondence: Transference of algorithms.** Every algorithm producing a projection approximation to a general matrix should have an analogous algorithm that produces a Nyström approximation to a psd matrix.

9.2 Randomly pivoted QR

Under the transference of algorithms, the RPCHOLESKY procedure has an analog for computing projection approximations to a general matrix using QR decomposition. We will call this algorithm *randomly pivoted* QR (RPQR) [DRVW06; DV06; CETW25]; its history will be discussed later in Section 9.6.

The randomly pivoted QR algorithm is straighforward: Execute a pivoted partial QR decomposition, drawing a random pivot column at each iteration

$$s_{i+1} \sim \mathbf{scn}(B^{(i)}).$$

Program 9.2 `rpqr.m`. A modified Gram–Schmidt-based implementation of the randomly pivoted QR algorithm for computing a column projection approximation. Subroutine `sqcolnorms` is defined in Program F.4.

```
function [Q,F,S] = rpqr(B,k)
% Input:  Matrix B and rank k
% Output: Factors Q and F defining rank-k approximation Bhat = Q*F',
%         set of pivots S

[m,n] = size(B);
Q = zeros(m,k);                    % Left factor (orthonormal cols)
F = zeros(n,k);                    % Right factor
S = zeros(k,1);                    % Pivots

for i = 1:k
    % Random sample using current diagonal as sampling weights
    [~,s] = datasample(1:n,1,"Weights",sqcolnorms(B));
    S(i) = s;                      % Set pivot
    Q(:,i) = B(:,s) / norm(B(:,s)); % Normalize column
    F(:,i) = B' * Q(:,i);          % Column of factor matrix
    B = B - Q(:,i) * F(:,i)';      % Modified Gram-Schmidt
end

end
```

sampled according to the *squared* column norms of the current residual matrix $\boldsymbol{B}^{(i)}$. The squared column norm distribution is the natural analog of the diagonal sampling $s_{i+1} \sim \mathbf{diag}(\boldsymbol{A}^{(i)})$ used in RPCHOLESKY in view of Corollary 9.5(e). See Program 9.2 for an implementation.

Theoretical results

Under the Gram correspondence, theoretical results for RPCHOLESKY immediately lead to results for RPQR. In particular, we have the following:

Corollary 9.6 (Randomly pivoted QR). *Let $\boldsymbol{B} \in \mathbb{K}^{m \times n}$ be a matrix, and fix $r \geq 1$ and $\varepsilon \geq 0$. Introduce the squared relative error of the best rank-r approximation:*

$$\eta := \frac{\left\| \boldsymbol{B} - [\![\boldsymbol{B}]\!]_r \right\|_{\mathrm{F}}^2}{\|\boldsymbol{B}\|_{\mathrm{F}}^2}. \tag{9.2}$$

Randomly pivoted QR produces an approximation $\widehat{\boldsymbol{B}}$ satisfying

$$\mathbb{E}\|\boldsymbol{B} - \widehat{\boldsymbol{B}}\|_{\mathrm{F}}^2 \leq (1 + \varepsilon)\|\boldsymbol{B} - [\![\boldsymbol{B}_r]\!]\|_{\mathrm{F}}^2.$$

provided the number of steps satisfies

$$k \geq \frac{r}{\varepsilon} + r \log\left(\frac{1}{\varepsilon\eta}\right).$$ (9.3)

More concisely, RPQR produces an $(r, \varepsilon/2, 2)$-approximation after (9.3) steps. (Recall Definition 3.6 and the Lyapunov inequality $\mathbb{E}\|\boldsymbol{B} - \widehat{\boldsymbol{B}}\|_{\mathrm{F}} \leq (\mathbb{E}\|\boldsymbol{B} - \widehat{\boldsymbol{B}}\|_{\mathrm{F}}^2)^{1/2}$.)

Implementation

We will not spend too much time on the implementation of the standard RPQR algorithm, as we mainly advocate the *accelerated* RPQR algorithm for practical computations, which will be introduced in the next section. Therefore, we describe only a few implementation details for RPQR.

The implementation of RPQR in Program 9.2 is based on a modified Gram–Schmidt procedure; see Remark 9.2 for discussion. This implementation is numerically stable enough for most use cases, but it may be worth using an implementation based on Householder reflectors if obtaining a $\boldsymbol{Q}$ matrix that has orthonormal columns up to machine accuracy is necessary.

A second potential implementation issue is hardware efficiency. As we described in the previous chapter, the fastest matrix algorithms are based on block matrix computations; the implementation of RPQR provided in Program 9.2 is inherently sequential, selecting a column and orthogonalizing the entire matrix against it at every iteration. This deficit will be addressed using rejection sampling with the accelerated RPQR algorithm, but there are other fixes that have been proposed for pivoted QR decomposition with deterministic greedy pivoting that are worth mentioning. For fully deterministic implementations, the state of the art is provided by the LAPACK routine xGEQP3. Quoting from the LAPACK manual, this routine "only updates one column and one row of the rest of the matrix (information necessary for the next pivoting phase) and delays the update of the rest of the matrix until a block of columns has been processed" [ABBB+99, §2.4.2.3]. This modification improves the efficiency of pivoted QR decompositions significantly and reorganizes computations so a significant fraction of them utilize block matrix operations. A second more recent idea is to use randomized dimensionality reduction, applying the slow sequential pivoted QR algorithm to the matrix $\boldsymbol{B}$ after it has been compressed using randomized dimensionality reduction; see [MBMD+25] for details.

Block randomly pivoted QR

For historical reasons, we also mention that there is a straightforward block implementation of RPQR, analogous to block RPCHOLESKY (Section 8.2). For most use cases, robust block random pivoting [DCMP24] or accelerated RPQR (introduced next section) provide a better alternative.

9.3   Accelerated randomly pivoted QR

The accelerated RPCHOLESKY method (introduced in Section 8.4) uses rejection sampling to simulate the performance of the original RPCHOLESKY algorithm. It is much faster than the standard RPCHOLESKY algorithm in practice due to runtime efficiencies from block-wise computations, and it produces the *same random output* as ordinary RPCHOLESKY. Under the Gram correspondence, the accelerated RPCHOLESKY algorithm has an analog, accelerated RPQR, that simulates the performance of the RPQR algorithm while running much raster due to block-matrix arithmetic.

Description of algorithm. One "round" of the accelerated RPQR algorithm may be described as follows. Suppose we have already generated pivots $s_1, \ldots, s_i$ sampled from the RPQR distribution, and, for each j, denote the residual as

$$\boldsymbol{B}^{(j)} \coloneqq \boldsymbol{B} - \boldsymbol{Q}^{(j)}(\boldsymbol{Q}^{(j)})^* \boldsymbol{B} \quad \text{for } \boldsymbol{Q}^{(j)} = \mathbf{Orth}(\boldsymbol{B}(:, \{s_1, \ldots, s_j\})).$$

We wish to generate new pivots $s_{i+1}, \ldots, s_{i+\ell}$ generated from the RPQR distribution

$$s_{i+j} \sim \mathbf{scn}(\boldsymbol{B}^{(i+j)}).$$

To do so, we fix a block size $b \geq 1$ and draw a collection $\mathsf{S}' = \{s'_1, \ldots, s'_b\}$ iid *with replacement* from the squared column norm distribution of $\boldsymbol{B}^{(i)}$

$$s'_1, \ldots, s'_b \overset{\text{iid}}{\sim} \mathbf{scn}(\boldsymbol{B}^{(i)}).$$

Then, we filter the proposals $\mathsf{T} \subseteq \mathsf{S}'$ using rejection sampling, which can be accomplished using the REJECTIONSAMPLESUBMATRIX subroutine (Program 8.5) on $\boldsymbol{H} = [\boldsymbol{B}^{(i)}(:, \mathsf{S}')]^* \boldsymbol{B}^{(i)}(:, \mathsf{S}')$ with $\boldsymbol{u} = \mathbf{diag}(\boldsymbol{H})$; see Section 8.4 for details on REJECTIONSAMPLESUBMATRIX. Once the new pivots $s_{i+1}, \ldots, s_{i+\ell}$ are inducted, we update $\boldsymbol{Q}^{(i+\ell)}$, columns of the factor matrix $\boldsymbol{F}$, and residual $\boldsymbol{B}^{(i+\ell)}$ as needed.

Below, we describe two implementations of accelerated RPQR using different types of orthogonalization. The first implementation (Program 9.3) is based on block Gram–Schmidt orthogonalization, and the second (Program 9.4) employs Householder reflectors.

Program 9.3 `acc_rpqr_bgs.m`. Block Gram–Schmidt-based implementation of accelerated RPQR algorithm for computing a column projection approximation to a general matrix. Subroutines `rejection_sample_submatrix` and `sqcolnorms` are provided in Programs 8.5 and F.4.

```matlab
function [Q,F,S] = acc_rpqr_bgs(B,k,b)
% Input:  Matrix B, rank k, block size b
% Output: Factors Q and F defining rank-k approximation Bhat = Q*F',
%         set of pivots S

[m,n] = size(B);
Q = zeros(m,k);                         % Left (orthonormal) factor
F = zeros(n,k);                         % Right factor
S = zeros(k,1);                         % Pivots
i = 0;                                  % Index to store current position
while i < k
    % Random sample using squared column norms as sampling weights
    Sp = datasample(1:n,b,"Weights",sqcolnorms(B),"Replace",true);
    H = B(:,Sp)'*B(:,Sp);                   % Gram matrix of residual
    T = rejection_sample_submatrix(H,diag(H),k-i);
    T = Sp(T);                              % Get selected pivots
    l = length(T);                          % Number of pivots
    S(i+1:i+l) = T;                         % Update pivots
    [Q(:,i+1:i+l),~] = qr(B(:,T),"econ");   % Orthonormalize selection
    F(:,i+1:i+l) = B'*Q(:,i+1:i+l);         % Update factor
    B = B - Q(:,i+1:i+l)*F(:,i+1:i+l)';     % Update residual
    B = B - Q(:,i+1:i+l)*(Q(:,i+1:i+l)'*B); % Orthogonalize twice
    i = i + l;                              % Update index
end

end
```

Implementation #1: Block Gram–Schmidt. Our first implementation of accelerated RPQR is shown in Program 9.3 and uses block Gram–Schmidt to perform orthogonalization. It is a fairly direct extension of the modified Gram–Schmidt implementation of standard RPQR in Program 9.2. Throughout the algorithm, B stores the current residual, the columns of the left orthonormal factor Q are obtained by computing a QR decomposition of selected columns of B, and the factor matrix is computed using the handy formula $F(:,i+1:i+\ell) = (B^{(i)})^*Q(:,i+1:i+\ell)$. Following best practices [GLRE05], we orthonormalize the residual matrix B against the newly orthonormalized columns $Q(:,i+1:i+\ell)$ *twice* to improve numerical stability. This implementation has worse stability properties than the Householder reflector-based implementation in Program 9.4.

Implementation #2: Householder QR. A more numerically robust implementation of accelerated RPQR can be obtained using Householder reflectors. Rather than storing the orthonormal factor $\boldsymbol{Q}$ directly, we represent $\boldsymbol{Q}$ implicitly as the first k columns of a product $\boldsymbol{H}_1 \cdots \boldsymbol{H}_k$ of *Householder reflector matrices* $\boldsymbol{H}_i$, each of which takes the form $\boldsymbol{H}_i = \mathbf{I} - 2\boldsymbol{u}_i\boldsymbol{u}_i^*$ for a unit vector $\boldsymbol{u}_i$. This representation remains storage-efficient, as we only need to store the unit vectors $\boldsymbol{u}_i$. See [GV13, §§5.1 & 5.2] for more on Householder reflectors and QR factorization.

The advantage of a Householder reflector-based implementation over a Gram–Schmidt-based implementation is that Householder QR methods are guaranteed to produce a $\boldsymbol{Q}$ matrix whose columns are orthonormal up to machine precision [Hig02, §19.3]. Gram–Schmidt-type methods do not possess such a guarantee [Hig02, §19.8].

Program 9.4 provides a Householder-reflector based implementation of accelerated RPQR. This implementation uses compact representations of the Householder QR decomposition and carefully updates a QR decomposition of the matrix $\boldsymbol{B}(:, \mathsf{S})$ throughout the algorithm, even as the pivot set S increases in size. See Appendix A for discussion.

9.4   Related work: Sketchy pivoting

Per-Gunnar Martinsson and collaborators have pioneered a different family of methods for column subset selection called *sketchy pivoting methods* [VM17; DM23b; DCMP24]. For completeness, we provide a brief introduction to the idea.

Fast random embeddings

The starting point for these methods is a type of matrix that we will call a *fast random embedding* or *fast sketching matrix*. Informally, a fast random embedding $\boldsymbol{S} \in \mathbb{K}^{m \times d}$ is a random matrix satisfying the following two properties:

- *Fix* an arbitrary k-dimensional subspace $\mathsf{U} \subseteq \mathbb{K}^m$. With high probability, $\boldsymbol{S}^*$ preserves the lengths of *all* vectors in U up to a constant factor

$$c_1\|\boldsymbol{x}\| \leq \|\boldsymbol{S}^*\boldsymbol{x}\| \leq c_2\|\boldsymbol{x}\| \quad \text{for all } \boldsymbol{x} \in \mathsf{U}. \tag{9.4}$$

  Here, the $c_1, c_2 \approx 1$ are absolute constants. The embedding dimension d should be nearly proportional to the subspace dimension, $d = O(k)$ or $d = O(k \log k)$.

Program 9.4 `acc_rpqr.m`. Householder reflector-based implementation of accelerated RPQR algorithm for computing a column projection approximation to a general matrix. Subroutines `rejection_sample_submatrix`, `hhqr`, `apply_Qt`, `get_Q`, and `sqcolnorms` are provided in Programs 8.5, A.1 to A.3 and F.4.

```matlab
function [Q,F,S] = acc_rpqr(B,k,b)
% Input:  Matrix B, rank k, block size b
% Output: Factors Q and F defining rank-k approximation Bhat = Q*F',
%         set of pivots S

[m,n] = size(B);
hous = zeros(m,k); tau = zeros(k,1);% Compact Householder QR
S = zeros(k,1);                      % Pivots
i = 0;                               % Index to store current position
while i < k
    % Random sample using squared column norms as sampling weights
    Sp = datasample(1:n,b,"Weights",sqcolnorms(B(i+1:end,:)),...
        "Replace",true);
    % Form Gram matrix of selected pivots
    Bp = B(i+1:end,Sp);          % Bottom part of B stores residual
    H = Bp'*Bp;                  % Gram matrix
    T = rejection_sample_submatrix(H,diag(H),k-i);
    T = Sp(T);                   % Get selected pivots
    l = length(T);               % Number of pivots
    S(i+1:i+l) = T;              % Update pivots
    % Update Householder QR factorization
    hous(:,i+1:i+l) = B(:,T);
    [hous(i+1:end,i+1:i+l),tau(i+1:i+l)]... % Compute Householder
        = hhqr(hous(i+1:end,i+1:i+l));      %    reflectors
    B(i+1:end,:) = apply_Qt(hous(i+1:end,i+1:i+l),... % Update
        tau(i+1:i+l),B(i+1:end,:));             %    residual
    i = i + l;
end

Q = get_Q(hous,tau);
F = B(1:k,:)';

end
```

- For any matrix $B \in \mathbb{K}^{m \times n}$, the product S^*B can be computed in at most $O(mn \log m)$ operations.

Sketching will play a major role in Part III of this thesis; see that section for formal definitions, constructions of fast random embeddings, and guidance on which type of embedding to use.

Sketching pivoting

Suppose we are interested in selecting k columns of a matrix $\boldsymbol{B}$, where the parameter k is *known to the algorithm in advance*. Using the sketchy pivoting approach, we first draw a fast random embedding of size, say $d = 2k$, and compute $\boldsymbol{S}^*\boldsymbol{B}$. Then, to select a subset of columns for $\boldsymbol{B}$, we apply some traditional column subset selection algorithm, such as a pivoted matrix decomposition, to the *sketched matrix* $\boldsymbol{S}^*\boldsymbol{B}$. The key feature of sketchy pivoting is that the traditional matrix decompositions needs only be applied to the sketched matrix $\boldsymbol{S}^*\boldsymbol{B}$, which is much smaller than the original matrix $\boldsymbol{B}$.

One major strength of sketchy pivoting is that the randomized embedding $\boldsymbol{S}^*$ has the effect of *regularizing* the matrix $\boldsymbol{B}$. Empirically, one finds that one obtains a nearly optimal set of columns even when one uses a poor column subset selection algorithm on $\boldsymbol{S}^*\boldsymbol{B}$ such as LU with partial pivoting (applied to the adjoint $(\boldsymbol{S}^*\boldsymbol{B})^*$) [DM23a]. We will abbreviate the sketchy pivoting with partial-pivoted LU scheme as SκLUPP. With an appropriate type of embedding (namely, a sparse sign embedding with constant sparsity parameter ζ; see Appendix B.6), SκLUPP requires just $O(mn + k^2 n)$ operations *to select the column set* S, faster than RPQR and its variants.

Quickly computing an inexact interpolative decomposition

To upgrade a skeleton set S computed by a sketchy pivoting method to an *exact* interpolative decomposition (in the sense of Definition 9.4) requires $O(kmn)$ operations, the same asymptotic cost as RPQR and its variants. However, a *nearly optimal inexact* ID (also as in Definition 9.4) can be obtained rapidly by applying another level of sketching. The idea is due to [DCMP24] and is called *oversampled sketchy interpolative decomposition*. The idea is simple and clever: Generate another random embedding $\boldsymbol{\Phi} \in \mathbb{K}^{m \times d}$ and solve the sketched least-squares problem

$$\widehat{\boldsymbol{W}} = \operatorname*{argmin}_{\widehat{\boldsymbol{W}}} \|\boldsymbol{\Phi}^*\boldsymbol{B} - (\boldsymbol{\Phi}^*\boldsymbol{B}(:,\mathsf{S}))\widehat{\boldsymbol{W}}^*\|_{\mathrm{F}} = [(\boldsymbol{\Phi}^*\boldsymbol{B}(:,\mathsf{S}))^{\dagger}(\boldsymbol{\Phi}^*\boldsymbol{B})]^*.$$

OSID is essentially an instantiation of the *sketch-and-solve* method for computing an approximate least-squares solution, which will be discussed in Section 21.7. Theoretical guarantees for sketch-and-solve are discussed in Section 21.7 and Appendix C, which confirm that—with appropriate parameter choices—OSID compiutes a nearly optimal inexact ID. The cost of sketchy pivoting with OSID may be as low as $O(mn + k^2 n)$ operations.

RPQR vs. sketchy pivoting

There are three main advantages of RPQR-based approaches over sketchy pivoting. First, and least importantly in my opinion, RPQR-based approaches come with theoretical guarantees of performance. At present, sketchy pivoting lacks such theoretical guarantees, thought it has been put through extensive numerical testing. Second, RPQR produces an exact ID, whereas the fast $O(mn + k^2n)$ version of SκLUPP produces an inexact ID. One can use the column set from SκLUPP to compute an exact ID, but then the total asymptotic cost of $O(kmn)$ is the same as accelerated RPQR. Finally, to be most efficient, sketchy pivoting methods require (an upper bound on) the column subset size k to be known in advance. To determine the number of columns k at runtime may require periodically regenerating the embedding matrix S^* and recomputing S^*B with a larger value of d. By contrast, RPQR methods can be run for a general number of steps k and stopped at whatever iteration one pleases.

Notwithstanding these limitations, sketchy pivoting methods are asymptotically faster for subset selection and inexact ID than RPQR methods. I regard both techniques (and adaptive randomized pivoting methods [CK24]) as effective, general methods for column subset selection and ID computation.

9.5 Experiments

Throughout this section, we use the Householder QR-based implementation of accelerated RPQR (Program 9.4) in all experiments. We also compare pivoted partial QR with greedy pivoting (often simply called *column-pivoted QR*, CPQR in the numerical literature) and SκLUPP. (Similarly to greedy pivoted Cholesky, CPQR selects the largest-norm column as pivot at every step of QR decomposition.) For SκLUPP, we compute the interpolative decomposition exactly (i.e., not with OSID) and choose the sketching matrix to be a sparse sign embedding with embedding dimension $d = 2k$; see Appendix B.6. We use the test matrix $B \in \mathbb{K}^{n \times n}$ of dimension $n = 2500$ with entries

$$b_{ij} = \frac{1}{\left\| x_i - y_j \right\|} \quad \text{for every } i, j = 1, \ldots, n. \tag{9.5}$$

Here, points $\{x_i\}$ and $\{y_j\}$ are equispaced Cartesian grids on $[0, 1] \times [0, 1]$ and $[1, 2] \times [0, 1]$, respectively. Matrices similar to this appear as discretizations of integral operators, and significant work has gone into developing ways of approximating them (e.g., by proxy point methods [YXY20]).
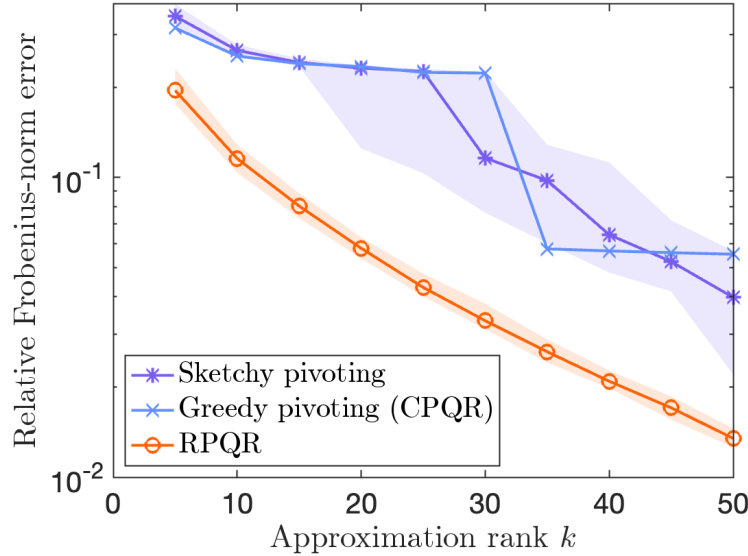
Figure 9.1: Accuracy of RPQR (orange circles), greedy pivoted QR (purple asterisks), and sketchy pivoting method (SκLUPP, blue crosses) for computing a low-rank approximation to the matrix (9.5). Lines shown median of 100 trials, and shaded regions show 10% and 90% quantiles.

Accuracy. Figure 9.1 presents an accuracy comparison of RPQR, greedy pivoted QR, and SκLUPP on this matrix. We see that RPQR consistently achieves smaller errors than the other methods on this matrix, demonstrating the virtues of incorporating randomness into pivot selection for computing a pivoted partial QR decomposition. At rank $k = 50$ on this example, RPQR is 3.0× more accurate than SκLUPP and 4.2× more accurate than greedy pivoted QR.

Speed. Figure 9.2 compares the runtime of the ordinary and accelerated RPQR algorithms (Programs 9.2 and 9.4). For accelerated RPQR, we set the block size to $b := \max\{k/2, 200\}$. We see that accelerated RPQR is 13.5× faster than standard RPQR when we reach a rank of $k = n$, at which point we have computed a full pivoted QR decomposition of the entire matrix. At this point, we can compare the performance of accelerated RPQR to MATLAB's greedy column-pivoted QR decomposition, executed as [Q,R,P] = qr(B). We omit comparisons with sketchy pivoting as it is not appropriate for computing a full pivoted QR decomposition. We see that MATLAB's built-in column-pivoted QR decomposition subroutine, calling appropriate LAPACK routines written in Fortran, is only 1.6× faster than accelerated RPQR, written in MATLAB. I consider these timing results to be promising, and it suggests that an implementation of accelerated RPQR in a low-level programming language might be faster than existing LAPACK routines for column-pivoted QR.
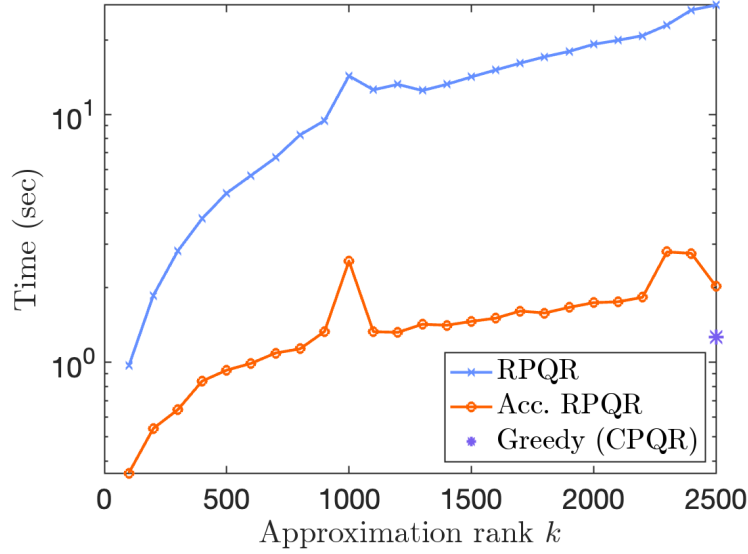
Figure 9.2: Runtime for standard RPQR (blue crosses) and accelerated RPQR (orange circles) for different approximation ranks k (single execution). The runtime for MATLAB's (full) greedy pivoted QR (i.e., `[Q,R,P] = qr(B)`) is shown as a purple asterisk.

More experiments. Many further demonstration of the virtues of random pivoting for approximation of general matrices appear in the paper [DCMP24], including comparisons of the blocked and unblocked RPQR algorithm and the RBRP QR algorithm.

9.6 RPCHOLESKY and RPQR: History

In our presentation, we introduced RPCHOLESKY first and used the transference of algorithms principle to derive the RPQR algorithm. Historically, it went the other way around. Here is the story, as best I can tell it.

Squared column norm sampling. In the late 1990s, theoretical computer scientists and researchers in adjacent fields became interested in using randomization to accelerate matrix computations, motivated by applications in data analysis [PTRV98; FKV98]. As a fast way of constructing a low-rank approximation to a large matrix, Frieze, Kannan, and Vempala suggested approximating a matrix by low-rank approximations spanned by columns $s_1, \ldots, s_k \overset{\text{iid}}{\sim} \mathbf{scn}(\boldsymbol{B})$ drawn *iid* from the squared column norm distribution [FKV98]. Their main result [FKV98, Thm. 2] establishes the guarantee

$$\mathbb{E}\big\|\boldsymbol{B} - \boldsymbol{\Pi}_{\boldsymbol{B}(:,\{s_1,\ldots,s_k\})}\boldsymbol{B}\big\|_{\mathrm{F}}^2 \leq \big\|\boldsymbol{B} - [\![\boldsymbol{B}]\!]_r\big\|_{\mathrm{F}}^2 + \frac{r}{k}\|\boldsymbol{B}\|_{\mathrm{F}}^2 \quad \text{for } s_1, \ldots, s_k \overset{\text{iid}}{\sim} \mathbf{scn}(\boldsymbol{B}).$$

Here, $r \geq 1$ is any fixed target rank. Introducing the squared relative-error of the best

rank-r approximation η from (9.2), we see that $k = O(r/\eta)$ columns are sufficient to obtain a low-rank approximation comparable to the best rank-r approximation to $\boldsymbol{B}$. This analysis is tight: *For a worst-case input matrix*, $k = \Omega(r/\eta)$ columns from the squared column norm distribution are necessary to obtain this guarantee; see [CETW25, Thm. C.3(b)].

Randomly pivoted QR introduced as "adaptive sampling". The Frieze–Kannan–Vempala algorithm suggests a natural improvement by iteration. Draw a single column (or a small batch) from the squared column norm distribution, subtract off the projection of the matrix onto these column(s), and repeat. This refinement was proposed in 2006 by Deshpande, Rademacher, Vempala, and Wang [DRVW06] and expanded on later that year by Deshpande and Vempala [DV06]; these authors called their method *adaptive sampling*. The algorithms we have called RPQR and block RPQR are essentially the same as Deshpande et al.'s adaptive sampling algorithm, up to implementation details. The papers [DRVW06; DV06] also proposed *volume sampling*, which draws a subset S of k columns of $\boldsymbol{B}$ with probability proportional to the squared volume $\det[\boldsymbol{B}(:,\mathsf{S})^*\boldsymbol{B}(:,\mathsf{S})]$; in modern language, the k-volume sampling distribution is the k-DPP distribution on the Gram matrix $\boldsymbol{B}^*\boldsymbol{B}$.

Desphande and coauthors combine their three primitives RPQR, block RPQR, and volume sampling in creative ways to prove existence results and algorithms for computing column projection approximations. Underlying these combinations are two main theoretical results, which we now summarize. The first result [DRVW06, Thm. 1.2] shows that, with a sufficiently large block size, block RPQR produces accurate low-rank approximations.

Fact 9.7 (Block RPQR: Large block size). *Let $\boldsymbol{B} \in \mathbb{K}^{m\times n}$ be a matrix, and fix target rank $r \geq 1$ and accuracy parameter $\varepsilon \in (0,1)$. Set the block size of block RPQR to be $b \geq r/\varepsilon$, and execute the procedure for t (block) steps. Then block RPQR outputs a low-rank approximation $\widehat{\boldsymbol{B}}$ satisfying*

$$\mathbb{E}\|\boldsymbol{B} - \widehat{\boldsymbol{B}}\|_{\mathrm{F}}^2 \leq (1-\varepsilon)^{-1}\|\boldsymbol{B} - [\![\boldsymbol{B}]\!]_r\|_{\mathrm{F}}^2 + \varepsilon^t\|\boldsymbol{B}\|_{\mathrm{F}}^2.$$

This result establishes the qualitative conclusion that block RPQR, *with a sufficiently large block size*, produces an approximation comparable to the best rank-r approximation after drawing $k = O(r\log(1/\eta))$ columns, where η is defined in (9.2). This result demonstrates an *exponential separation* between iid squared column norm sampling and block RPQR in the parameter η.

The second result provides a sharp bound on column projection approximations built from the volume sampling distribution and a very crude bound for the plain RPQR algorithm:

Fact 9.8 (RPQR and volume sampling). *Let $B \in \mathbb{K}^{m \times n}$ and construct k-column projection approximations $\widehat{B}_{\mathrm{vol}}$ and $\widehat{B}_{RPQR}$ using volume sampling and RPQR (with block size 1), respectively. Then*

$$\mathbb{E}\|B - \widehat{B}_{\mathrm{vol}}\|_{\mathrm{F}}^2 \leq (k+1)\big\|B - [\![B]\!]_k\big\|_{\mathrm{F}}^2, \tag{9.6}$$

$$\mathbb{E}\big\|B - \widehat{B}_{RPQR}\big\|_{\mathrm{F}}^2 \leq (k+1)!\big\|B - [\![B]\!]_k\big\|_{\mathrm{F}}^2. \tag{9.7}$$

The volume sampling bound is [DRVW06, Thm 1.3], and the RPQR bound is [DV06, Prop. 2]. The volume sampling bound (9.6) is optimal in the sense that no method can achieve a prefactor smaller than $k + 1$ (which follows by Theorem 3.12 and the Gram correspondence). By contrast, the bound (9.7) is quite weak, and seems to suggest that the unblocked versions of RPQR produce approximations of extremely low quality. Using the weak bound (9.7), Deshpande and Vempala produce algorithms with accuracy guarantees by combining r steps of RPQR with multiple rounds of block RPQR with a large block size $b = \Omega(r)$. The very weak bound (9.6) is the only analysis of RPQR with block size $b = 1$ established by Deshpande et al.

Empirical evaluation of adaptive sampling/RPQR. In the years following the publication of Deshpande and coauthors work, there were various efforts to analyze the empirical performance of adaptive sampling/RPQR-type algorithms and compare them to alternatives. There was particular interest in using variants of the adaptive sampling idea for Nyström approximation of psd matrices. A comparison of column Nyström methods by Kumar, Mohri, and Talwalkar concluded [KMT12, p. 989] that adaptive sampling "requires a full pass through [the kernel matrix] K at each iteration and is thus inefficient for large K". As solution, the papers [KMT12; WZ13] proposed cheaper, approximate versions of adaptive sampling that were more tractable. Despite its appealing properties and attempts to address the method's weaknesses, the adaptive sampling algorithm has not seen wide use for applied computation in the 2010s and early 2020s; indeed, the algorithm is not mentioned in a recent comparison of popular methods for column selection [DM23a].

Randomly pivoted Cholesky. In 2017, Musco and Woodruff considered the problem of computing a low-rank approximation to a psd matrix from a small number of

entry accesses [MW17]. Their goal was to produce a low-rank approximation $\widehat{A}$ to A that is competitive with the best rank-r approximation when measured in the *Frobenius norm*:

$$\|A - \widehat{A}\|_{\mathrm{F}}^2 \leq (1 + \varepsilon)\big\|A - [\![A]\!]_r\big\|_{\mathrm{F}}^2 \quad \text{with } \mathrm{rank}(\widehat{A}) \leq \mathrm{poly}(r, 1/\varepsilon). \tag{9.8}$$

Producing a low-rank approximation of this quality from a small number of entry accesses is a stringent requirement; see Section 11.2 for more discussion.

To motivate why psd low-rank approximation is even *possible* without reading the whole input matrix, Musco and Woodruff use what we have called the transference of algorithms principle to extend adaptive sampling to an algorithm for psd low-rank approximation. They write:

> Since $A^{1/2}A^{1/2} = A$, the entry a_{ij} is just the dot product between the ith and jth columns of $A^{1/2}$. So with A in hand, the dot products have been 'precomputed' and [adaptive sampling] yields a low-rank approximation algorithm for $A^{1/2}$ running in just $n \cdot \mathrm{poly}(k/\varepsilon)$ time. Note that, aligning with our initial intuition that reading the diagonal entries of A is necessary to avoid the $\mathrm{nnz}(A)$ time lower bound for general matrices [i.e., Proposition 3.3], the diagonal entries of A are the column norms of $A^{1/2}$, and hence their values are critical to computing the adaptive sampling probabilities.

The algorithm Musco and Woodruff describe is, in its essence, RPCHOLESKY.

The discussion of RPCHOLESKY in Musco and Woodruff's work is brief, serving to motivate the development of more sophisticated algorithms with stronger theoretical guarantees. The original paper of Musco and Woodruff does not provide pseudocode for RPCHOLESKY, document an implementation of the algorithm, or report any numerical experiments. (I have since learned from Cameron and Christopher Musco that numerical experiments were done and not published.) Also, given the brief treatment, some the subtleties of the adaptive sampling in Desphande and coauthor's work go unmentioned, such as the specific mixtures of unblocked and blocked adaptive sampling that Deshpande and Vempala use to obtain algorithms with relative error guarantees [DV06, §3.2]. To the best of my knowledge, Musco and Woodruff's work is the sole mention of the RPCHOLESKY algorithm for psd low-rank approximation in the literature prior to our work. (As mentioned in Sec-

tion 4.5, Poulson [Pou20] also uses the same computational steps as RPCHOLESKY for projection DPP sampling.)

Our paper [CETW25], originally released as a preprint in 2022, revisited the RP-CHOLESKY algorithm, reinterpreted the procedure as a partial Cholesky decomposition, and suggested the name *randomly pivoted Cholesky*. To simply the algorithm and remove the large block size limitation in Fact 9.7, we established new theoretical results ensuring that RPCHOLESKY produces near-optimal low-rank approximations with block size $b = 1$. (Our subsequent work [ETW25] extends this to general block sizes.) This paper provided numerical experiments comparing RPCHOLESKY to other algorithms for column Nyström and using RPCHOLESKY to accelerate kernel computations in scientific machine learning.

From adaptive sampling to randomly pivoted QR. After we released our paper on RPCHOLESKY, we were surprised by the amount of interest our colleagues had in the QR version of the procedure, even though it requires significantly more computational work than RPCHOLESKY ($O(kn^2)$ operations versus $O(k^2n)$ operations). This feedback, as well as input from anonymous referees during peer review, led us to devote more attention to the column projection approximations for general matrices in revisions of our RPCHOLESKY manuscript [CETW25] and in its followup [ETW25]; I have tried to add further details in this thesis.

Let me summarize our contributions and proposals for RPQR. First, we reinterpreted the adaptive sampling procedure of Deshpande and coauthors as a pivoted partial QR decomposition with a random pivoting rule, and suggested the name *randomly pivoted QR*. We hope the new name will make this algorithm more transparent to researchers in numerical analysis and scientific computing and help to differentiate between the QR and Cholesky versions of the adaptive sampling idea. Second, we proved new theoretical results [CETW25, Cor. 5.2] and [ETW25, Cor. 5.2]. These results demonstrate that RPCHOLESKY and RPQR produce approximations comparable to the best rank-*r* approximation when the block size b is set to any value between 1 and $O(r)$. In particular, these results justify the use of the algorithm without the multistep procedures and large block sizes used in Deshpande et al.'s existing work. Third, we developed accelerated block implementations of RPQR (Section 9.3). Finally, in this thesis, I discussed stable numerical implementations using modified Gram–Schmidt and Householder reflectors; related ideas appear in [CK24].

Chapter 10

CUR DECOMPOSITIONS

> *Thus, **C** and/or **R** can be used in place of the eigencolumns and
> eigenrows, but since they consist of actual data elements they will be
> interpretable in terms of the field from which the data are drawn (to the
> extent that the original data points and/or features are interpretable).*
>
> Michael W. Mahoney and Petros Drineas, *CUR matrix decompositions
> for improved data analysis* [MD09]

A CUR decomposition or CUR approximation refers to a low-rank approximation
of a matrix B of the form

$$B \approx CUR \quad \text{where } C = B(:, \mathsf{S}) \text{ and } R = B(\mathsf{T}, :).$$

The matrix B is approximated by a low-rank approximation spanned by a subset
of both its rows and columns. The CUR decomposition is valued for its increased
interpretability over other types of low-rank approximations like the SVD [MD09].
The CUR decomposition is also favored in certain contexts because the C and R
matrices inherit structural properties like sparsity from the matrix B or because,
in some cases, CUR decompositions can be constructed without looking at the
entire input matrix (see Remark 3.5). This section reviews the theory of CUR
decompositions and discussed algorithms for computing them based on random
pivoting.

Sources. This chapter presents original research that has not previously been
published. The references [MD09; PN25] have been helpful in shaping this chapter.

Outline. The CUR decomposition does not fit into our existing taxonomy of ran-
domized low-rank approximation algorithms from Chapter 2, and it is not evident
how we should pick the middle factor U. To remedy this deficit, we shall begin
by briefly reviewing two additional types of low-rank approximation: two-sided
projection approximations (Section 10.1) and generalized Nyström approximations
(Section 10.2). These two classes yield two *inequivalent* types of CUR approxi-
mations, which we will refer to as CUR projection approximations and CUR cross
approximations. Section 10.3 discusses numerically stable representation of CUR

approximations, and Section 10.4 discusses random pivoting algorithms for computing CUR decompositions. Finally, Section 10.5 reviews the Mahoney–Drineas algorithm for computing CUR decompositions and Section 10.6 presents a numerical comparison.

10.1 Two-sided and CUR projection approximation

Given a test matrix $\boldsymbol{\Omega} \in \mathbb{K}^{n \times k}$, the projection approximation is the Frobenius-norm optimal approximation to a matrix $\boldsymbol{B} \in \mathbb{K}^{m \times n}$ in the column span of $\boldsymbol{B\Omega}$. Given a second test matrix $\boldsymbol{\Psi} \in \mathbb{K}^{m \times \ell}$, it is natural to define a *two-sided* projection approximation as the Frobenius-norm optimal approximation to $\boldsymbol{B}$ that both lies in the column span of $\boldsymbol{B\Omega}$ and lies in the row span of $\boldsymbol{\Psi}^* \boldsymbol{B}$. To this end, we have the following result [Ste99, §4].

Proposition 10.1 (Optimal two-sided approximation). *Fix matrices $\boldsymbol{B} \in \mathbb{K}^{m \times n}$, $\boldsymbol{Y} \in \mathbb{K}^{m \times k}$, and $\boldsymbol{Z} \in \mathbb{K}^{n \times \ell}$. Then the approximation*

$$\widehat{\boldsymbol{B}} := \boldsymbol{\Pi}_{\boldsymbol{Y}} \boldsymbol{B} \boldsymbol{\Pi}_{\boldsymbol{Z}} = \boldsymbol{YUZ}^* \quad \textit{for } \boldsymbol{U} := \boldsymbol{Y}^\dagger \boldsymbol{B} \boldsymbol{Z}^{\dagger *}$$

is the unique Frobenius-norm optimal approximation to $\boldsymbol{B}$ whose columns are spanned by the columns of $\boldsymbol{Y}$ and whose rows are spanned by the rows of $\boldsymbol{Z}^*$.

Proof. Consider the space of all approximations

$$\mathsf{S} := \{\boldsymbol{YTZ}^* : \boldsymbol{T} \in \mathbb{K}^{k \times \ell}\}.$$

Geometrically, we are interested in characterizing the unique point $\boldsymbol{B}_\star \in \mathsf{S}$ that is closest to $\boldsymbol{B}$. The vectorization operation $\mathbf{vec} : (\mathbb{K}^{m \times n}, \|\cdot\|_{\mathrm{F}}) \to (\mathbb{K}^{mn}, \|\cdot\|)$ is a linear isometry, so we can equivalently find the point $\mathbf{vec}(\boldsymbol{B}_\star)$ closest to $\mathbf{vec}(\boldsymbol{B})$ in

$$\mathbf{vec}(\mathsf{S}) = \{\mathbf{vec}(\boldsymbol{YTZ}^*) : \boldsymbol{T} \in \mathbb{K}^{k \times \ell}\} = \{(\overline{\boldsymbol{Z}} \otimes \boldsymbol{Y})\boldsymbol{t} : \boldsymbol{t} = \mathbf{vec}(\boldsymbol{T}) \in \mathbb{K}^{k\ell}\}.$$

The second equality is the vec–Kronecker product identity. Thus, we have

$$\mathbf{vec}(\boldsymbol{B}_\star) = \boldsymbol{\Pi}_{\overline{\boldsymbol{Z}} \otimes \boldsymbol{Y}}(\mathbf{vec}(\boldsymbol{B})) = (\overline{\boldsymbol{Z}} \otimes \boldsymbol{Y})(\overline{\boldsymbol{Z}} \otimes \boldsymbol{Y})^\dagger \, \mathbf{vec}(\boldsymbol{B})$$

$$= (\overline{\boldsymbol{Z}} \otimes \boldsymbol{Y})(\overline{\boldsymbol{Z}}^\dagger \otimes \boldsymbol{Y}^\dagger) \, \mathbf{vec}(\boldsymbol{B}) = \mathbf{vec}(\boldsymbol{YY}^\dagger \boldsymbol{B} \boldsymbol{Z}^{\dagger *} \boldsymbol{Z}^*).$$

In the penultimate identity, we used the fact that the pseudoinverse of a Kronecker product is the Kroncker product of the pseudoinverses, which follows by the SVD and the mixed product property for the Kronecker product. In the last equality, we applied the vec–Kronecker product identity twice. Ergo, $\boldsymbol{B}_\star = \boldsymbol{YY}^\dagger \boldsymbol{B} \boldsymbol{Z}^{\dagger *} \boldsymbol{Z}^*$, as promised. $\qquad\square$

This result motivates the following definition:

Definition 10.2 (Two-sided projection approximation and CUR projection approximation)**.** Given test matrices $\boldsymbol{\Omega} \in \mathbb{K}^{n \times k}$ and $\boldsymbol{\Psi} \in \mathbb{K}^{m \times \ell}$, the *two-sided projection approximation* to $\boldsymbol{B} \in \mathbb{K}^{m \times n}$ is

$$\widehat{\boldsymbol{B}} \coloneqq \boldsymbol{\Pi}_{\boldsymbol{B\Omega}} \boldsymbol{B} \boldsymbol{\Pi}_{\boldsymbol{B}^* \boldsymbol{\Psi}}.$$

If $\boldsymbol{\Omega} = \boldsymbol{I}(:, \mathsf{S})$ and $\boldsymbol{\Psi} = \boldsymbol{I}(:, \mathsf{T})$ are column submatrices of the identity matrix, the resulting then the two-sided projection approximation is called a *CUR projection approximation*. It takes the form

$$\widehat{\boldsymbol{B}} = \boldsymbol{CUR} \quad \text{where } \boldsymbol{C} = \boldsymbol{B}(:, \mathsf{S}), \ \boldsymbol{R} \coloneqq \boldsymbol{B}(\mathsf{T}, :), \text{ and } \boldsymbol{U} = \boldsymbol{C}^\dagger \boldsymbol{B} \boldsymbol{R}^\dagger.$$

This definition provides the first natural choice $\boldsymbol{U} = \boldsymbol{C}^\dagger \boldsymbol{B} \boldsymbol{R}^\dagger$ for the core matrix in a CUR decomposition. As a consequence of Proposition 10.1, this type of approximation yields the smallest Frobenius norm error for any middle factor in a CUR approximation. In view of this property, Park and Nakatsukasa [PN25] call this type of CUR approximation a *CUR best approximation*. For this thesis, we will use the term *CUR projection approximation* to emphasize the connection with (two-sided) projection approximations. It is important to emphasize that a CUR projection approximation achieves the lowest possible Frobenius norm error *for a given choice of $\boldsymbol{C}$ and $\boldsymbol{R}$*; with a bad choice of columns and rows, the accuracy of this "best approximation" can be quite poor.

Remark 10.3 (Decomposition or approximation). In the literature, the term "decomposition" is often used to describe the factored approximation $\widehat{\boldsymbol{B}} = \boldsymbol{CUR} \approx \boldsymbol{B}$. I prefer the term CUR approximation, as one typically has $\widehat{\boldsymbol{B}} \neq \boldsymbol{B}$ in applications. Conditions for exactness of the CUR approximation are provided in [HH20]. ◇

Before going forward, we catalog the following useful and standard (for example, see [MD09; SE16; DM23a; CK24]) result:

Proposition 10.4 (One-sided to two-sided)**.** *Fix matrices $\boldsymbol{B} \in \mathbb{K}^{m \times n}$, $\boldsymbol{Y} \in \mathbb{K}^{m \times k}$, and $\boldsymbol{Z} \in \mathbb{K}^{n \times \ell}$. In any unitarily invariant norm $\|\|\cdot\|\|$, we have*

$$\|\|\boldsymbol{B} - \boldsymbol{\Pi}_{\boldsymbol{B\Omega}} \boldsymbol{B} \boldsymbol{\Pi}_{\boldsymbol{B}^* \boldsymbol{\Psi}}\|\| \leq \|\|\boldsymbol{B} - \boldsymbol{\Pi}_{\boldsymbol{B\Omega}} \boldsymbol{B}\|\| + \|\|\boldsymbol{B} - \boldsymbol{B} \boldsymbol{\Pi}_{\boldsymbol{B}^* \boldsymbol{\Psi}}\|\|.$$

Proof. Apply the triangle inequality:

$$\|\|\boldsymbol{B} - \boldsymbol{\Pi}_{\boldsymbol{B\Omega}} \boldsymbol{B} \boldsymbol{\Pi}_{\boldsymbol{B}^* \boldsymbol{\Psi}}\|\| \leq \|\|\boldsymbol{B} - \boldsymbol{\Pi}_{\boldsymbol{B\Omega}} \boldsymbol{B}\|\| + \|\|\boldsymbol{\Pi}_{\boldsymbol{B\Omega}} (\boldsymbol{B} - \boldsymbol{B} \boldsymbol{\Pi}_{\boldsymbol{B}^* \boldsymbol{\Psi}})\|\|.$$

By the operator ideal property $\|\|\boldsymbol{FG}\|\| \leq \|\boldsymbol{F}\| \cdot \|\|\boldsymbol{G}\|\|$, the second term may be bounded as

$$\|\|\boldsymbol{\Pi}_{B\Omega}(\boldsymbol{B} - \boldsymbol{B}\boldsymbol{\Pi}_{B^*\Psi})\|\| \leq \|\boldsymbol{\Pi}_{B\Omega}\| \cdot \|\|\boldsymbol{B} - \boldsymbol{B}\boldsymbol{\Pi}_{B^*\Psi}\|\| = \|\|\boldsymbol{B} - \boldsymbol{B}\boldsymbol{\Pi}_{B^*\Psi}\|\|.$$

This completes the proof. $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ $\square$

10.2 Generalized Nyström approximation and CUR cross approximation

The generalized Nyström approximation is a powerful low-rank approximation format that encompasses both projection approximations and Nyström approximations as special cases. They also lead to another useful, inequalivant way of constructing the middle factor for CUR approximations.

Definition 10.5 (Generalized Nyström approximation and CUR cross approximation)**.** Given test matrices $\boldsymbol{\Omega} \in \mathbb{K}^{n \times k}$ and $\boldsymbol{\Psi} \in \mathbb{K}^{m \times \ell}$, the *generalized Nyström approximation* to $\boldsymbol{B} \in \mathbb{K}^{m \times n}$ is

$$\boldsymbol{B}\langle\boldsymbol{\Omega}, \boldsymbol{\Psi}\rangle \coloneqq (\boldsymbol{B}\boldsymbol{\Omega})(\boldsymbol{\Psi}^*\boldsymbol{B}\boldsymbol{\Omega})^\dagger(\boldsymbol{\Psi}^*\boldsymbol{B}).$$

If $\boldsymbol{\Omega} = \boldsymbol{I}(:, \mathsf{S})$ and $\boldsymbol{\Psi} = \boldsymbol{I}(:, \mathsf{T})$ are column submatrices of the identity matrix, then the generalized Nyström approximation is called a CUR *cross approximation*. It takes the form

$$\boldsymbol{B}\langle\mathsf{S}, \mathsf{T}\rangle \coloneqq \boldsymbol{CUR} \quad \text{where } \boldsymbol{C} = \boldsymbol{B}(:, \mathsf{S}), \boldsymbol{R} \coloneqq \boldsymbol{B}(\mathsf{T}, :), \text{ and } \boldsymbol{U} = \boldsymbol{B}(\mathsf{T}, \mathsf{S})^\dagger.$$

Generalized Nyström approximation includes projection approximation and classical Nyström approximation as the special cases $\boldsymbol{\Psi} = \boldsymbol{I}$ and $\boldsymbol{\Psi} = \boldsymbol{\Omega}$.

The generalized Nyström approximation is *interpolatory*. Consider the case when $k \leq \ell$ and $\boldsymbol{\Psi}^*\boldsymbol{B}\boldsymbol{\Omega}$ is full-rank. Then

$$\boldsymbol{B}\langle\boldsymbol{\Omega}, \boldsymbol{\Psi}\rangle \cdot \boldsymbol{\Omega} = \boldsymbol{B}\boldsymbol{\Omega}(\boldsymbol{\Psi}^*\boldsymbol{B}\boldsymbol{\Omega})^\dagger(\boldsymbol{\Psi}^*\boldsymbol{B}\boldsymbol{\Omega}) = \boldsymbol{B}\boldsymbol{\Omega}.$$

Consequently, if $|\mathsf{S}| \leq |\mathsf{T}|$ and $\boldsymbol{B}(\mathsf{S}, \mathsf{T})$ has full rank, then $\widehat{\boldsymbol{B}} \coloneqq \boldsymbol{B}\langle\mathsf{S}, \mathsf{T}\rangle$ agrees with the matrix $\boldsymbol{B}$ in the selected columns: $\boldsymbol{B}(:, \mathsf{S}) = \widehat{\boldsymbol{B}}(:, \mathsf{S})$. Analogous statements hold in the opposite case ($k \geq \ell$).

CUR projection approximation $\boldsymbol{U} = \boldsymbol{C}^\dagger\boldsymbol{B}\boldsymbol{R}^\dagger$ and CUR cross approximation $\boldsymbol{U} = \boldsymbol{B}(\mathsf{S}, \mathsf{T})^\dagger$ are both natural. The former enjoys the optimality result Proposition 10.1, and the latter satisfies the interpolatory property just discussed. The CUR cross

approximation is particularly attractive for approximating very large matrices, owing to the following property: *Once the index sets* S *and* T *have been identified*, forming the CUR cross approximation only requires accessing $mk + n\ell$ entries of $\boldsymbol{B}$. Thus, the CUR cross approximation forms the basis for low-rank approximation algorithms for general matrices that avoid reading the entire matrix (Remark 3.5).

Remark 10.6 (History and terminology). The generalized Nyström approximation has an interesting history. The first references in the modern randomized linear algebra literature are the seemingly independent works of Woolfe, Liberty, Rokhlin, and Tygert [WLRT08] and of Clarkson and Woodruff [CW09]. An algebraically equivalent type of approximation appears in [TYUC17b], and they suggest an implementation with improved stability properties. The name *generalized Nyström approximation* was suggested by Yuji Nakatsukasa [Nak20], who investigated different stable implementations that zero out small singular values in the core matrix $\boldsymbol{\Psi}^{*}\boldsymbol{B}\boldsymbol{\Omega}$. Per-Gunnar Martinsson and Alex Townsend trace the origin of this type of approximation all the way back to the work of Wedderburn [Wed34, p. 69] in 1934, whose writings contain the formula for a rank $k = \ell = 1$ generalized Nyström approximation. The generalization to $k = \ell > 1$ appears 45 years later in [CF79]; see the paper [CFG95] for more discussion. An important difference between the premodern usage of generalized Nyström approximation and its contemporary usage are the settings of the parameters k and ℓ; the early literature exclusively focused on the case $k = \ell$, whereas the modern literature has exposed benefits of oversampling (e.g., $\ell = \lceil 1.5k \rceil$ or $k = \lceil 1.5\ell \rceil$).

The name CUR cross approximation is taken from [PN25]. The name *cross approximation* for the CUR cross approximation is classical [BR03], particularly when the row and column subset have the same size $|\mathsf{S}| = |\mathsf{T}|$. The name refers to the fact that the decomposition can be computed from only a subset of columns and rows of the matrix. If these subsets are contiguous, they form the shape of a cross on the matrix [GT01]; see Fig. 10.1 for illustration. Earlier literature also uses the term *pseudoskeleton approximations* [GTZ97; GZT97; GT01]. ◇

Weighted CUR cross approximations. One can also consider weighted versions of the CUR cross approximation.

Definition 10.7 (Weighted CUR cross approximation). Let $\mathsf{S} \subseteq \{1, \ldots, n\}$ be a set of k indices and $\mathsf{T} \subseteq \{1, \ldots, m\}$ be a subset of ℓ indices, and let $\boldsymbol{W}_1 \in \mathbb{K}^{k \times k}$ and

$$\boldsymbol{A} = \boldsymbol{A}(:,\mathsf{S}) \cdot \boldsymbol{A}(\mathsf{T},\mathsf{S})^{\dagger} \cdot \boldsymbol{A}(\mathsf{T},:)$$

Figure 10.1: Diagram of cross approximation, showing how the entries selected to form the approximation form a cross.

$\boldsymbol{W}_2 \in \mathbb{K}^{\ell \times \ell}$ be positive *dejinite*. The weighted CUR cross approximation is

$$\begin{aligned}\widehat{\boldsymbol{B}} &:= \boldsymbol{B}(:,\mathsf{S})\boldsymbol{W}_1(\boldsymbol{W}_2\boldsymbol{B}(\mathsf{T},\mathsf{S})\boldsymbol{W}_1)^{\dagger}\boldsymbol{W}_2\boldsymbol{B}(\mathsf{T},:) \\ &= \boldsymbol{B}(:,\mathsf{S})\boldsymbol{U}\boldsymbol{B}(\mathsf{T},:) \qquad\qquad \text{for } \boldsymbol{U} = \boldsymbol{W}_1(\boldsymbol{W}_2\boldsymbol{B}(\mathsf{T},\mathsf{S})\boldsymbol{W}_1)^{\dagger}\boldsymbol{W}_2.\end{aligned}$$

If the matrix $\boldsymbol{B}(\mathsf{T},\mathsf{S})$ has full rank, then only one of the weight matrices plays a role in the approximation. Indeed, if $\ell \geq k$ and $\boldsymbol{B}(\mathsf{T},\mathsf{S})$ has full column-rank, then

$$(\boldsymbol{W}_2\boldsymbol{B}(\mathsf{T},\mathsf{S})\boldsymbol{W}_1)^{\dagger} = \boldsymbol{W}_1^{-1} \cdot (\boldsymbol{W}_2\boldsymbol{B}(\mathsf{T},\mathsf{S}))^{\dagger},$$

and the core matrix only depends on $\boldsymbol{W}_2$:

$$\boldsymbol{U} = (\boldsymbol{W}_2\boldsymbol{B}(\mathsf{T},\mathsf{S}))^{\dagger}\boldsymbol{W}_2.$$

A similar result holds if $k \geq \ell$. In particular, if $k = \ell$ and $\boldsymbol{B}(\mathsf{T},\mathsf{S})$ is full-rank, then $\widehat{\boldsymbol{B}}$ depends on neither weight matrix, and $\widehat{\boldsymbol{B}}$ is just the ordinary CUR cross approximation.

The justification for considering weighted CUR approximations is provided by the following result.

Theorem 10.8 (The value of weighting for CUR approximations)**.** *Fix $k \geq 1$ and any m that is divisible by $k + 1$. There exists a matrix $\boldsymbol{B} \in \mathbb{R}^{m \times 2k}$ and a column subset $\mathsf{S} = \{1, \ldots, 2k\}$ of k elements such that the squared* unweighted *CUR cross approximation error is at least*

$$\left\|\boldsymbol{B} - \boldsymbol{B}(:,\mathsf{S})\boldsymbol{B}(\mathsf{T},\mathsf{S})^{\dagger}\boldsymbol{B}(\mathsf{T},:)\right\|_{\mathrm{F}}^2 \geq 1.5\left\|\boldsymbol{B} - \boldsymbol{B}(:,\mathsf{S})\boldsymbol{B}(:,\mathsf{S})^{\dagger}\boldsymbol{B}(\mathsf{T},:)\right\|_{\mathrm{F}}^2$$

unless the row subset T comprises at least

$$\ell \geq (3 - 2\sqrt{2})m \approx 0.172 \cdot m \text{ elements.}$$

Conversely, for (diagonally) weighted *CUR cross approximations, a row subset of size*

$$\ell = O(k \log k + k/\varepsilon) \tag{10.1}$$

is sufficient to obtain the guarantee

$$\left\|\boldsymbol{B} - \boldsymbol{B}(:,\mathsf{S})(\boldsymbol{W}_2\boldsymbol{B}(\mathsf{T},\mathsf{S}))^\dagger \boldsymbol{W}_2\boldsymbol{B}(\mathsf{T},:)\right\|_{\mathrm{F}}^2 \leq (1+\varepsilon)\left\|\boldsymbol{B} - \boldsymbol{B}(:,\mathsf{S})\boldsymbol{B}(:,\mathsf{S})^\dagger\boldsymbol{B}(\mathsf{T},:)\right\|_{\mathrm{F}}^2.$$

The proof is somewhat lengthy, so we defer it to Appendix D.2. I believe that the dependence of ℓ in (10.1) can be reduced to $\ell = O(k/\varepsilon)$ by extending the results of [CP19] to least-squares problems with multiple right-hand sides.

This result establishes a fundamental separation between the approximation capabilities of weighted and unweighted CUR approximations. In short, the conclusion of Theorem 10.8 may be summarized as follows.

> Weighted CUR cross approximations achieve $(1 + \varepsilon)$-type approximation guarantees using a row-subset of size $\ell = O(k \log k + k/\varepsilon)$, whereas unweighted approximations a constant fraction of the matrix rows $\ell = \Omega(m)$ to achieve such a guarantee.

In practice, the examples which demonstrate a separation between the weighted and unweighted approximations are somewhat pathological, and unweighted CUR cross approximations are usually fine for applications.

10.3 Numerically stable representations

Standard implementations of CUR approximations suffer from issues of numerical stability, as Martinsson and Tropp explain [MT20, §13.1]:

> A disadvantage of the [CUR cross approximation] is that, when the singular values of $\boldsymbol{B}$ decay rapidly, the factorization $[\boldsymbol{B}\langle\mathsf{S},\mathsf{T}\rangle = \boldsymbol{C}\boldsymbol{U}\boldsymbol{R}]$ is typically numerically ill-conditioned. The reason is that, whenever the factorization is a good representation of $\boldsymbol{B}$, the singular values of $\boldsymbol{B}(\mathsf{T},\mathsf{S})$ should approximate the k dominant singular values of $\boldsymbol{B}$, so the singular values of $\boldsymbol{U}$ end up approximating the inverses of these singular values. This means that $\boldsymbol{U}$ will have elements of magnitude $1/\sigma_k$, which is clearly undesirable when σ_k is small. In contrast, the ID is numerically benign.

(We have modified this quote to have consistent notation with the rest of this chapter.) The issues identified in this quote show that working with the *standard representation* $\widehat{\boldsymbol{B}} = \boldsymbol{C}\boldsymbol{U}\boldsymbol{R}$ of the CUR approximation can be numerically problematic.

To fix this instability, the authors of [ADMM+15] propose representing the CUR projection approximation using the factorization

$$\widehat{B} = Q_1 \Xi Q_2^*, \tag{10.2}$$

where $Q_1 := \mathrm{Orth}(B(:,\mathsf{S}))$, $Q_2 := \mathrm{Orth}(B(\mathsf{T},:)^*)$, and $\Xi := Q_1^* B Q_2$. While this format is stable, it no longer represents the low-rank approximation as a CUR factorization $\widehat{B} = CUR$. In particular, the factors Q_1 and Q_2 no longer inherit structural properties of B such as sparsity, and Q_1 and Q_2 lack the interpretability benefits of approximating B using columns and rows.

To remedy the numerical stability issues with CUR approximation while maintaining a factored representation using selected columns and rows, I propose the following new way to stably represent the CUR approximation. The observation is that it can be more numerically stable to work with an *implicit representation* of the core matrix U. Specifically, we represent U as a product

$$U = P^{-1} \cdot G \tag{10.3}$$

of a *well-conditioned* matrix G and a (possibly ill-conditioned) upper-triangular matrix P. This representation can be computed in the following ways:

1. For CUR projection approximations, first use a stable algorithm to compute a *row* interpolative decomposition

$$B \approx W B(\mathsf{T},:) \quad \text{for } W = B B(\mathsf{T},:)^\dagger.$$

   To compute a stable representation (10.3), we compute a QR decomposition of the column submatrix $B(:,\mathsf{S})$:

$$B(:,\mathsf{S}) = QP$$

   and define $G := Q^* W$. The core matrix U admits the factorization (10.3).

2. For (weighted) CUR cross approximations with $\ell \geq k$ and weight matrix W_2, we compute a QR decomposition

$$W_2 B(\mathsf{T},\mathsf{S}) = QP.$$

   and define $G := Q^* W_2$. The core matrix U admits the factorization (10.3).

As we will see in Section 10.6, working with G and P matrices, rather than the standard core matrix U, can lead to significant stability improvements to the CUR approximation while maintaining the core essence of the decomposition $B \approx CUR$. We emphasize that, to reap the numerical stability benefits of this approach, it is critical to store the triangular matrix P, *not its inverse P^{-1}*.

Motivation for improved stability. To motivate why the representation (10.3) is more stable, first consider a square matrix $B \in \mathbb{K}^{n \times n}$, and work with the extreme case when $S = T = \{1, \ldots, n\}$. The standard CUR decomposition $B = CUR$ has factors $C = R = B$ and $U = B^{-1}$; in particular, forming U as a dense array requires explicit computation of the inverse B^{-1}. As can be seen in the following MATLAB code segment, the reconstruction error with this CUR decomposition can be large:

```
B = rand_with_evals(logspace(0,-15,1000)); % Condition num = 1e15
norm(B - B*inv(B)*B)/norm(B)                % Outputs 9.7e-3
```

If we instead compute a QR decomposition $B = QP$ and represent the inverse as $B^{-1} = P^{-1}Q^*$, the reconstruction error drops to the level of the machine precision:

```
[Q,P] = qr(B);
norm(B - B*(P\(Q'*B)))/norm(B)              % Outputs 1.6e-15
```

The stability difference between these two approaches is consequence of the fact that, as a way of solving the system $BX = Y$, the procedure `inv(B)*y` is *forward but not backward stable* [DT12], whereas `P\(Q'*B)` is backward stable. Notions of forward and backward stability are discussed a great deal in Part III of this thesis.

The above example demonstrates the ineffectiveness of explicit inversion in storing and representing the core matrix U in a CUR decomposition in extreme case when $S = T = \{1, \ldots, n\}$. The same principle—that explicit computation of the inverse of a matrix is bad—motivates the use of the implicitly represented core matrix $U = P^{-1} \cdot G$ for storing general CUR approximations.

Using the stable representation. The stable representation (10.3) of the core matrix allows the CUR decomposition to be used in a numerically stable way for downstream tasks. The approximation $\widehat{B}$ can be stably reconstituted as a dense array by the right–left evaluation $C(P^{-1}(GR))$ or the left–right evaluation $((CP^{-1})G)R$; the mixed evaluation order $C((P^{-1}G)R)$ should be avoided, however. The left–right and right–left evaluation orders can also be used to efficiently and stably generate

submatrices or individual entries of $\widehat{\boldsymbol{B}}$. One can also compute matrix products as

$$\widehat{\boldsymbol{B}} \cdot \boldsymbol{X} = \boldsymbol{C}(\boldsymbol{P}^{-1}(\boldsymbol{G}(\boldsymbol{R}\boldsymbol{X}))) \quad \text{or} \quad \boldsymbol{Y}^*\widehat{\boldsymbol{B}} = (((\boldsymbol{Y}^*\boldsymbol{C})\boldsymbol{P}^{-1})\boldsymbol{G})\boldsymbol{R}.$$

Storing a CUR decomposition with stable representation (10.3) is economical, as it only requires retaining the two submatrices $\boldsymbol{C} = \boldsymbol{B}(:, \mathsf{S})$ and $\boldsymbol{R} = \boldsymbol{B}(\mathsf{T}, :)$, plus $O(k\ell)$ storage for the factored core matrix (10.3). If the matrix is structured, storing $\boldsymbol{C}$ and $\boldsymbol{R}$ may be significantly more efficient then storing general matrices $\boldsymbol{Q}_1$ and $\boldsymbol{Q}_2$ in the representation (10.2). In particular, when working with a function matrix $\boldsymbol{B} = \xi(\mathsf{X}, \mathsf{Y})$ (Definition 3.1), the entries of $\boldsymbol{C} = \boldsymbol{B}(:, \mathsf{S})$ and $\boldsymbol{R} = \boldsymbol{B}(\mathsf{T}, :)$ need not ever be stored all at once, as they can be generated on-the-fly using the data X and Y and the function ξ.

Remark 10.9 (Comparison with [PN25]). Park and Nakatsukasa [PN25] also consider the numerical stability of CUR approximation, focusing on unweighted CUR cross approximations. These authors are principally concerned with developing stable algorithms for *reconstructing* a CUR cross approximation as a dense array, which differs from our emphasis on finding a stable way to *store* a CUR approximation while maintaining the traditional $\boldsymbol{CUR}$ factorization.

To convert the CUR approximation $\widehat{\boldsymbol{B}} = \boldsymbol{CUR}$ to a dense matrix, Park and Nakatsukasa compute an (economy-size) SVD $\boldsymbol{B}(\mathsf{T}, \mathsf{S}) = \boldsymbol{W}\boldsymbol{\Sigma}\boldsymbol{V}^*$ and reconstruction $\widehat{\boldsymbol{B}}$ using the formula $\widehat{\boldsymbol{B}} = (\boldsymbol{CV}\boldsymbol{\Sigma}^{-1})(\boldsymbol{W}^*\boldsymbol{R})$. (They also mention the possibility of using a QR decomposition in place of the SVD, which agrees with our approach.) Park and Nakatsukasa provide a formal proof of stability for their approach, and discuss a variant where the pseudoinverse $\boldsymbol{B}(\mathsf{T}, \mathsf{S})^{\dagger}$ is regularized by setting the small singular values of $\boldsymbol{B}(\mathsf{T}, \mathsf{S})$ to zero.

This work extends the insights of Nakatsuksa and Park by proposing the $\boldsymbol{C}(\boldsymbol{P}^{-1}\boldsymbol{G})\boldsymbol{R}$ format as a stable way of representing a CUR decomposition of *any* type. In particular, we show how to use this format to stably represent weighted CUR cross approximations and CUR projection approximations, which is beyond the scope of [PN25]. My hope is that this new work makes clear that, when appropriate care for numerical stability is taken, the CUR approximation *storage format* is safe for use in general-purpose computations, resolving the issues identified by Martinsson and Tropp in the beginning of this section. ◇

Program 10.1 `rpcur2.m`. Implementation of RPCUR2 for computing a CUR projection approximation. Subroutine `rpqr` is provided in Program 9.2.

```
function [S,T,U,G,C] = rpcur2(B,k,l)
% Input:  Matrix B and ranks k and l
% Output: Column and row sets S and T, k*l core matrix U defining CUR
%         approximation Bhat = B(:,S) * U * B(T,:), matrices G and R
%         defining (more) stable representation U = G / R

[~,F,T] = acc_rpqr(B',l,floor(l/2));  % RPQR approximation B ~ F*Q'
W = F/tril(F(T,:));                   % Row interpolation matrix
[Q,F,S] = acc_rpqr(B,k,floor(k/2));   % RPQR approximation B ~ Q*F'
C = tril(F(S,:))';                    % R factor for qr(B(:,S))
G = Q'*W;                             % Well-conditioned matrix
U = C \ G;                            % Standard core matrix U


end
```

10.4   Algorithms

As CUR decompositions require identifying subsets S and T of the columns and rows of the matrix $\boldsymbol{B}$ that approximately span the matrix. As demonstrated throughout this part of the thesis, random pivoting algorithms are well-suited to exactly this task. There are a wide array of possible strategies for computing CUR decompositions using the random pivoting approach. This thesis discusses two possible approaches, which I will call RPCUR2 and RPCURLEV. The former approach separately applies RPQR to $\boldsymbol{B}$ and $\boldsymbol{B}^*$, and the latter approach applies RPQR to obtain a low-rank approximation $\boldsymbol{B} \approx \boldsymbol{QF}^*$, then applies leverage score sampling to $\boldsymbol{Q}$.

RPCUR2: Running RPQR twice

Our first strategy for computing a CUR decomposition is to simply run RPQR twice, once on $\boldsymbol{B}$ to obtain a column set S and once on $\boldsymbol{B}^*$ to obtain a row set T. One may be then combine these subsets to create a CUR decomposition of $\boldsymbol{B}$. While RPCUR2 can be used to create either type of CUR decomposition, it is most natural when used for computing a CUR projection approximation. Code is provided in Program 10.1.

Implementation. RPCUR2 requires two separate executions of RPQR, plus additional post-processing to construct the core matrix. The total runtime is $O(mn(k+\ell))$ operations. For runtime speed and numerical stability, I use the Householder-based accelerated RPQR implementation (Program 9.4) in my code. (I have found examples where RPCUR2 fails catastrophically when using the modified Gram–Schmidt

implementation in Program 9.3.)

Analysis. *A priori* error bounds for the RPCUR2 algorithm can easily be inferred from the results for RPQR (Corollary 9.6) and Proposition 10.4.

Corollary 10.10 (RPCUR2). *Let $B \in \mathbb{K}^{m \times n}$ be a matrix, fix $r \geq 1$, and introduce the squared relative error of the best rank-r approximation:*

$$\eta := \frac{\left\| B - [\![B]\!]_r \right\|_F^2}{\| B \|_F^2}.$$

RPCUR2 (Program 10.1) produces an approximation $\widehat{B}$ satisfying

$$\mathbb{E} \| B - \widehat{B} \|_F^2 \leq 4 \| B - [\![B]\!]_r \|_F^2,$$

provided the parameters k and ℓ satisfy $k, \ell \geq r \log(e/\eta)$.

Automatic rank determination. If one wishes to compute a CUR decomposition with error controlled by a tolerance τ in some *unitarily invariant* norm, one can achieve this goal by running each RPQR step with a tolerance of $\tau/2$, in view of Proposition 10.4.

RPCURLᴇᴠ: Combining RPQR and leverage score sampling

Our second algorithm for computing a CUR decomposition combines RPQR to compute the column set S with leverage score sampling (Definition 3.15) to determine the pivot set T. It most naturally outputs a (diagonally) *weighted* CUR cross approximation, but it can also be used with to compute a CUR projection approximation or unweighted CUR cross approximation if desired.

The RPCURLᴇᴠ algorithm proceeds as follows. First, run RPQR to obtain a column projection approximation $B \approx QF^*$. Recall that, for the matrix Q with orthonormal columns, its leverage scores $\lambda = \mathbf{srn}(Q)$ are its squared row norms. We then form T by sampling iid from the leverage score distribution

$$\mathsf{T} = \{t_1, \ldots, t_\ell\} \quad \text{where } t_1, \ldots, t_\ell \overset{\text{iid}}{\sim} \lambda.$$

We emphasize sampling is to be done *with replacement*. For reasons that will soon become more clear, it is most natural to output a weighted CUR decomposition with weight matrix $W_2 = \mathbf{Diag}(\lambda(\mathsf{T}))^{-1/2}$.

Code is given in Program 10.2. We highlight that this implementation computes the pseudoinverse numerically using a column-pivoted QR decomposition, discarding

Program 10.2 `rpcur_lev.m`. Implementation of RPCURLᴇᴠ for computing a weighted CUR cross approximation to a matrix. Subroutines `rpqr` and `sqrownorms` are provided in Programs 9.2 and F.3.

```
function [S,T,U,G,P] = rpcur_lev(B,k,l)
% Input:  Matrix B and ranks k and l
% Output: Column and row sets S and T, k*l core matrix U defining CUR
%         approximation B ~ B(:,S) * U * B(T,:), matrices G and P
%         defining (more) stable representation U = P \ G

[Q,~,S] = acc_rpqr(B,k,floor(k/2)); % RPQR approximation B ~ Q*F'
lev = sqrownorms(Q);                % Leverage scores of Q
T = datasample(1:size(B,1),l,"Weights",lev); % Leverage score sample
w = lev(T) .^ (-1/2);               % Reweight rows
[Q,P,p] = qr(w .* B(T,S),"econ","vector");   % Pivoted QR
dP = abs(diag(P));                  % Diagonal of triangular factor
idx = find(dP > 20*eps*max(dP));    % Find large diagonal entries
Q = Q(:,idx); P = P(idx,idx);       % Delete negligible values in Q,P
S = S(p(idx));                      % Filter pivots
G = Q' .* w.';                      % Well-conditioned matrix
U = P \ G;                          % Standard core matrix U

end
```

entries from the index set S to ensure that $\boldsymbol{B}(\mathsf{T},\mathsf{S})$ is numerically full-rank. The use of column pivoting makes this approach robust, even for sparse problems which could have many zero entries.

Runtime. RPCURLᴇᴠ requires running k steps of RPQR, sampling ℓ elements from a weighted probability distribution on m items, and doing post-processing on an $O(k\ell)$ matrix. The runtime of RPCURLᴇᴠ is $O(mnk + k^2\ell)$. I use accelerated RPQR (Program 9.4) in my implementation. As we will see in Fig. 10.3, the RPCURLᴇᴠ algorithm can be meaningfully faster than the RPCUR2 algorithm.

Analysis. To analyze RPCURLᴇᴠ, we can make use of existing results from the leverage score literature. We shall use the following result:

Fact 10.11 (Leverage score sampling for least squares)**.** *Let $\boldsymbol{C} \in \mathbb{K}^{m \times k}$ be a full-rank matrix, and let $\boldsymbol{F} \in \mathbb{K}^{m \times n}$. Let $\boldsymbol{\lambda} = \mathbf{srn}(\mathbf{Orth}(\boldsymbol{C}))$ be the leverage scores of $\boldsymbol{C}$ and sample an index set T of ℓ elements iid with replacement from $\boldsymbol{\lambda}$. Introduce the diagonal weight matrix $\boldsymbol{W} \coloneqq \mathbf{Diag}(\boldsymbol{\lambda}(\mathsf{T}))^{-1/2}$. Provided*

$$\ell \geq \Omega\left(k \log k + \frac{k}{\varepsilon}\right),$$

it holds with at least 99% probability that

$$\left\|F - C[WC(\mathsf{T},:)]^\dagger[WF(\mathsf{T},:)]\right\|_{\mathrm{F}} \le (1+\varepsilon)\left\|F - CC^\dagger F\right\|_{\mathrm{F}}.$$

This result is a variant of [CW17, Thm. 7.9], with the version stated here following straightforwardly from the proof of that result. This result leads immediately to a error bound for RPCURLᴇᴠ.

Corollary 10.12 (RPCURLᴇᴠ). *Let $B \in \mathbb{K}^{m \times n}$ be a matrix, fix $r \ge 1$ and $\varepsilon \in (0, 1)$, and introduce the squared relative error of the best rank-r approximation:*

$$\eta := \frac{\left\|B - [\![B]\!]_r\right\|_{\mathrm{F}}^2}{\|B\|_{\mathrm{F}}^2}.$$

With 99% probability, the weighted CUR *cross approximation $\widehat{B}$ produced by RPCUR2 (Program 10.1) produces an approximation satisfying*

$$\mathbb{E}_{\mathsf{S}}\|B - \widehat{B}\|_{\mathrm{F}}^2 \le (1+\varepsilon)\left\|B - [\![B]\!]_r\right\|_{\mathrm{F}}^2,$$

provided the parameters k and ℓ satisfy

$$k \ge \frac{2r}{\varepsilon} + r \log\left(\frac{2}{\eta\varepsilon}\right), \quad \ell \ge \Omega\left(k \log k + \frac{k}{\varepsilon}\right)$$

Here, $\mathbb{E}_{\mathsf{S}}$ denotes the expectation over the randomness in the first index set S.

The proof is immediate from Corollary 9.6 and Fact 10.11. This result suggests (correctly) that both oversampling and weighting $\ell > k$ are necessary for RPCURLᴇᴠ to produce high-quality approximations in general.

The RPCURLᴇᴠ procedure represents only one way of combining RPQR with an extra step of row sampling to produce a (weighted) CUR cross approximation. In my experience, it reliably produces high-quality approximations *when implemented with moderately high oversampling* $\ell = O(k \log k)$; approaches using pivoting [SE16; PN25; CK24] are more effective for small $\ell \approx k$. In place of leverage score sampling, natural alternatives are volume sampling [DRVW06; DW17; Der18], adaptive randomized pivoting [CK24], leveraged volume sampling [DH18], the Chen–Price method [CP19], and pivoting strategies such as the Park–Nakatsukasa algorithm [PN25, Alg. 3.2].

Parameter settings. On worst-case examples, the logarithmic oversampling $\ell \ge \Omega(k \log k)$ suggested by Corollary 10.12 is necessary for RPCURLᴇᴠ to succeed.

We recommend $\ell = \lceil 1.5k \log k \rceil$ as a sensible default value. If one wants to control the error up to a tolerance τ in the Frobenius norm, we recommend running RPQR with a lower tolerance ($\tau/3$, say) to allow for the additional error produced by selection of the row subset T.

10.5 Related work: Mahoney and Drineas' algorithm

In their influential work on CUR decompositions for data analysis, Mahoney and Drineas [MD09] proposed a pure leverage score sampling-based approach to select index sets S and T for CUR approximations; see also the follow-up work [DMM08] which obtains better theoretical results. Up to some small tweaks, their algorithm works as follows:

1. **SVD.** Compute an SVD $\boldsymbol{B} = \boldsymbol{U\Sigma V}^*$.

2. **Column sampling.** Choose a parameter $k' \leq k$, and sample k indices $\mathsf{S} = \{s_1, \ldots, s_k\}$ iid from the leverage score distribution of $\boldsymbol{V}(:, 1 : k')$:

$$s_1, \ldots, s_k \overset{\text{iid}}{\sim} \mathbf{srn}(\boldsymbol{V}(:, 1 : k')).$$

3. **Row sampling.** Choose a parameter $\ell' \leq \ell$, and sample ℓ indices $\mathsf{T} = \{t_1, \ldots, t_\ell\}$ iid from the leverage score distribution of $\boldsymbol{U}(:, 1 : \ell')$:

$$t_1, \ldots, t_k \overset{\text{iid}}{\sim} \mathbf{srn}(\boldsymbol{U}(:, 1 : \ell')).$$

4. **Output.** Return the CUR projection approximation with index sets S and T.

To achieve a $(r, 1+\varepsilon, 2)$-approximation, Mahoney and Drineas suggest choosing $k' = \ell' = r$ and $k, \ell = O((r \log r)/\varepsilon^2)$. I believe the parameters $k, \ell = O(r/\varepsilon + r \log r)$ should suffice with modern proof techniques, but I have not confirmed this. In their original algorithm, Drineas and Mahoney suggest including or not including each element $1 \leq s \leq n$ or $1 \leq t \leq m$ in the sets S or T independently with some probability so that the *expected* number of accepted indices is $\mathbb{E}[k], \mathbb{E}[\ell] = O((r \log r)/\varepsilon^2)$; we have modified the algorithm here to pick a fixed number of indices. In our experiments, we set $k' = \lceil k/(2 \log k) \rceil$ and $\ell' = \lceil \ell/(2 \log \ell) \rceil$. We refer to this algorithm as Mahoney–Drineas CUR (MDCUR) and provide an implementation in Program 10.3.

Runtime. The dominant cost of the MDCUR algorithm is the SVD computation. Using a dense SVD, its cost is $O(mn \min\{m, n\})$ operations. We can accelerate the

Program 10.3 `md_cur.m`. Implementation of MDCUR for computing a CUR projection approximation.

```
function [S,T,U,G,P] = md_cur(B,k,l,U,V)
% Input:  Matrix B, ranks k and l, and top left and right singular
%         vectors U and V
% Output: Column and row sets S and T, k*l core matrix U defining CUR
%         approximation B ~ B(:,S) * U * B(T,:), matrices G and P
%         defining (more) stable representation U = P \ G

[m,n] = size(B);
S = datasample(1:n,k,"Replace",false,"Weights",sqrownorms(V));
T = datasample(1:m,l,"Replace",false,"Weights",sqrownorms(U));
[Q,R] = qr(B(T,:)',"econ");
W = (B*Q)/R';                        % Row interpolation matrix
[Q,P] = qr(B(:,S),"econ");
G = Q'*W;                            % Well-conditioned matrix
U = P \ G;                           % Standard core matrix U

end
```

algorithm by using a randomized SVD, reducing the cost to $O(mn(k' + \ell'))$. The remaining steps of the algorithm incur a post-processing cost of $O(mk^2 + n\ell^2)$.

Further alternatives. Another approach to computing CUR decompositions is to use sketchy pivoting, discussed in Section 9.4 [DM23b]. The comparison of randomly pivoted CUR methods to sketchy pivoting methods for CUR is similar to the comparison of RPQR methods to sketchy pivoting methods for interpolative decomposition. In particular, sketchy pivoting methods can be faster but generally require an upper bound on the approximation rank k to be known in advance, which is a significant limitation for some applications. See Section 9.4 and [DCMP24] for further discussion on the relative merits of sketchy pivoting and random pivoting.

There are many other algorithms for computing CUR decompositions including squared row and column norm sampling [DKM06], adaptive cross approximation [BR03], adaptive randomized pivoting [CK24], and more sophisticated multi-stage sampling approaches [BW17]. Comparison of the random pivoting algorithms against these approaches is a natural subject for future work.

10.6 Experiments

To begin exploring the performance of the two randomly pivoted CUR algorithms, we present some preliminary tests on three different example matrices.
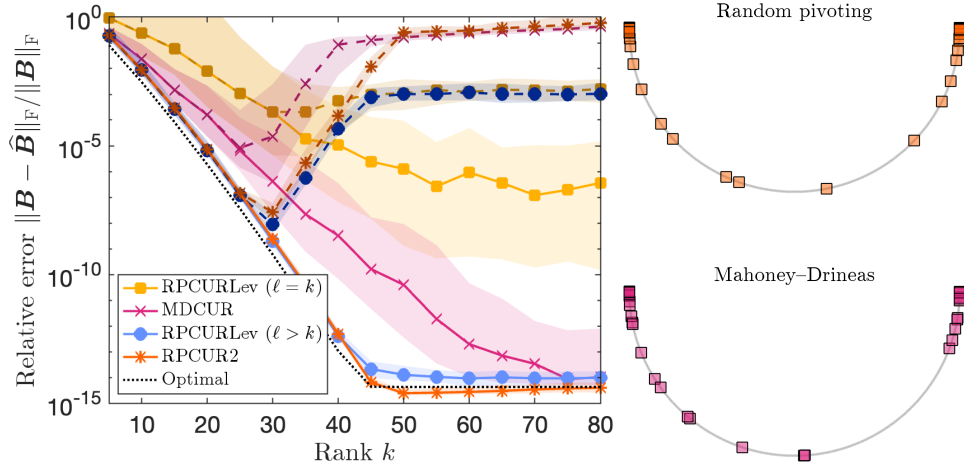
Figure 10.2: *Left:* Relative error $\|\boldsymbol{B} - \widehat{\boldsymbol{B}}\|_{\mathrm{F}}/\|\boldsymbol{B}\|_{\mathrm{F}}$ computed by the following methods: RPCUR2 (orange stars), RPCURLEV with (blue circles) and without oversampling (yellow squares), and MDCUR (pink crosses). Solid light lines for CUR algorithms use the stable representation $\boldsymbol{U} = \boldsymbol{P}^{-1} \cdot \boldsymbol{G}$ from Section 10.3, and dashed dark lines store $\boldsymbol{U}$ as a dense array. The error for the best rank-k approximation $\widehat{\boldsymbol{B}} = [\![\boldsymbol{B}]\!]_k$ is shown as a dotted black line. We plot the median of 100 trials, with error bars showing 10% and 90% quantiles. *Right:* Points w_j in column subset $j \in \mathsf{S}$ selected by random pivoting methods (*top*) and the Mahoney–Drineas algorithm (*bottom*) with $k = 30$.

Experiment #1: Function matrix. First, as a simple test of approximation quality and numerical stability, we test the two randomly pivoted CUR algorithms of the previous section on a simple test matrix $\boldsymbol{B} \in \mathbb{C}^{n \times n}$ with entries

$$b_{ij} = \frac{1}{z_i - w_j}.$$

The points z_i and w_j are chosen to be equispaced on the complex unit circle $\mathbb{T}(\mathbb{C})$:

$$z_i \coloneqq \exp\left(2\pi \mathrm{i} \cdot \frac{i-1}{2n}\right), \quad w_i \coloneqq \exp\left(2\pi \mathrm{i} \cdot \frac{n+i-1}{2n}\right) \quad \text{for } i = 1, 2, \ldots, n,$$

and we set $n \coloneqq 1000$. The points z_i trace the portion of the unit circle in the upper half plane, $\mathbb{T}(\mathbb{C}) \cap \{\mathrm{Im}(z) \geq 0\}$, and the w_j trace the lower half, $\mathbb{T}(\mathbb{C}) \cap \{\mathrm{Im}(z) \leq 0\}$. Matrices similar to $\boldsymbol{B}$ occur in the design of algorithms for structured systems of linear equations and least-squares problems [CGSX+08; XXG12; XXCB14; WEB25; BKW25].

Figure 10.2 shows results for RPCUR2, RPCURLEV (with oversampling $\ell = \lceil 1.5k \log k \rceil$ and no oversampling $\ell = k$), and MDCUR. As points of comparison, I also include the optimal rank-k approximation. We note the following conclusions:

- **Near-optimality for random pivoting.** On this example, RPCUR2 and RPCURLᴇᴠ with oversampling both produces approximations of comparable quality to the optimal rank-k approximation. MDCUR, by contrast, converges at a slower rate. These results demonstrates the benefits of using random pivoting to select at least one of the row or column subsets.

- **Necessity of oversampling for RPCURLᴇᴠ.** Comparing the accuracy of the RPCURLᴇᴠ method with and without oversampling, we see that oversampling $\ell > k$ is crucial to maintain accuracy of the procedure.

- **Stability benefits of factored U matrix representation (10.3).** The solid light curves show the results of the CUR algorithms with the stable representation (10.3), which eventually achieve machine accuracy and even slightly beat the optimal rank-k approximation *due to finite-precision effects*. The dark dashed curves show the results of the CUR algorithms with an explicit U matrix. When the accuracy is smaller ($\lessgtr u^{1/2}$ for the random pivoting methods), the explicit U implementations have similar accuracy to the factored representation. However, while the relative error of the stable methods improve past this point, the relative error of the unstable implementations begins to *increase* for large values of k. These results confirm the significant numerical issues of the standard implementation of the CUR decomposition for high-accuracy matrix approximation.

- **Visualization of column subsets.** Column subsets S of $k = 30$ elements selected by random pivoting methods and the Mahoney–Drineas method are depicted visually in the right panels of Fig. 10.2. Both methods cluster landmarks near the edges of the semicircle, with random pivoting methods producing a more even distribution. Because the Mahoney–Drineas method samples points iid, its landmark set contains examples of nearly overlapping landmarks and regions that lack landmarks, which explains the differences in approximation quality with the random pivoting methods.

Example #2: Sparse matrix. As a second example, we evaluate the methods on the `Meszaros/large` matrix from the SuiteSparse collection. This example was used in the recent paper [CK24, §6.1]. We store this matrix as a dense matrix for the experiments in this section.

We evaluate the runtime and accuracy of RPCUR2, RPCURLᴇᴠ (with oversampling $\ell = \lceil 1.5k \log k \rceil$), and MDCUR on this example. To make the MDCUR competitive
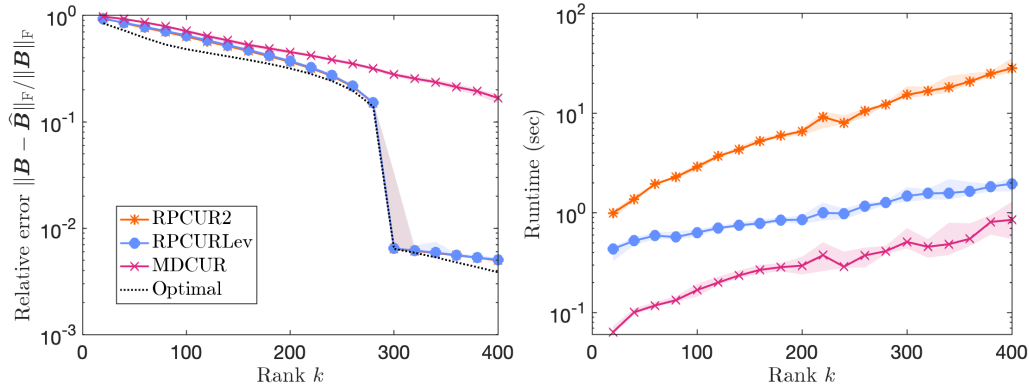
Figure 10.3: Relative error (*left*) and runtime (*right*) for RPCUR2 (orange stars), RPCURLᴇᴠ (blue circles), and MDCUR (pink crosses) on `Meszaros/large` sparse matrix. We plot the median of 100 trials, with error bars showing 10% and 90% quantiles.

in terms of runtime, we use the randomized SVD (Program 2.1) with rank k' to approximate the dominant left and right singular vectors of $\boldsymbol{B}$. RPCUR2 and MDCUR are implemented without oversampling $k = \ell$. Due to the sparsity of $\boldsymbol{B}$, the column pivoted QR decomposition in Program 10.2 is necessary for the algorithm to succeed.

Results are shown in Fig. 10.3. The MDCUR algorithm is the fastest, and RPCUR2 is the slowest. Both RPCUR2 and RPCURLᴇᴠ achieve accuracy comparable to the optimal rank-k approximation, whereas MDCUR lags behind the other methods, particularly when $k \geq 300$. These results demonstrate how, on difficult examples, MDCUR can produce approximations comparable with the best rank-$(r < k)$ approximation, whereas the random pivoting approximations empirically produce approximations comparable to the best rank-k approximation. The need for oversampling presents a weakness of pure leverage score sampling based approaches over random pivoting approaches.

C h a p t e r 11

RANDOM PIVOTING: OPEN PROBLEMS

We conclude this part of the thesis with two open problems: improved error bounds for RPCHOLESKY (Section 11.1) and whether there are simple, effective algorithms for approximating a psd matrix from a small number of entry accesses when errors are measured using the Frobenius norm, rather than the trace (Section 11.2).

11.1 Open problem: RPCHOLESKY error bounds

The bound Theorem 4.1 already provides a useful description of the performance of RPCHOLESKY, showing that one achieves an (r, ε)-approximation (Definition 3.6) in

$$k \leq \frac{r}{\varepsilon} + r \log\left(\frac{1}{\varepsilon\eta}\right) \text{ steps.} \tag{11.1}$$

Here,

$$\eta = \frac{\text{tr}(A - [\![A]\!]_r)}{\text{tr}(A)}$$

is the relative error of the best rank-r approximation. The bound Theorem 4.3 provides weak—but sometimes useful control—related to the spectral norm. There are a few directions along which one could hope to improve our theoretical understanding of RPCHOLESKY.

Bounds in the limit $\eta \downarrow 0$***.*** As noted in Remark 4.2, the bound (11.1) degenerates to infinity in the limit when the relative error $\eta \downarrow 0$. At least in some cases, this bound badly mischaracterizes the performance of the RPCHOLESKY algorithm, which converges in exactly r steps when applied to a matrix with rank $A = r$.

For cases when η is small, [CETW25, Thm. 5.1] establishes that the (r, ε)-guarantee holds when

$$k \geq \frac{r}{\varepsilon} + r + \max\left\{0, r^2 \log(2) + r \log(1/\varepsilon)\right\}.$$

This bound is nice because it purges the relative error η from the bound entirely On the other hand, k is required to be *quadratic* in r to produce an approximation comparable to the best-r approximation—ouch!

R. J. Webber and I conjecture that this bound can be improved as follows.

Conjecture 11.1 (Better η-free RPCHOLESKY bounds)**.** *RPCHOLESKY achieves an (r, ε)-approximation to any psd matrix in $r/\varepsilon + O(r \log(r/\varepsilon))$ operations.*

Here is one concrete approach that could resolve this conjecture. Specifically, we make another conjecture:

Conjecture 11.2 (Improved r-step bound). *For any psd matrix A, the r-step RP-CHOLESKY residual $A^{(r)}$ satisfies*

$$\operatorname{tr}(A - A^{(r)}) \le C(r) \operatorname{tr}(A - [\![A]\!]_r) \quad \textit{for } C(r) = \operatorname{poly}(r). \qquad (11.2)$$

Conjecture 11.2 would imply Conjecture 11.1. Unfortunately, we are currently exponentially far from proving Conjecture 11.2; our best bound [CETW25, Lem. 5.5] establishes (11.2) with $C(r) = 2^r$.

Spectral-norm bounds. For reasons that will be described in Section 11.2, it is not possible for RPCHOLESKY or any psd low-rank approximation algorithm that uses limited entry accesses to produce a relative-error approximation in the spectral norm (that is, an (r, ε, ∞)-approximation) in $O(n \operatorname{poly}(r/\varepsilon))$ entry access operations. However, it is reasonable to hope that RPCHOLESKY produces approximations that are accurate when measured in the spectral norm once the number of steps is comparable to the *effective dimension*, defined in Definition 3.16. Indeed, we saw earlier in Fact 3.17 that ridge leverage score sampling achieves such a bound. We conjecture that RPCHOLESKY does equally well.

Conjecture 11.3 (RPCHOLESKY: Spectral norm bounds). *Let $\lambda \ge 0$. If RPCHOLESKY is executed for $k = O(\operatorname{d_{eff}}(\lambda) \operatorname{polylog}(\operatorname{d_{eff}}(\lambda), \|A\|/\lambda))$ steps, it produces an index set satisfying*

$$A\langle \mathsf{S} \rangle \preceq A \preceq A\langle \mathsf{S} \rangle + \lambda I$$

with at least 99% probability. Here, polylog *denotes an unspecified polylogarithmic function.*

If such a result were true, it would provide theoretical support for the empirical observation that RPCHOLESKY performs similarly to or better than RLS sampling for column Nyström preconditioning (Section 6.1). To be conservative, I have included polylogarithmic factors in the conjecture, but I believe the conjecture will also be true without them.

11.2 Open problem: Frobenius-norm psd low-rank approximation

The goal of this part of the thesis has been to develop algorithms that evaluate a small number of entries to compute an $(r, \varepsilon, 1)$-approximation to a psd matrix. Recall

from Definition 3.6 that an (r, ε, p)-approximation is a random matrix $\widehat{A}$ such that

$$\mathbb{E}\|A - \widehat{A}\|_{\mathsf{S}_p} \le (1 + \varepsilon)\|A - [\![A]\!]_r\|_{\mathsf{S}_p}.$$

Recall that $\|\cdot\|_{\mathsf{S}_p}$ is the Schatten p-norm. It is frame the question: Does the psd low-rank approximation problem admit efficient algorithms when $p > 1$?

Many of the algorithms we have considered so far, including RPCHOLESKY, uniform column Nyström approximation, greedy pivoted partial Cholesky, and the entire family of Gibbs RPCHOLESKY methods (Section 4.4), are *diagonal–column access algorithms*. These methods interact with the matrix A only by reading the diagonal and full columns. As we will see, diagonal–column access algorithms have fundamental limitations for psd low-rank approximation, and a more general class of algorithms are needed order to compute (r, ε, p)-approximations for $p > 1$.

The following result establishes establishes limits on both general and diagonal–column access psd low rank approximation algorithms.

Proposition 11.4 (Psd low-rank approximation: Lower bound). *Fix $p \in [1, \infty]$. On a worst-case input matrix,*

- *A diagonal–column access algorithm must read $\Omega(n^{2-1/p})$ entries to guarantee a $(1, \mathrm{c}, p)$-approximation.*

- *Any algorithm must read $\Omega(\max\{n, n^{2-2/p}\})$ entries to guarantee a $(1, \mathrm{c}, p)$-approximation.*

Here, $\mathrm{c} > 0$ is a universal constant.

Proof sketch. Choose n to be large, and construct the matrix

$$A = P \begin{bmatrix} \mathbf{I}_{n-\lceil n^{1/p} \rceil} & \mathbf{0} \\ \mathbf{0} & \mathbf{1}_{\lceil 2n^{1/p} \rceil} \mathbf{1}^*_{\lceil 2n^{1/p} \rceil} \end{bmatrix} P^*,$$

where P is a uniformly random permutation matrix. The optimal rank-one approximation to this matrix is

$$[\![A]\!]_1 = P \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1}_{\lceil 2n^{1/p} \rceil} \mathbf{1}^*_{\lceil 2n^{1/p} \rceil} \end{bmatrix} P^*.$$

We compute

$$\|A\|_{\mathsf{S}_p} = \left((n - \lceil 2n^{1/p}\rceil)\cdot 1 + (\lceil 2n^{1/p}\rceil)^p\right)^{1/p} \geq (2 + o(1))n^{1/p},$$

$$\left\|A - [\![A]\!]_1\right\|_{\mathsf{S}_p} = \left((n - \lceil 2n^{1/p}\rceil)\cdot 1\right)^{1/p} = (1 + o(1))n^{1/p}.$$

To produce a nontrivial low-rank approximation requires identifying a nonzero off-diagonal entry.

Since each diagonal entry is one, any diagonal–column access algorithm must keep accessing columns until it finds a nonzero off-diagonal entry. Each column access has a $\lceil 2n^{1/p}\rceil/n = (2+o(1))n^{1/p-1}$ probability of identifying a nonzero off-diagonal entry, so it takes $\Omega(n^{1-1/p})$ column accesses $(= \Omega(n^{2-1/p})$ entry accesses) to find a nonzero off-diagonal entry with high probability.

A general psd low-rank approximation algorithm is free to query off-diagonal entries one-by-one. Until a nonzero entry is found, each query finds a nonzero off-diagonal entry with probability $O(n^{2/p-2})$, so it requires $\Omega(n^{2-2/p})$ accesses to find a nonzero off-diagonal entry with high probability. The fact that it always requires $\Omega(n)$ accesses to produce a nontrivial low-rank approximation can be established by considering the matrix $\mathbf{e}_i\mathbf{e}_i^*$ where $i \sim \mathrm{U_{NIF}}\{1,\ldots,n\}$. □

Proposition 11.4 shows that, for RPCʜᴏʟᴇsᴋʏ and other diagonal–column access algorithms, approximating matrices in the trace norm is essentially as good as one can expect, at least for an algorithm with an $O(n)$ runtime. However, this result leaves open the possibility of achieving relative-error approximations in the Frobenius norm with $O(n)$ operations using a more general access pattern. Indeed, such $O(n)$ algorithms for producing $(r, \varepsilon, 2)$-approximations do exist. The first such algorithm was discovered by Musco and Woodruff [MW17], which produces an $(r, \varepsilon, 2)$ in $\widetilde{O}(nr/\varepsilon^{2.5})$ entry accesses; this was improved to $\widetilde{O}(nr/\varepsilon)$ entries in [BCW20]. Despite these appealingly low entry access counts, the algorithms of [MW17; BCW20] are complicated and have large prefactor constants. That leads us to the open question:

> Is there a simple, effective, and practically performant algorithm for psd low-rank approximation that achieves relative error guarantees in the Frobenius norm?

As Proposition 11.4 shows, such an algorithm must use a more sophisticated access model than diagonal and column accesses alone. Additionally, even basic existence

questions are open, as far as I am aware. For instance, does there exist a set S of $\text{poly}(r/\varepsilon)$ pivots defining a column Nyström approximation $A\langle\mathsf{S}\rangle$ that is an $(r, \varepsilon, 2)$ approximation? Or must we use more sophisticated low-rank approximation formats, such as weighted CUR cross approximations, to produce Frobenius-norm relative-error approximations?

Part II

Leave-one-out randomized matrix algorithms

Dedicated to my parents Meg and Tom Epperly.

Chapter 12

MATRIX ATTRIBUTE ESTIMATION PROBLEMS

> *This kind of structure [sparsity] is readily exploited by the iterative*
> *methods we shall discuss, for these algorithms use a matrix in the form*
> *of a* black box. *The iterative algorithm requires nothing more than the*
> *ability to determine* $\boldsymbol{Ax}$ *for any* $\boldsymbol{x}$, *which in a computer program will be*
> *effected by a procedure whose internal workings need be of no concern*
> *to the designer of the iterative algorithm.*
>
> Lloyd N. Trefethen and David Bau, III, *Numerical Linear Algebra*
> [TB22, p. 244]

Frequently in applications, we encounter problems where we wish to learn information about an unknown matrix $\boldsymbol{B} \in \mathbb{C}^{m \times n}$ that can only be accessed by matrix–vector products $\omega \mapsto \boldsymbol{B}\omega$ and, possibly, matrix–vector products with the adjoint $\omega \mapsto \boldsymbol{B}^*\omega$. As examples, we might be interested in learning the matrix trace $\mathrm{tr}(\boldsymbol{B})$, the matrix diagonal $\mathbf{diag}(\boldsymbol{B})$, or individual entries b_{ij}.

In this part of the thesis, I will present a new approach to designing randomized algorithms for such *matrix attribute estimation* problems called the *leave-one-out* approach. The leave-one-out approach leads to fast, resource-efficient algorithms with state-of-the-art accuracy. After introducing the necessary preliminaries in Chapter 13, we will discuss the leave-one-out approach in Chapter 14. This initial chapter serves to motivate the problem of estimating attributes of a matrix through matrix–vector products. We will address the questions "When are we able to access a matrix $\boldsymbol{B}$ only through matrix–vector products?" and "Which attributes of $\boldsymbol{B}$ do we want to estimate in applications?"

Sources. This is an introductory chapter and is not based on any particular research article. The resources [US17; Pop23] are useful resources for applications of trace and diagonal estimation algorithms, respectively.

Outline. Section 12.1 discusses matrix attribute estimation problems and the matrix–vector ("matvec") model for measuring their computational cost. Section 12.2 discusses examples of matrix attribute estimation problems including trace, diagonal, and row-norm estimation. It also discusses applications.

12.1 Matrix attribute estimation and the matvec model

Linear algebraic algorithms that access a matrix B through matrix–vector products (affectionately abbreviated *matvecs*) are extremely common. Examples include power iteration for computing eigenvalues [GV13, §8.2] as well as Krylov iteration for solving linear systems [Saa03], computing eigenvalues [Saa11], and applying matrix functions [Che24]. For these classical iterative methods, the matrix B is often stored explicitly in memory as either a dense or sparse array, and matvec algorithms are used due to their computational efficiency (particularly when B is sparse, making $z \mapsto Bz$ is cheap to compute).

This thesis focuses on an alternative setting where the matrix B is not stored explicitly and B can *only* be accessed through matvecs (and possibly matvecs with the adjoint B^*). This more restrictive setting is motivated by the following examples:

- **Products of matrices.** The matrix B is a product of several other matrices, $B = C_1 C_2 \cdots C_\ell$. It is expensive to evaluate the matrix B, as it requires forming the product, but matvecs with B can be computed efficiently by multiplying a vector against the C_i matrices one at a time: $B\omega = C_1(C_2(\cdots (C_\ell \omega) \cdots))$. Important special cases are the Gram matrix $B = C^* C$ and powers $B = C^k$.

- **Matrix functions.** Let $C \in \mathbb{C}^{n \times n}$ be a matrix and $f : \mathbb{C} \to \mathbb{C}$ be a function defined on the spectrum of C. Set $B \coloneqq f(C)$ using the standard extension of scalar functions to matrix inputs [Hig08, §1.2]. Computing $B = f(C)$ explicitly is typically costly, but matvecs $B\omega = f(C)\omega$ can often be computed efficiently by algorithms such as the Arnoldi method or, if C is Hermitian, the Lanczos method [Hig08, §13.2]. (In particular, for the special case $f(z) = z^{-1}$, computing products $C^{-1}\omega$ is equivalent to solving linear systems of the form $Cy = \omega$, for which there has been *much* work [Saa03].)

- **Automatic differentiation.** Given a twice differentiable function $f : \mathbb{R}^n \to \mathbb{R}$ and an input $x \in \mathbb{R}^n$, automatic differentiation allows one to compute matvecs $D^2 f(x) \cdot \omega$ with the *Hessian matrix* $D^2 f(x) \in \mathbb{R}^{n \times n}$ in a small multiple of the runtime required to evaluate $f(x)$ [BR24]. Similarly, given a function $g : \mathbb{R}^n \to \mathbb{R}^m$, matvecs $Dg(x) \cdot \omega$ with the Jacobian matrix $Dg(x) \in \mathbb{R}^{m \times n}$ are also efficiently computable by automatic differentiation.

In all of these settings, matvecs with the adjoint B^* are typically also available. These examples motivate the *matvec* model:

Matvec model. A matrix $B \in \mathbb{R}^{m \times n}$ is provided that can only be accessed through matvecs $\omega \mapsto B\omega$ and adjoint matvecs $\omega \mapsto B^*\omega$. The cost of an algorithm is measured by the number of matvecs used.

Under the matvec model, the entire matrix B can be recovered one column at a time by computing the matvecs $B\mathbf{e}_1, \ldots, B\mathbf{e}_n$ with each of the standard basis vectors $\mathbf{e}_i$. Thus, in the matvec model, any problem can be solved at the "trivial cost" of n matvecs. Thus, in this model, an algorithm is considered "efficient" if it beats this trivial cost of n matvecs. In this part of the thesis, we will seek algorithms that produce approximate solutions but will expend a number of matvecs that is *independent* of the dimensions m and n of the matrix.

Problems that are trivial for a matrix B stored in memory become more challenging when considered in the matvec model. For instance, consider the trace

$$\mathrm{tr}(B) = \sum_{i=1}^{n} b_{ii}$$

of a square matrix $B \in \mathbb{R}^{n \times n}$. If B is stored in memory, the trace can be computed exactly in $O(n)$ operations. However, in the matvec model, computing the trace *exactly*—by either a deterministic or randomized algorithm—incurs the trivial cost of n matvecs. This observation motivates the study of algorithms for *approximating* the trace in the matvec model. We will discuss trace estimation in the matvec model throughout this part of the thesis.

Trace estimation is a member of the family of *matrix attribute estimation problems*:

Matrix attribute estimation problem. Let $Q(B)$ denote some attribute of a matrix such as its trace $\mathrm{tr}(B)$ or its entry b_{ij} in position (i, j). The matrix attribute estimation problem is to compute an estimate $\widehat{Q}$ for $Q(B)$. The cost an algorithm is measured by the number of matvecs, and the quality of the solution is measured by the error $|\widehat{Q} - Q(B)|$.

We have written $Q(B)$ for a scalar-valued attribute $Q(B) \in \mathbb{K}$, but vector-valued or matrix-valued attributes $q(B) \in \mathbb{K}^d$ or $Q(B) \in \mathbb{K}^{m' \times n'}$ are fine as well. In these cases, we will measure the error using an appropriate norm.

In a sense, any linear algebra problem is an example of a matrix attribute estimation problem. In this thesis, we will use this term more narrowly to describe attributes like

the trace that are linear or quadratic functions of the matrix $\boldsymbol{B}$ and are computationally challenging only within the matvec model.

As we will see in this part of the thesis, the matvec model provides a convenient framework for designing and analyzing algorithms. However, like all computational models, the matvec model is idealized. We note a few weaknesses:

- **Post-processing costs.** In the matvec model, we measure the cost of an algorithm only by counting the number of matvecs. However, for problems where the matvec operations are relatively cheap, the computational cost can be dominated by post-processing to assemble the collected matvecs into the estimator. Responding to this weakness, this thesis will focus on algorithms with fast post-processing. We will even develop improved implementations of one estimator with up to 7× faster post-processing (Remark 14.5).

- **Blocking.** Another deficit of measuring algorithm cost through matvec count is that it ignores the computational benefits of blocking. For reasons discussed in Section 8.1, performing ℓ matvecs $\boldsymbol{B}\boldsymbol{\omega}_1, \ldots, \boldsymbol{B}\boldsymbol{\omega}_\ell$ individually can be much more expensive than performing these matvecs in a batch, i.e., by computing the matrix–matrix product $\boldsymbol{B}[\boldsymbol{\omega}_1 \quad \cdots \quad \boldsymbol{\omega}_\ell]$. The algorithms in this thesis will always compute matvecs in batches, so they remain efficient even accounting for the benefits of blocking.

- **Opening the black box.** The matvec model treats the matvec subroutine $\boldsymbol{\omega} \mapsto \boldsymbol{B}\boldsymbol{\omega}$ as a black box. But, for matrix functions $\boldsymbol{B} = f(\boldsymbol{C})$, matvecs $f(\boldsymbol{C})\boldsymbol{\omega}$ are often computed by the Arnoldi or Lanczos methods, which utilize matvecs with $\boldsymbol{C}$. In this context, matvecs with $\boldsymbol{C}$ are the primitive operation that should be minimized. Tyler Chen and collaborators have productively used this observation to develop faster "Krylov-aware" matrix attribute estimation algorithms [CH23; PCM25].

Notwithstanding these limitations, we will use the matvec model in this thesis.

12.2 Examples of matrix attribute estimation problems: Trace, diagonal, and row-norm estimation

There are several interesting and practically useful matrix attribute estimation problems, including trace, diagonal, and row-norm estimation.

Trace estimation

The trace estimation problem is to approximate $\mathrm{tr}(\boldsymbol{B})$ in the matrix–vector product (matvec) model. Trace estimation has a number of applications, some of which were surveyed in the paper [US17]. Here is a partial list:

- **Triangle counting.** Counting the number of triangles in a graph is a basic problem in network science, and it is necessary to compute the clustering coefficient [AD18]. This problem is a trace estimation problem, as the number of triangles in a graph with adjacency matrix $\boldsymbol{A}$ is $\mathrm{tr}(\boldsymbol{A}^3)/6$. This problem is a natural demonstration of the matvec model, as matvecs with $\boldsymbol{A}^3$ can be computed by iterated multiplication $\boldsymbol{A}(\boldsymbol{A}(\boldsymbol{A}\boldsymbol{\omega}))$.

- **Log-determinant [MMB21; WPHC+22].** Estimates of the log-determinant are used in maximum likelihood estimation for Gaussian process methods (Remark 5.22). This problem can be viewed as a trace estimation problem, in view of the identity $\log\det(\boldsymbol{A}) = \mathrm{tr}\log(\boldsymbol{A})$. Matvecs with $\log(\boldsymbol{A})$ can be computed using the Lanczos method.

- **Continuous normalizing flows [CRBD18; GCBS+19; SSKK+21; Now22].** Several machine learning models—such as neural ODEs [CRBD18], FFJORD [GCBS+19], and diffusion models [SSKK+21]—evolve a random initial value $\boldsymbol{x}(0)$ under continuous dynamics $\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{x}(t) = f(\boldsymbol{x}(t), t)$. These algorithms must estimate the instantaneous rate of change of the log-likelihood of $\boldsymbol{x}(t)$, which is the negative trace of the Jacobian $-\mathrm{tr}(\mathrm{D}f)$. Matvecs with the Jacobian can be computed using automatic differentiation.

- **Statistical physics.** The partition function of a quantum system with Hamiltonian $\boldsymbol{A}$ at inverse-temperature $\beta > 0$ is $z \coloneqq \mathrm{tr}\exp(-\beta\boldsymbol{A})$. Other thermodynamics quantities can also be expressed as matrix traces; for instance, the average energy is $e \coloneqq z^{-1}\mathrm{tr}(\boldsymbol{A}\exp(-\beta\boldsymbol{A}))$. Trace-exponentials also appear in network science as Estrada index centrality measures [Est22].

- **Norm estimation.** Computing or estimating the norm of a matrix is a ubiquitous task in matrix computations. This computation can be seen as a trace estimation problem, as the Schatten p-norm is $\|\boldsymbol{B}\|_{\mathrm{S}_p} \coloneqq \mathrm{tr}[(\boldsymbol{B}^*\boldsymbol{B})^{p/2}]$. Matvecs with $(\boldsymbol{B}^*\boldsymbol{B})^{p/2}$ can be computed by iterated multiplication (if p is an even integer) or by the Lanczos method.

Diagonal estimation

Given a square matrix $\boldsymbol{B} \in \mathbb{K}^{n \times n}$, the diagonal estimation problem is to estimate $\mathbf{diag}(\boldsymbol{B}) \in \mathbb{K}^n$. There are several applications of diagonal estimation algorithms.

- **Statistics.** The diagonal entries of a covariance matrix $\boldsymbol{A}$ for a random vector $\boldsymbol{x} \in \mathbb{K}^n$ are the variances of the individual entries $a_{ii} = \mathrm{Var}(x_i)$. Similarly, *for jointly Gaussian random variables*, the *reciprocals* of the diagonal of the inverse-covariance matrix (precision matrix) $\boldsymbol{A}^{-1}$ store the *conditional variances*:

$$\frac{1}{\boldsymbol{A}^{-1}(i,i)} = \mathrm{Var}(x_i \mid x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n).$$

- **Centrality measures.** Many centrality measures for graphs are defined using the diagonal of functions of the graph adjacency matrix $\boldsymbol{A}$ (or related matrices such as the graph Laplacian). For instance, the number of triangles incident on vertex i is the ith diagonal entry of $\boldsymbol{A}^3/2$ [AD18], and the *subgraph centrality* of vertex i is the ith diagonal entry of $\exp(\boldsymbol{A})$ [Est22]. For a survey on centrality measures, see [BJT23].

- **Optimization.** The convergence of gradient descent methods can be slow if the problem is poorly scaled. This issue can be remedied by using the diagonal of the Hessian matrix to precondition the descent method [YGSM+21]. Another application of diagonal estimation to semidefinite programming appears in [Lin23].

Other applications of diagonal estimation include electronic structure calculations in materials science [BKS07; LLYC+09], uncertainty quantification for linear regression [Pop23], low-rank matrix approximation algorithms [FL24]. Chapter 16 discusses diagonal estimation algorithms.

Row-norm estimation

The problem of estimating the (squared) row- or column-norms of a matrix $\boldsymbol{B} \in \mathbb{K}^{m \times n}$ also has many applications. This problem has received significantly less attention than trace or diagonal estimation. Assuming matvec access to both $\boldsymbol{B}$ and $\boldsymbol{B}^*$, the row- and column-norm problems are equivalent, as column-norm estimation on $\boldsymbol{B}$ is row-norm estimation on $\boldsymbol{B}^*$. Here are several applications:

- **Leverage scores.** As introduced in Definition 3.15, the leverage scores of a matrix A are the squared row norms of any orthonormal basis matrix U for range(A). Row-norm estimation algorithms play a crucial role in fast algorithms for leverage score estimation [DMMW12]. Related ideas are used to estimate effective resistances in networks [SS08].

- **Matrix computations.** Many standard linear algebra algorithms require computation of row norms, and many of these algorithms are robust to those row norms being computed approximately. For such algorithms, row-norm estimates can allow for significant acceleration (see, e.g., [FL24]). Many of the codes in this thesis require row or column norms, and can also be accelerated using row-norm estimates (see Remark 18.2).

- **Electronic structure theory.** In electronic structure theory calculations, the electron density can be obtained from the row norms of an implicit matrix, making it possible to accelerate the computation using row-norm estimation algorithms. This application appears in the thesis of Aleksandros Sobczyk [Sob24].

Row-norm estimation also has close connections to the diagonal estimation problem (Section 17.2), and many of the most effective algorithms for diagonal estimation for *psd* matrices proceed through row-norm estimation [MMB21; Lin23; FL24]. See Chapter 17 for more discussion of and algorithms for row-norm estimation.

FUNDAMENTAL TOOLS: LOW-RANK AND MONTE CARLO APPROXIMATIONS

> *What we are calling a quadratic trace estimator is often called the Hutchinson's trace estimator, especially when **v** is chosen uniformly from the set of vectors with entries $\pm n^{-1/2}$. However, [Hut89] was not the first use of quadratic trace estimators for the task of approximating the trace of an implicit matrix; [Hut89] itself cites [Gir87] which addresses the same task by using samples of **v** drawn uniformly from the unit hypersphere. Algorithms based on the use of random vectors date back at least to the mid 1970s [ABKS75; WW76; WW77; Dd89].*
>
> Tyler Chen, *Lanczos-based methods for matrix functions* [Che22, §4.1.1]

This chapter introduces two approaches to matrix attribute estimation—Monte Carlo approximation and low-rank approximation—which serve as building blocks for matrix attribute estimation algorithms. After introducing these techniques individually, we will see that they become more powerful when used in combination, as demonstrated in the famous HUTCH++ algorithm. This ideas will be refined in the next section, where we will use them in combination with the *leave-one-out approach* to the design of randomized matrix algorithm.

Sources. The main aims of this section are expository, and it is not based on any particular research article. Section 13.3 discusses variance reduction technique for matrix attribute estimation and HUTCH++ algorithm from the paper [MMMW21b]. The concept of "resphering" matrix attribute estimation algorithms, which we discuss throughout this section, was first introduced in [ETW24].

Outline. Section 13.1 introduces the Monte Carlo approach to matrix attribute estimation, and it shows how this approach can be used to develop Monte Carlo estimators of the trace, diagonal, and row norms. This section also discusses choice of random distribution for such algorithms and introduces the idea of "resphering" a Monte Carlo estimator for a matrix with a known nullspace. Section 13.2 discusses the use of low-rank approximation to estimate matrix attributes, and Section 13.3 describes how low-rank approximation can be used as a *variance reduction* technique

to improve the accuracy of Monte Carlo estimators. This variance technique is exemplified by the Hutch++ algorithm, which is discussed in this section. We conclude Section 13.3 by discussing weaknesses of the Hutch++ algorithm which will be addressed by the leave-one-out approach in Chapter 14.

13.1 Monte Carlo approximation

Monte Carlo approximation is a simple yet powerful method for constructing randomized estimators of a matrix attribute. In its basic form, the approach is to design an unbiased estimator $\widehat{q}$ for a quantity of interest q (so that $\mathbb{E}[\widehat{q}] = q$), and then average several independent copies of $\widehat{q}$ to reduce variance [Liu04]. More general versions of the method also allow for estimators that are not fully independent or that introduce slight bias.

For matrix computations, many Monte Carlo estimators can be built using *isotropic random vectors*.

Definition 13.1 (Isotropic random vector)**.** A random vector $\omega \in \mathbb{K}^n$ is *isotropic* if it satisfies $\mathbb{E}[\omega \omega^*] = \mathbf{I}$.

Examples of isotropic random vectors include standard Gaussian vectors $\omega \sim \text{Normal}_{\mathbb{K}}(\mathbf{0}, \mathbf{I})$ or vectors $\omega \sim \text{Unif}(\sqrt{n}\, \mathbb{S}(\mathbb{K}^n))$ drawn uniformly from the sphere of radius $\sqrt{n}$. Both of these constructions are defined in either the field of real or complex numbers, $\mathbb{K} \in \{\mathbb{R}, \mathbb{C}\}$. Another popular isotropic random distribution is the random sign distribution $\text{Unif}\{\pm 1\}^n$.

An isotropic random vector ω gives rise to a rank-one matrix $\omega \omega^*$ that serves as an unbiased estimator for to the identity matrix. This estimator can be improved by forming an average $s^{-1} \sum_{i=1}^{s} \omega_i \omega_i^*$ of independent copies $\omega_1, \ldots, \omega_s \overset{\text{iid}}{\sim} \omega$. Using this observation, one can design unbiased estimates for matrix attributes by introducing a copy of the identity matrix and replacing it by a stochastic approximation. In the following examples, we develop several classical Monte Carlo estimators in matrix computations using this perspective.

Example: Girard–Hutchinson trace estimator

As a first example of a Monte Carlo algorithm for matrix attribute estimation, we consider the trace estimation problem. Let $\mathbf{B} \in \mathbb{K}^{n \times n}$ be a square matrix. To estimate $\text{tr}(\mathbf{B})$, introduce a copy of the identity matrix

$$\text{tr}(\mathbf{B}) = \text{tr}(\mathbf{B} \cdot \mathbf{I})$$

Program 13.1 `girard_hutchinson.m`. Unoptimized MATLAB implementation of the Girard–Hutchinson estimator (13.1). The `random_signs` subroutine is defined in Program F.2.

```matlab
function tr = girard_hutchinson(B,n,s)
% Input:  Function B() computing matrix products B(X) = B*X, number
%         of rows n, and number of matvecs s
% Output: Estimate tr of trace(B)
tr = 0;
for i = 1:s
    om = random_signs(n,1);  % Generate vector of random signs
    tr = tr + om'*B(om) / s; % Update trace estimate
end

end
```

and use the stochastic approximation $s^{-1} \sum_{i=1}^{s} \omega_i \omega_i^* \approx \mathbf{I}$, resulting in the estimator

$$\widehat{\mathrm{tr}}_{\mathrm{GH}} := \mathrm{tr}\left(\mathbf{B} \cdot \frac{1}{s} \sum_{i=1}^{s} \omega_i \omega_i^*\right) \approx \mathrm{tr}(\mathbf{B} \cdot \mathbf{I}) = \mathrm{tr}(\mathbf{B}).$$

Invoking the linearity and the cyclic property of the trace, we can rewrite the estimator $\widehat{\mathrm{tr}}_{\mathrm{GH}}$ in the more computationally useful form

$$\widehat{\mathrm{tr}}_{\mathrm{GH}} = \frac{1}{s} \sum_{i=1}^{s} \omega_i^* \left(\mathbf{B}\omega_i\right). \tag{13.1}$$

The formula (13.1) clearly demonstrates that $\widehat{\mathrm{tr}}_{\mathrm{GH}}$ can be computed using s matvecs. See Program 13.1 for un-optimized MATLAB code.

Estimators of the form (13.1) were used by Girard [Gir87; Gir89] and popularized by Hutchinson [Hut89], so we will call this approximation the *Girard–Hutchinson estimator*. Tyler Chen has conducted an in-depth study of the history of trace estimation. His analysis identifies estimators related to (13.1) in work from quantum physics dating back as early as the 1920s [Che22, §4.1.1].

Example: BKS diagonal estimator

Another Monte Carlo algorithm for a matrix attribute estimation problem is the BKS diagonal estimator. Again, we begin with a square matrix $\mathbf{B}$ and write $\mathbf{B} = \mathbf{B} \cdot \mathbf{I}$. Replacing $\mathbf{I}$ with the stochastic approximation $s^{-1} \sum_{i=1}^{s} \omega_i \omega_i^* \approx \mathbf{I}$ yields the diagonal estimator

$$\widehat{\mathbf{diag}}_{\mathrm{BKS}} := \mathbf{diag}\left(\mathbf{B} \cdot \frac{1}{s} \sum_{i=1}^{s} \omega_i \omega_i^*\right) \approx \mathbf{diag}(\mathbf{B}). \tag{13.2}$$

Program 13.2 `bks.m`. Bekas–Kokiopoulou–Saad estimator for the diagonal of a matrix. Subroutine `random_signs` is provided in Program F.2.

```
function d = bks(B,n,s)
% Input:  Function B() computing matrix products B(X) = B*X,
%         number of columns n and number of matvecs s
% Output: Estimate d of the diagonal of B

Om = random_signs(n,s);    % Matrix of random signs
d = mean(B(Om) .* Om,2);   % BKS diagonal estimate

end
```

This estimator can be simplified by means of the identity

$$\mathbf{diag}(ab^*) = a \odot \overline{b} \quad \text{for } a, b \in \mathbb{K}^n. \tag{13.3}$$

Here, $\odot$ denotes the entrywise product of vectors, and $\overline{(\cdot)}$ denotes the entrywise complex conjugate. Applying (13.3) and the linearity of the diagonal map yields a simpler form for the diagonal estimator (13.2):

$$\widehat{\mathbf{diag}}_{\text{BKS}} = \frac{1}{s} \sum_{i=1}^{s} (B\omega_i) \odot \overline{\omega_i}. \tag{13.4}$$

Estimators similar to (13.4) appear to have originally developed by Bekas, Kokiopoulou, and Saad [BKS07]. In this work, we will call $\widehat{\mathbf{diag}}_{\text{BKS}}$ the *BKS diagonal estimator*. Code for the BKS diagonal estimator appears in Program 13.2. We discuss diagonal estimation more thoroughly in Chapter 16.

Example: The Johnson–Lindenstrauss row-norm estimator

As a final example, we consider the problem of estimating the row norms of a *possibly rectangular* matrix $B \in \mathbb{K}^{m \times n}$. Row-norm estimation is an interesting problem because the row norms are *nonlinear* functions of the matrix. We shall focus on developing unbiased estimates for the *squared* row norms of B, which we denote

$$\mathbf{srn}(B) := \left(\|B(i, :)\|^2 : 1 \leq i \leq m \right) \in \mathbb{R}_+^m. \tag{13.5}$$

To fashion an estimator for $\mathbf{srn}(B)$, we first invoke the identity

$$\mathbf{srn}(B) = \mathbf{diag}(BB^*).$$

Now insert a copy of the identity matrix in the middle of the matrix product $BB^* = B \cdot I \cdot B^*$, and use the stochastic approximation $s^{-1} \sum_{i=1}^{s} \omega_i \omega_i^* \approx I$, resulting in the

Program 13.3 `jl_rownorm.m`. Johnson–Lindenstrauss estimator for the (squared) row norms of a matrix. Subroutines `random_signs` and `sqrownorms` re provided in Programs F.2 and F.3.

```
function srn = jl_rownorm(B,n,s)
% Input:  Function B() computing matrix products B(X) = B*X,
%         number of columns n and number of matvecs s
% Output: Estimate srn of the squared row norms of B

Om = random_signs(n,s);    % Matrix of random signs
srn = sqrownorms(B(Om))/s; % JL row norm estimate

end
```

Monte Carlo estimator

$$\widehat{\mathbf{srn}}_{\mathrm{JL}} := \mathbf{diag}\left(\mathbf{B} \cdot \frac{1}{s} \sum_{i=1}^{s} \omega_i \omega_i^* \cdot \mathbf{B}^*\right) \approx \mathbf{srn}(\mathbf{B}).$$

This estimator can more conveniently be written as

$$\widehat{\mathbf{srn}}_{\mathrm{JL}} = \frac{1}{s} \sum_{i=1}^{s} |\mathbf{B}\omega_i|^2, \tag{13.6}$$

where the function $|\cdot|^2$ denotes the squared modulus, evaluated entrywise for a vector input. We call this estimator the *Johnson–Lindenstrauss row-norm estimator*, as it can be analyzed using the Johnson–Lindenstauss lemma [JL84]. Code is provided in Program 13.3. We discuss row-norm estimation more in Chapter 17.

Which isotropic vector to use?

Having seen that isotropic random vectors can be used to build Monte Carlo estimators of matrix attributes, we now discuss the choice of which isotropic distribution to use.

Let us first catalog the popular options, many of which have separate definitions over the fields $\mathbb{K} = \mathbb{R}$ and $\mathbb{K} = \mathbb{C}$ of real and complex numbers:

- **Gaussian.** A standard Gaussian vector $\omega \sim \mathrm{Normal}_{\mathbb{K}}(\mathbf{0}, \mathbf{I})$ is isotropic. Its entries are independent and drawn from the real or complex standard Gaussian distribution. (Recall that a complex standard Gaussian random variable $g \sim \mathrm{Normal}_{\mathbb{C}}(0, 1)$ takes the form $g = (g_1 + ig_2)/\sqrt{2}$, where $g_1, g_2 \sim \mathrm{Normal}_{\mathbb{R}}(0, 1)$ are independent real standard Gaussians.)

- **Sphere.** A random vector $\omega \sim \text{Unif}(\sqrt{n}\mathbb{S}(\mathbb{K}^n))$ is isotropic. One can be generated by drawing a standard Gaussian vector $\boldsymbol{g} \in \text{Normal}_{\mathbb{K}}(\boldsymbol{0}, \mathbf{I})$ and scaling it to have constant length $\omega \coloneqq \sqrt{n}/\|\boldsymbol{g}\| \cdot \boldsymbol{g}$.

- **Random signs.** A vector of random signs $\omega \sim \text{Unif}\{\pm 1\}^n$ is isotropic. This distribution has very low resource requirements, requiring only n independent uniformly random *bits* to generate. The random sign distribution is often called the *Rademacher distribution*.

- **Random phases.** An analog of the random sign distribution for the complex field is a vector of random phases $\omega \sim \text{Unif}(\mathbb{T}(\mathbb{C})^n)$, defined to be a random vector whose entries are independent and drawn uniformly from the complex unit circle $\mathbb{T}(\mathbb{C}) \coloneqq \{\phi \in \mathbb{C} : |\phi| = 1\}$. The random phase distribution is also called the *Steinhaus distribution*.

  Note that the random signs and random phases distributions can be unified $\omega \sim \text{Unif}(\mathbb{T}(\mathbb{K})^n)$ by defining the "unit circle" of the real numbers to be $\mathbb{T}(\mathbb{R}) \coloneqq \{\phi \in \mathbb{R} : |\phi| = 1\} = \{\pm 1\}$.

- **Random coordinate.** Last, one can also generate an isotropic random vector by drawing a random standard basis element $\omega \sim \text{Unif}\{\sqrt{n}\mathbf{e}_i : 1 \leq i \leq n\}$, appropriately rescaled.

Which of these options should one use? To answer this question with some degree of precision, we will focus on which of these random vectors to use for estimating the trace of a real symmetric matrix $A \in \mathbb{R}^{n \times n}$ using the Girard–Hutchinson trace estimator, although the same principles apply for other matrix attribute estimation problems and algorithms as well.

Recall that the Girard–Hutchinson estimator for A is defined as

$$\widehat{\text{tr}}_{\text{GH}} \coloneqq \frac{1}{m} \sum_{i=1}^{m} \omega_i^* A \omega_i \quad \text{for } \omega_1, \ldots, \omega_m \overset{\text{iid}}{\sim} \omega.$$

We will compare the accuracy of this estimator with several random isotropic vectors ω by using the mean-squared error $\mathbb{E}(\widehat{\text{tr}}_{\text{GH}} - \text{tr}(A))^2$. Because the terms $\omega_i^* A \omega_i$ are iid and unbiased estimators for $\text{tr}(A)$, the mean-squared error is

$$\mathbb{E}(\widehat{\text{tr}}_{\text{GH}} - \text{tr}(A))^2 = \frac{1}{m} \text{Var}(\omega^* A \omega). \tag{13.7}$$

In particular, we see that the mean-squared error decays at a $O(1/m)$ rate regardless of the choice of isotropic test vector ω. The $O(1/m)$ convergence rate in the mean-squared error is typical of Monte Carlo methods. The quality of a distribution for trace estimation is quanitified by the prefactor $\mathrm{Var}(\omega^* A \omega)$, which depends on the choice of test vector ω.

Fact 13.2 (Girard–Hutchinson estimator: Variance formulas). *Let $A \in \mathbb{R}^{n \times n}$ be a* real *symmetric matrix with eigenvalues $\lambda_1, \ldots, \lambda_n \in \mathbb{R}$. Denote the mean eigenvalue $\overline{\lambda} := n^{-1} \operatorname{tr}(A) = n^{-1} \sum_{i=1}^n \lambda_i$ and the mean diagonal element $\overline{a} := n^{-1} \sum_{i=1}^n a_{ii}$. The following equations give the variance of the basic Girard–Hutchinson estimate $\omega^* A \omega$ for several choices for the isotropic random vector $\omega \in \mathbb{C}^n$:*

$$\mathrm{Var}(\omega^* A \omega) = \mathrm{C}_{\mathbb{K}} \|A\|_{\mathrm{F}}^2 \qquad = \mathrm{C}_{\mathbb{K}} \sum_{i=1}^n \lambda_i^2 \qquad \textit{for } \omega \sim \mathrm{N{\scriptsize ORMAL}}_{\mathbb{K}}(\mathbf{0}, \mathbf{I}),$$

$$\mathrm{Var}(\omega^* A \omega) = K_{\mathbb{K}} \left\| A - \overline{\lambda} \mathbf{I} \right\|_{\mathrm{F}}^2 \qquad = K_{\mathbb{K}} \sum_{i=1}^n (\lambda_i - \overline{\lambda})^2 \quad \textit{for } \omega \sim \mathrm{U{\scriptsize NIF}}(\sqrt{n}\mathbb{S}(\mathbb{K}^n)),$$

$$\mathrm{Var}(\omega^* A \omega) = \mathrm{C}_{\mathbb{K}} \sum_{i \neq j} a_{ij}^2 \qquad\qquad \textit{for } \omega \sim \mathrm{U{\scriptsize NIF}}(\mathbb{T}(\mathbb{K})^n),$$

$$\mathrm{Var}(\omega^* A \omega) = n^2 \sum_{i=1}^n (a_{ii} - \overline{a})^2 \qquad\qquad \textit{for } \omega \sim \mathrm{U{\scriptsize NIF}}\{\sqrt{n}\mathbf{e}_i\}_{1 \leq i \leq n}.$$

The prefactors are $\mathrm{C}_{\mathbb{R}} = 2$, $\mathrm{C}_{\mathbb{C}} = 1$, $K_{\mathbb{R}} = 2n/(n+2)$, and $K_{\mathbb{C}} = n/(n+1)$. These equalities become upper bounds for a real nonsymmetric matrix.

These variance formulas are standard, and I have collected the simplest proofs of these formulas I know in [Epp23b]. The random phase distribution $\mathrm{U{\scriptsize NIF}}(\mathbb{T}(\mathbb{K})^n)$ and sphere distribution $\mathrm{U{\scriptsize NIF}}(\sqrt{n}\mathbb{S}(\mathbb{K}^n))$ are known to be optimal in certain senses, and these optimality results are also discussed in [Epp23b]. We may draw several conclusions from these formulas:

The Gaussian distribution is dominated. The variance for the Gaussian distribution $\omega \sim \mathrm{N{\scriptsize ORMAL}}_{\mathbb{K}}(\mathbf{0}, \mathbf{I})$ is higher than both the sphere and random sign/phase distributions. Compared to the sphere distribution, the Gaussian distribution has a variance depending on the *aggregate size* of A's eigenvalues, whereas the sphere distribution depends only on the *spread* of A's eigenvalues. This size/spread distinction can make a big difference for a matrix A with large eigenvalues that are tightly clustered. Compared to the random phase distribution, the variance for the Gaussian distribution depends on Frobenius norm $\|A\|_{\mathrm{F}}^2 = \sum_{i,j} a_{ij}^2$, which reflects the magnitude of

all of A's entries. By contrast, the variance of the random sign/phase distribution depends on $\sum_{i\neq j} a_{ij}^2$, which omits the contribution from A's diagonal entries. This property makes the random sign/phase distribution especially effective for matrices with a heavy diagonal. I describe the case against Gaussians for stochastic trace estimation in more detail and provide numerical evidence in [Epp24a].

Sphere vs. signs/phases. The sphere and sign/phase distributions both dominate the Gaussian distribution, but which should be preferred? Ultimately, both choices of distribution are effective, and we will use both in this thesis. The random sign/phase distribution has the benefit that it ignores the influence of the diagonal of A, making it effective for matrices with a heavy diagonal. The random sphere distribution, however, is known to be (minimax) optimal for matrices A drawn from unitarily invariant families, such as the class of all symmetric matrices with $\|A\|_F \leq 1$. This optimality result appears to have first been discovered in unpublished work of Richard Kueng [Tro20, Prob. 1.23]; see [Epp23b] for a proof.

Real vs. complex. The variance formulas in Fact 13.2 show that using complex-valued test vectors ω results in a variance about half that of their real-valued counterparts. However, when applied to real matrices A, this benefit is typically offset by the increased computational cost of using complex arithmetic. Further, existing codes for evaluating the matvec operation might not be compatible with complex data. As such, the reduced variance of complex-valued test vectors is usually not worth the additional cost when the matrix A is real-valued. For a complex-valued matrix $B \in \mathbb{C}^{n\times n}$, however, it is generally preferable to use complex-valued test vectors ω.

Coordinate sampling is often dangerous. The variance for trace estimation with random coordinate vectors is often much higher than alternate approaches. This can be seen by a simple "back of the envelope" computation: Consider a matrix A where all the entries are roughly of unit magnitude $|a_{ij}| \approx 1$, with a similar number of positive and negative entries, so that the average of diagonal entries is small: $\overline{a} \approx 0$. Then the variance of the Girard–Hutchinson estimator with random coordinate sampling has scaling $\sim n^3$, whereas the other isotropic distributions in Fact 13.2 have a variance of $\sim n^2$. Thus, in view of (13.7), the coordinate sampling distribution may require up to n matvecs to achieve accuracy comparable to what other estimators can achieve with a single matvec. This is a dismal state of affairs, particular since the trace estimation problem can be solved exactly in n matvecs. Random coordinate vectors do have their uses, particularly in computational models

more restrictive than the matvec model [BKM22], but they are best avoided except in special situations.

Resphering: Improved Monte Carlo for rank-deficient matrices

So far, we have designed Monte Carlo methods for matrix attribute estimation based on introducing the identity matrix $B = B \cdot I$ and replacing I with an unbiased stochastic approximation. As we will see shortly, we will have many occasions to apply Monte Carlo approximations to a rank-deficient matrix B with a known (right or left) nullspace. In such cases, we can use *resphering* to obtain lower variance Monte Carlo estimators. Here is the main definition:

Definition 13.3 (Isotropic random vector on a subspace)**.** Let $\mathsf{U} \subseteq \mathbb{K}^n$ be a subspace over the real or complex numbers, and let $\mathbf{\Pi}$ denote the orthoprojector on U. A random vector $\mathbf{v}$ is said to be *isotropic on* U if $\mathbb{E}[\mathbf{v}\mathbf{v}^*] = \mathbf{\Pi}$.

For any subspace U of dimension s, a uniformly random vector $\mathbf{v} \sim \mathrm{U_{NIF}}(\sqrt{s}\,\mathbb{S}(\mathsf{U}))$ from the sphere of radius $\sqrt{s}$ in U is isotropic on U. This will be our only example of an isotropic vector on a subspace in this thesis.

To see how isotropic random vectors on a subspace can be employed, let $B \in \mathbb{C}^{n\times n}$ be a square matrix and suppose we have access to a matrix $Q \in \mathbb{C}^{n\times k}$ with orthonormal columns for which $BQ = 0$. To design a trace estimator for B, write

$$B = B \cdot \mathbf{\Pi} \quad \text{for } \mathbf{\Pi} := I - QQ^*.$$

To approximate $\mathbf{\Pi}$, generate isotropic random vectors $\mathbf{v}_1, \ldots, \mathbf{v}_s$ on the subspace $\mathrm{range}(Q)^\perp$ and introduce the stochastic approximation $s^{-1}\sum_{i=1}^s \mathbf{v}_i\mathbf{v}_i^*$. This leads to the *resphered Girard–Hutchinson trace estimator*

$$\widehat{\mathrm{tr}}_{\mathrm{SGH}} := \mathrm{tr}\left(B \cdot \frac{1}{s}\sum_{i=1}^s \mathbf{v}_i\mathbf{v}_i^*\right) = \frac{1}{s}\sum_{i=1}^s \mathbf{v}_i^*\left(B\mathbf{v}_i\right).$$

To generate isotropic random vectors

$$\mathbf{v} \sim \mathrm{U_{NIF}}\left(\sqrt{n-k}\cdot\mathbb{S}(\mathrm{range}(Q)^\perp)\right)$$

on the subspace $\mathrm{range}(Q)^\perp$ proceed as follows: Draw a random vector ω from the normal or sphere distribution, orthogonalize it against Q, and rescale to the proper norm

$$\mathbf{v} := \frac{\sqrt{n-k}}{\|(I - QQ^*)\omega\|} \cdot (I - QQ^*)\omega = \frac{\sqrt{n-k}}{\|\omega - Q(Q^*\omega)\|} \cdot (\omega - Q(Q^*\omega)). \quad (13.8)$$

We call the process of replacing ω by ν in this way *resphering*.

When applied to a rank-deficient matrix $\boldsymbol{B}$ whose nonzero eigenvalues are clustered, the resphered Girard–Hutchinson estimator can lead to significantly lower variance than the standard Girard–Hutchinson estimator. Here is an example result.

Corollary 13.4 (Resphered Girard–Hutchinson estimator: Variance)**.** *Let $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ be a symmetric matrix with at most r nonzero eigenvalues $\lambda_1, \ldots, \lambda_r$, let $\overline{\lambda} :=$ $r^{-1} \sum_{i=1}^{r} \lambda_i$ denote their average, and let $\nu \sim \mathrm{U_{NIF}}(\sqrt{r} \cdot \mathbb{S}(\mathrm{range}(\boldsymbol{A})))$ be a test vector. Then the resphered Girard–Hutchinson estimator $\nu^* \boldsymbol{A} \nu$ has variance*

$$\mathrm{Var}(\nu^* \boldsymbol{A} \nu) = \frac{2r}{r+2} \sum_{i=1}^{r} (\lambda_i - \overline{\lambda})^2.$$

Excepting the trivial case when $\lambda_i = 0$ for all i, the variance $\mathrm{Var}(\nu^* \boldsymbol{A} \nu)$ is always strictly smaller than for the plain Girard–Hutchinson estimator $\mathrm{Var}(\omega^* \boldsymbol{A} \omega)$ with $\omega \sim \mathrm{U_{NIF}}(\sqrt{n} \cdot \mathbb{S}(\mathbb{R}^n))$.

The technique now called resphering was first introduced in [ETW24], where it appeared under the name "normalization." In this thesis, I adopt the more descriptive term "resphering," a name suggested to me by Joel Tropp.

13.2 Low-rank approximation

Low-rank approximation provides another paradigm for solving matrix attribute estimation problems. As we saw in Chapter 2, we can cheaply compute near-optimal low-rank approximations to a general matrix $\boldsymbol{B}$ using the randomized SVD (Section 2.3) or to a psd matrix $\boldsymbol{A}$ using Nyström approximation (Section 2.5).

Low-rank approximation gives a natural approach to any matrix attribute estimation problem. To estimate an attribute $Q(\boldsymbol{B})$ of a matrix $\boldsymbol{B} \in \mathbb{C}^{m \times n}$, simply replace $\boldsymbol{B}$ by a low-rank approximation $\widehat{\boldsymbol{B}}$ and use the $Q(\widehat{\boldsymbol{B}})$ as an estimator for $Q(\boldsymbol{B})$. This use of low-rank approximation in this way was proposed by Saibaba, Alexanderian, and Ipsen for trace estimation [SAI17].

The quality of the approximation $Q(\widehat{\boldsymbol{B}}) \approx Q(\boldsymbol{B})$ is dictated by the quality of the low-rank approximation $\boldsymbol{B} \approx \widehat{\boldsymbol{B}}$ and, consequently, the rate of singular value decay in the matrix $\boldsymbol{B}$ (recall Facts 2.5 and 2.11). For matrices with rapid singular value decay, low-rank approximation-based estimators can produce highly accurate results. On the other hand, for matrices with slow singular value decay, plain low-rank approximation based estimators are often wholly inaccurate. This inconsistent

performance makes pure low-rank approximation based estimators only situationally useful.

13.3 Combining Monte Carlo and low-rank approximation

The Monte Carlo and low-rank approximation approaches can be combined to achieve better and more consistent results than either approach yields by itself.

Let us illustrate by deriving a simplified version of the Hutch++ algorithm for trace estimation [MMMW21a]. Suppose we wish to estimate the trace of a matrix $B \in \mathbb{C}^{n \times n}$ and are given a fixed budget of s matvecs to accomplish the task. Assume, for simplicity, that s is divisible by 3. Begin by running the randomized SVD (Section 2.3) to compute a matrix $Q \in \mathbb{C}^{n \times (s/3)}$, which defines a rank-$(s/3)$ approximation $\widehat{B} = QQ^*B$ to B. By linearity, we may decompose the trace of B:

$$\operatorname{tr}(B) = \operatorname{tr}(\widehat{B}) + \operatorname{tr}(B - \widehat{B}) = \operatorname{tr}(QQ^*B) + \operatorname{tr}((I - QQ^*)B).$$

The first term, $\operatorname{tr}(\widehat{B}) = \operatorname{tr}(QQ^*B)$, can be computed exactly by forming Q^*B. To estimate the second term, we employ a Monte Carlo method, specifically the Girard–Hutchinson estimator:

$$\operatorname{tr}(B - \widehat{B}) \approx \frac{1}{s/3} \sum_{i=1}^{s/3} \gamma_i^* \left((I - QQ^*) (B\gamma_i) \right)$$

Here, $\gamma_1, \ldots, \gamma_{s/3}$ denote freshly generated isotropic random vectors, independent of each other and the random test matrix used to execute the randomized SVD. Combining the exact computation of the first term and the stochastic approximation of the second yields the (simplified) Hutch++ estimator

$$\widehat{\operatorname{tr}}_{\mathrm{SH++}} := \operatorname{tr}(Q(Q^*B)) + \frac{1}{s/3} \sum_{i=1}^{s/3} \gamma_i^* \left((I - QQ^*) (B\gamma_i) \right) \quad \text{for } Q = \mathbf{Orth}(B\Omega).$$

(13.9)

The computational cost of the simplified Hutch++ estimator is s matvecs ($2s/3$ with B and $s/3$ with B^*) plus $O(s^2 n)$ additional arithmetic operations.

The full Hutch++ estimator contains two optimizations over the simplified version (13.9):

1. In some contexts, matvecs with B^* are expensive or are entirely unavailable. To avoid matvecs with B^*, we use the cyclic property of the trace to write

$$\operatorname{tr}(\widehat{B}) = \operatorname{tr}(Q(Q^*B)) = \operatorname{tr}(Q^*(BQ)).$$

Program 13.4 `hutchpp.m`.   Hutch++ algorithm for trace estimation.   The `random_signs` subroutine is defined in Program F.2.

```matlab
function tr = hutchpp(B,n,s)
% Input:  Function B() computing matrix products B(X) = B*X, number
%         of rows n, and number of matvecs s
% Output: Estimate tr of trace(B)

k = floor(s/3);                      % Rank is s/3
Om = randn(n,k);                     % Random Gaussian matrix
Y = B(Om);                           % Collect matvecs
[Q,~] = qr(Y,"econ");                % Randomized SVD
BQ = B(Q);                           % Collect matvecs

k = s-2*k;                           % Remaining matvecs
Ga = random_signs(n,k);              % Random sign vectors
X = Ga - Q*(Q'*Ga);                  % Orthogonalize against Q
BX = B(X);                           % Collect matvecs
tr = trace(Q'*BQ) + trace(X'*BX) / k; % Hutch++ estimator

end
```

The expression $\mathrm{tr}(\boldsymbol{Q}^*(\boldsymbol{BQ}))$ can be evaluated using only matvecs with $\boldsymbol{B}$, removing matvecs with $\boldsymbol{B}^*$ from the Hutch++ algorithm entirely.

2. The matrix $\mathbf{I} - \boldsymbol{QQ}^*$ is an orthoprojector and thus satisfies $\mathbf{I} - \boldsymbol{QQ}^* = (\mathbf{I} - \boldsymbol{QQ}^*)^2$. Therefore, the residual trace can be written more symmetrically as

$$\mathrm{tr}(\boldsymbol{B} - \widehat{\boldsymbol{B}}) = \mathrm{tr}((\mathbf{I} - \boldsymbol{QQ}^*)\boldsymbol{B}) = \mathrm{tr}((\mathbf{I} - \boldsymbol{QQ}^*)^2 \boldsymbol{B}) = \mathrm{tr}((\mathbf{I} - \boldsymbol{QQ}^*)\boldsymbol{B}(\mathbf{I} - \boldsymbol{QQ}^*)).$$

The symmetrically projected matrix $(\mathbf{I} - \boldsymbol{QQ}^*)\boldsymbol{B}(\mathbf{I} - \boldsymbol{QQ}^*)$ always has a smaller Frobenius norm than the one-sided projection $(\mathbf{I} - \boldsymbol{QQ}^*)\boldsymbol{B}$, so we expect a smaller error in applying the Girard–Hutchinson estimator to the former matrix rather than the latter (cf. Fact 13.2).

Combining these two optimizations yields the standard Hutch++ estimator

$$\widehat{\mathrm{tr}}_{\mathrm{H++}} := \mathrm{tr}(\boldsymbol{Q}^*(\boldsymbol{BQ})) + \frac{1}{s/3} \sum_{i=1}^{s/3} \boldsymbol{\gamma}_i^* \left((\mathbf{I} - \boldsymbol{QQ}^*) \left(\boldsymbol{B} \left((\mathbf{I} - \boldsymbol{QQ}^*) \boldsymbol{\gamma}_i \right) \right) \right), \quad (13.10)$$

where $\boldsymbol{Q} = \mathbf{Orth}(\boldsymbol{B\Omega})$. Code is provided in Program 13.4.

The combination of low-rank approximation with Monte Carlo for trace estimation and related problems was explored prior to the original Hutch++ paper. Notable

examples include the works of Gambhir, Stathopoulos, and Originos [GSO17] and Lin [Lin17]. Meyer, Musco, Musco, and Woodruff [MMMW21a] crystallized these ideas in the Hutch++ algorithm, provided mathematical analysis, and established lower bounds on the best-possible accuracy for any trace estimation algorithm.

The Hutch++ algorithm illustrates the use of *variance reduction* in Monte Carlo methods [Liu04, §2.3]. Specialized to trace estimation, the key idea of variance reduction is to choose a matrix $\widehat{B}$, called a *control variate*, that closely approximates B and whose trace can be computed exactly. Instead of applying a Monte Carlo estimator to estimate $\mathrm{tr}(B)$ directly, we instead estimate the trace of the residual $B - \widehat{B}$ and add the result to $\mathrm{tr}(\widehat{B})$. This strategy typically yields an estimator with lower variance. In principle, this variance reduction strategy can be applied using any type of approximation $\widehat{B} \approx B$, but low-rank approximations have proven the most effective for trace estimation so far.

Theoretical analysis of Hutch++

Error bounds for Hutch++ can be derived by combining error bounds for the randomized SVD ((2.5)) with variance bounds for the Girard–Hutchinson estimator (Fact 13.2). We will state and prove such a bound here, as it will serve as a useful comparison for the error bounds for the XTrace and XNysTrace estimators developed in the next section; see Section 15.1.

Theorem 13.5 (Hutch++: mean-squared error)**.** *Let $B \in \mathbb{R}^{n \times n}$ be a real matrix and let $\widehat{\mathrm{tr}}_{\mathrm{H++}}$ be the Hutch++ estimator* (13.11) *with a real standard Gaussian test matrix $\Omega \in \mathbb{R}^{n \times (s/3)}$ and iid $\gamma_1, \ldots, \gamma_{s/3} \in \mathbb{R}^n$ drawn from any of the real isotropic distributions from Fact 13.2, except the random coordinate distribution. Then*

$$\mathbb{E}\left(\widehat{\mathrm{tr}}_{\mathrm{H++}} - \mathrm{tr}(B)\right)^2 \leq \frac{6}{s} \cdot \min_{r \leq s/3-2} \frac{s-3}{s-3r-3} \cdot \left\|B - [\![B]\!]_r\right\|_{\mathrm{F}}^2. \tag{13.11}$$

Proof. Using the *reduced matrix*

$$B_{\mathrm{red}} = (\mathbf{I} - QQ^*)B(\mathbf{I} - QQ^*),$$

the error of Hutch++ may be expressed as

$$\widehat{\mathrm{tr}}_{\mathrm{H++}} - \mathrm{tr}(B) = \frac{1}{s/3} \sum_{i=1}^{s/3} \omega_i^* B_{\mathrm{red}} \omega_i - \mathrm{tr}(B_{\mathrm{red}}) = \frac{1}{s/3} \sum_{i=1}^{s/3} (\omega_i^* B_{\mathrm{red}} \omega_i - \mathrm{tr}(B_{\mathrm{red}})).$$

$$\tag{13.12}$$

The right-hand side of (13.12) is a sum of mean-zero random variables that are iid conditional on $\boldsymbol{B}_{\text{red}}$. Therefore,

$$\mathbb{E}\left[(\widehat{\text{tr}}_{\text{H++}} - \text{tr}(\boldsymbol{B}))^2 \,\big|\, \boldsymbol{B}_{\text{red}}\right] = \frac{1}{s/3} \,\text{Var}\left(\boldsymbol{\omega}_1^* \boldsymbol{B}_{\text{red}} \boldsymbol{\omega}_1 \,\big|\, \boldsymbol{B}_{\text{red}}\right).$$

Applying Fact 13.2, we obtain

$$\mathbb{E}\left[(\widehat{\text{tr}}_{\text{H++}} - \text{tr}(\boldsymbol{B}))^2 \,\big|\, \boldsymbol{B}_{\text{red}}\right] \le \frac{6}{s} \cdot \|\boldsymbol{B}_{\text{red}}\|_{\text{F}}^2.$$

Multiplying by an orthoprojector can only reduce the Frobenius norm. Therefore,

$$\|\boldsymbol{B}_{\text{red}}\|_{\text{F}} \le \|(\mathbf{I} - \boldsymbol{Q}\boldsymbol{Q}^*)\boldsymbol{B}\|_{\text{F}}.$$

Combining the two previous displays and taking the expectation, we obtain

$$\mathbb{E}(\widehat{\text{tr}}_{\text{H++}} - \text{tr}(\boldsymbol{B}))^2 \le \frac{6}{s} \cdot \mathbb{E}\|(\mathbf{I} - \boldsymbol{Q}\boldsymbol{Q}^*)\boldsymbol{B}\|_{\text{F}}^2.$$

Invoking the randomized SVD error bound (2.5) establishes the desired result. $\quad\square$

In Theorem 13.5, we assumed the matrix $\boldsymbol{B}$ was real to use Fact 13.2. Versions of this result are easy to derive for complex vectors.

The error bound (13.11) shows that the mean-squared error of HUTCH++ is proportional to $1/s$ times the (squared) Frobenius norm error of the best rank-r approximation, where $r \approx s/3$. A bound similar to Theorem 13.5 appears in [ETW24, Thm. 1.1]; see also [Mey21] and [MMMW21b, §5] in the *arXiv version* of the HUTCH++ paper.

The bound (13.11) describes the mean-squared error of the HUTCH++ method in practice, but it can be a bit imposing. Therefore, it can be informative to derive simplified versions of the bound. Choosing $r = s/6 - 1$ in the minimum (13.11) yields

$$\mathbb{E}\left(\widehat{\text{tr}}_{\text{H++}} - \text{tr}(\boldsymbol{B})\right)^2 \le \frac{12}{s} \cdot \left\|\boldsymbol{B} - [\![\boldsymbol{B}]\!]_{s/6-1}\right\|_{\text{F}}^2. \tag{13.13}$$

To further simplify, we can employ a crude bound on the best rank-r approximation error (see, e.g., [GSTV07, Lem. 7], [MMMW21a, Lem. 13], and [ETW24, Fact 5.5] for versons of this result).

Fact 13.6 (Rank-r approximation error). *For any matrix $\boldsymbol{B} \in \mathbb{K}^{m \times n}$ and $r \ge 1$,*

$$\left\|\boldsymbol{B} - [\![\boldsymbol{B}]\!]_r\right\| \le \frac{\|\boldsymbol{B}\|_*}{r+1} \quad \textit{and} \quad \left\|\boldsymbol{B} - [\![\boldsymbol{B}]\!]_r\right\|_{\text{F}} \le \frac{\|\boldsymbol{B}\|_*}{2\sqrt{r}}.$$

Using Fact 13.6, we can further bound (13.13) as

$$\mathbb{E}\left(\widehat{tr}_{\text{H++}} - \text{tr}(\boldsymbol{B})\right)^2 \leq \frac{72}{s^2} \cdot \|\boldsymbol{B}\|_*^2. \tag{13.14}$$

This result shows that the mean-squared error of Hutch++ is dominated by a quantity that decays at a $O(1/s^2)$ rate, improving on the Monte Carlo $O(1/s)$ rate (13.7) of the Girard–Hutchinson estimator. However, the reader should be aware that this "$O(1/s^2)$ convergence rate" for Hutch++ is easy to misinterpret; see Section 15.2.

As a final simplification, suppose we apply Hutch++ to a psd matrix $\boldsymbol{A}$. In this case, the trace norm and the trace are the same, and (13.14) implies that

$$\left(\mathbb{E}\left(\widehat{tr}_{\text{H++}} - \text{tr}(\boldsymbol{A})\right)^2\right)^{1/2} \leq \varepsilon \, \text{tr}(\boldsymbol{A}) \quad \text{when } s \geq \frac{6\sqrt{2}}{\varepsilon}. \tag{13.15}$$

We obtain a root-mean-squared error of $\varepsilon \, \text{tr}(\boldsymbol{A})$ when the number of matvecs s is $s = O(1/\varepsilon)$. Meyer et al. show that the $s = O(1/\varepsilon)$ parameter complexity is optimal for trace estimation; in particular, no algorithm achieves relative error ε using $s = O(1/\varepsilon^{0.999})$ matvecs for every input matrix $\boldsymbol{A}$ [MMMW21a, §4] (see also [Mey24, §2.3.5]).

Remark 13.7 (High probability bounds). It is straightforward to prove error bounds for Hutch++ that control the error $|\widehat{tr}_{\text{H++}} - \text{tr}(\boldsymbol{B})|$ with high probability by combining high-probability error bounds for the randomized SVD with the Hanson–Wright inequality. For the former, see relevant results in [HMT11; MM20; TW23]. For the latter, see [RV13; Epp22b]. Using this approach, Meyer et al. established the following result [MMMW21a, Thm. 1.1]:

Fact 13.8 (Hutch++: High probability error bound). *When applied to a* real *psd matrix* $\boldsymbol{A}$, *Hutch++ with random sign vectors achieves the guarantee*

$$|\widehat{tr}_{\text{H++}} - \text{tr}(\boldsymbol{A})| \leq \varepsilon \, \text{tr}(\boldsymbol{A}) \quad \text{with probability at least } 1 - \delta \tag{13.16}$$

using $s = O(\varepsilon^{-1} \cdot \sqrt{\log(1/\delta)} + \log(1/\delta))$ *matvecs.*

This result establishes that the probability of failing to produce a trace approximation of relative error ε decreases exponentially in the number of matvecs s. ◇

Improving Hutch++ using resphering

In passing, let us observe that we can enhance the Hutch++ algorithm by using resphering, introduced in Section 13.1. Simply use the resphered Girard–Hutchinson

Program 13.5 `hutchpp_resphere.m`. Hutch++ algorithm with resphering for trace estimation. The `sqcolnorms` subroutine is defined in Program F.4.

```
function tr = hutchpp_resphere(B,n,s)
% Input:  Function B() computing matrix products B(X) = B*X, number
%         of rows n, and number of matvecs s
% Output: Estimate tr of trace(B)

k = floor(s/3);                         % Rank is s/3
Om = randn(n,k);                        % Random Gaussian matrix
Y = B(Om);                              % Collect matvecs
[Q,~] = qr(Y,"econ");                   % Randomized SVD
BQ = B(Q);                              % Collect matvecs

k2 = s-2*k;                             % Remaining matvecs
Ga = randn(n,k2);                       % Gaussian random vectors
X = Ga - Q*(Q'*Ga);                     % Orthogonalize against Q
X = sqrt(n-k) * X ./ sqcolnorms(X)'.^0.5; % Resphere
BX = B(X);                              % Collect matvecs
tr = trace(Q'*BQ) + trace(X'*BX) / k2;  % Hutch++ estimator

end
```

estimator rather than the Girard–Hutchinson estimator to estimate $\mathrm{tr}((\mathbf{I}-\boldsymbol{QQ}^*)\boldsymbol{B}(\mathbf{I}-\boldsymbol{QQ}^*))$. Code is provided in Program 13.5.

The benefits of resphering are demonstrated in Fig. 13.1, which compares the Girard–Hutchinson estimator to the Hutch++ algorithm with and without resphering. I use input matrix

$$\texttt{step} \coloneqq \boldsymbol{U}\,\mathbf{Diag}(\underbrace{1,\dots,1}_{50\text{ times}},\underbrace{10^{-3},\dots,10^{-3}}_{950\text{ times}})\boldsymbol{U}^* \in \mathbb{R}^{10^3\times10^3}. \tag{13.17}$$

Here, $\boldsymbol{U}$ is a Haar random orthogonal matrix. The matrix `step` has $k = 50$ large eigenvalues, after which the spectrum is flat. It takes the Hutch++ methods roughly $s \approx 3k = 150$ matvecs to compute a low-rank approximation capturing these dominant eigenvalues, explaining the substantial drop in the error of the Hutch++ methods at $s = 150$ matvecs. With resphering, the error drops two more orders of magnitude as s is increased beyond 150, owing to the relatively flat distribution of *nonzero* eigenvalues of the projected matrix $(\mathbf{I}-\boldsymbol{QQ}^*)\texttt{step}(\mathbf{I}-\boldsymbol{QQ})$. For $s = 300$, Hutch++ with resphering is 50× more accurate than Hutch++ without resphering and 1800× more accurate than the Girard–Hutchinson estimator.
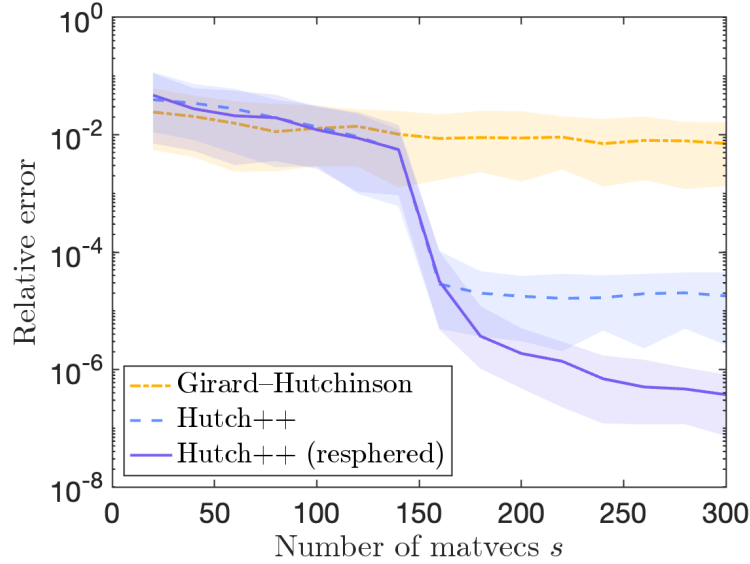
Figure 13.1: Comparison of Girard–Hutchinson and Hutch++ estimators with and without resphering on the `step` matrix (13.17). Lines show median of 100 trials, and error bars show 10% and 90% quantiles.

Weaknesses of Hutch++

The Hutch++ algorithm is an elegant approach for trace estimation, and it achieves excellent accuracy compared to the Girard–Hutchinson estimator when applied to matrices with rapidly decaying singular values. Even so, we recognize opportunities for improvement, which will be realized by the XTrace and XNysTrace estimators in the next chapter.

Apportionment. In its standard form, the Hutch++ algorithm uses a fixed division of $2s/3$ matvecs for low-rank approximation and $s/3$ matvecs for residual trace estimation. For a given matrix, this apportionment need not be optimal. For matrices with rapidly decaying singular values, the ideal strategy would be to put *all* matvecs into low-rank approximation; for matrices with slowly decreasing singular values, the opposite strategy is warranted. The apportionment problem for Hutch++ can be addressed with an adaptive scheme that apportions matvecs between low-rank approximation and residual trace estimation, as is done in [PCK22]. The XTrace and XNysTrace estimators introduced in next section solve the apportionment problem by redesigning the estimator to exploit a leave-one-out approach.

Exchangeability. The second weakness of Hutch++ is more conceptual. We have the following property of the optimal (i.e., minimum-variance) trace estimator.

> **Exchangeability principle.** Consider the class of estimators for $\operatorname{tr}(A)$ computed using matvecs $A\omega_1, \ldots, A\omega_k$ with *iid* random vectors $\omega_1, \ldots, \omega_k$. Among these methods, the minimum-variance unbiased estimator for $\operatorname{tr}(A)$ is a *permutation-invariant* function of $\omega_1, \ldots, \omega_k$.

This principle is due to Halmos [Hal46], and it is easy to confirm. Indeed, the variance of any estimator $\widehat{\operatorname{tr}}(\omega_1, \ldots, \omega_k)$ is reduced by averaging over all permutations of the test vectors $\omega_1, \ldots, \omega_k$:

$$
\begin{aligned}
\operatorname{Var}&\left(\frac{1}{k!} \sum_{\sigma \in \mathsf{S}_k} \widehat{\operatorname{tr}}(\omega_{\sigma(1)}, \ldots, \omega_{\sigma(k)}) \right) \\
&= \frac{1}{(k!)^2} \sum_{\sigma, \sigma' \in \mathsf{S}_k} \operatorname{Cov}\left(\widehat{\operatorname{tr}}(\omega_{\sigma(1)}, \ldots, \omega_{\sigma(k)}), \widehat{\operatorname{tr}}(\omega_{\sigma'(1)}, \ldots, \omega_{\sigma'(k)}) \right) \\
&\le \frac{1}{(k!)^2} \sum_{\sigma, \sigma' \in \mathsf{S}_k} \left[\operatorname{Var}\left(\widehat{\operatorname{tr}}(\omega_{\sigma(1)}, \ldots, \omega_{\sigma(k)}) \right) \operatorname{Var}\left(\widehat{\operatorname{tr}}(\omega_{\sigma'(1)}, \ldots, \omega_{\sigma'(k)}) \right) \right]^{1/2} \\
&= \operatorname{Var}(\widehat{\operatorname{tr}}(\omega_1, \ldots, \omega_k)).
\end{aligned}
$$

The first identity is sesquilinearity of the covariance, the inequality is Cauchy–Schwarz for the variance, and the second identity follows from the observation that $\widehat{\operatorname{tr}}(\omega_{\sigma(1)}, \ldots, \omega_{\sigma(k)})$ has the same distribution for every permutation σ.

The HUTCH++ estimator is not exchangeable: Half the test vectors are used for low-rank approximation, while the other half are used for residual trace estimation. As such, there must be an exchangeable estimator with lower variance than HUTCH++. One way we could fashion such an estimator is by averaging the value of the HUTCH++ estimator over all possible splits of test vectors between these two tasks, as in the derivation above. Unfortunately, computing this symmetrized HUTCH++ estimator is computationally infeasible because there are $\binom{2s/3}{s/3} \ge 2^{s/3}$ divisions of $2s/3$ test vectors into two equal groups. A new, different approach is needed to obtain an exchangeable, variance-reduced trace estimator. The XTRACE estimator will remedy this shortcoming of HUTCH++ by using a leave-one-out design, which is exchangeable by construction.

Chapter 14

THE LEAVE-ONE-OUT APPROACH AND APPLICATION TO TRACE ESTIMATION

> *An idea which can be used only once is a trick. If one can use it more than once it becomes a method.*
>
> George Pólya and Gabor Szegő, *Problems and Theorems in Analysis I*
> [PS98, p. VIII]

In the previous chapter, we developed our basic tools—randomized Monte Carlo and low-rank approximations for matrices—and used them to estimate matrix attributes. This discussion culminated with the Hutch++ algorithm, which showed how combining Monte Carlo estimation and low-rank approximation can lead to more accurate trace estimates than either approach individually.

In this chapter, we will present the leave-one-out approach for matrix attribute estimation, a flexibility methodology that combines Monte Carlo and low-rank approximation in a way that squeezes as much information possible out of every matvec. When the leave-one-out approach is applied to trace estimation, it results in the XTrace and XNysTrace algorithms. Each method is an exchangeable estimator that uses every matvec both for low-rank approximation and for Monte Carlo estimation. This strategy ameliorates the weaknesses with Hutch++ identified at the end of last chapter.

This chapter presents a tutorial-style introduction to leave-one out randomized matrix algorithm design, focusing on developing trace estimators for general and psd matrices. The basic idea is simple, but the formulas appearing in the final algorithms can be complicated. This chapter will attempt to demystify these expressions and to provide a clear example of how to derive a leave-one-out randomized matrix algorithm. Subsequent chapters will present several additional applications of the leave-one-out approach to matrix attribute estimation.

Sources. Both the leave-one-out approach and the XTrace and XNysTrace algorithms were developed in the paper:

Ethan N. Epperly, Joel A. Tropp, and Robert J. Webber. "XTrace: Making the Most

of Every Sample in Stochastic Trace Estimation". In: *SIAM Journal on Matrix Analysis and Applications* (Mar. 2024), pp. 1–23. DOI: 10.1137/23M1548323.

This chapter refines the original paper [ETW24] by developing new implementations of the XNYSTRACE algorithm with processing costs that are up to 7× faster than the implementations given in the original paper.

Outline. Section 14.1 introduces the leave-one-out approach to matrix attribute estimation and uses it to derive the XTRACE algorithm for trace estimation. To implement this algorithm efficiently, we require a downdating formula for the randomized SVD, which is developed in Section 14.2. We devise an efficient XTRACE implementation using this formula in Section 14.3. Section 14.4 presents XNYSTRACE, an improved version of XTRACE for psd matrices. Sections 14.5 and 14.6 discusses implementation of XNYSTRACE. Section 14.7 contains experimental comparison of XTRACE and XNYSTRACE to HUTCH++ and the Girard–Hutchinson estimators, Section 14.8 introduces resphered versions of XTRACE and XNYSTRACE, and Section 14.9 presents an application of trace estimators to computing the Estrada index of a network. We conclude in Section 14.10 by summarizing the leave-one-out approach to matrix attribute estimation.

14.1 XTRACE: The leave-one-out approach

The leave-one-out approach to matrix attribute estimation consists of five steps:

1. Compute a low-rank approximation to the input matrix by multiplying it against a collection of test vectors.

2. Decompose the quantity of interest into known piece depending on the low-rank approximation plus a residual.

3. Construct a Monte Carlo estimate of the residual using a single random vector.

4. Downdate the low-rank approximation by recomputing it with a test vector removed, and use the left-out test vector as the random vector in step 3.

5. Average the estimator from step 4 over all choices of vectors to leave out.

At present, these steps are fairly abstract. To make this program more complete, we shall derive XTRACE, a leave-one-out algorithm for estimating the trace of a general square matrix B.

Step 1: Compute a low-rank approximation to the input matrix by multiplying it against a collection of test vectors. The first step of the leave-one-out approach is to construct a low-rank approximation of the matrix $\boldsymbol{B}$. Here, since $\boldsymbol{B}$ is a general square matrix, we employ the randomized SVD (Section 2.3). Let $\boldsymbol{\Omega} \in \mathbb{K}^{n \times k}$ be a random matrix with isotropic columns, compute the product $\boldsymbol{B\Omega}$, and orthogonalize, obtaining $\boldsymbol{Q} = \mathbf{Orth}(\boldsymbol{B\Omega})$. The resulting randomized SVD low-rank approximation is

$$\widehat{\boldsymbol{B}} := \boldsymbol{QQ}^*\boldsymbol{B}.$$

Step 2: Decompose the quantity of interest into known piece depending on the low-rank approximation plus a residual. Next, we decompose the quantity of interest, $\mathrm{tr}(\boldsymbol{B})$ in this case, into a known piece depending on the low-rank approximation and an unknown residual term. The simplest decomposition exploits the linearity of the trace:

$$\mathrm{tr}(\boldsymbol{B}) = \mathrm{tr}(\widehat{\boldsymbol{B}}) + \mathrm{tr}(\boldsymbol{B} - \widehat{\boldsymbol{B}}) = \mathrm{tr}(\boldsymbol{Q}^*\boldsymbol{BQ}) + \mathrm{tr}((\mathbf{I} - \boldsymbol{QQ}^*)\boldsymbol{B}).$$

We saw a better decomposition from our derivation of the Hutch++ algorithm. Indeed, using the identity $(\mathbf{I} - \boldsymbol{QQ}^*)^2 = \mathbf{I} - \boldsymbol{QQ}^*$ and the cyclic property of the trace, the residual trace is

$$\mathrm{tr}((\mathbf{I} - \boldsymbol{QQ}^*)\boldsymbol{B}) = \mathrm{tr}((\mathbf{I} - \boldsymbol{QQ}^*)^2\boldsymbol{B}) = \mathrm{tr}((\mathbf{I} - \boldsymbol{QQ}^*)\boldsymbol{B}(\mathbf{I} - \boldsymbol{QQ}^*)).$$

Thus, in XTrace, we shall also use the following decomposition of the trace:

$$\mathrm{tr}(\boldsymbol{B}) = \mathrm{tr}(\widehat{\boldsymbol{B}}) + \mathrm{tr}(\boldsymbol{B} - \widehat{\boldsymbol{B}}) = \mathrm{tr}(\boldsymbol{Q}^*\boldsymbol{BQ}) + \mathrm{tr}((\mathbf{I} - \boldsymbol{QQ}^*)\boldsymbol{B}(\mathbf{I} - \boldsymbol{QQ}^*)).$$

Step 3: Construct a Monte Carlo estimate of the residual using a single random vector. Now, we construct a Monte Carlo approximation to the residual trace $\mathrm{tr}((\mathbf{I} - \boldsymbol{QQ}^*)\boldsymbol{B}(\mathbf{I} - \boldsymbol{QQ}^*))$ using a single random vector ω. Assuming ω is isotropic, the natural estimator is the single-vector Girard–Hutchinson estimator

$$\omega^*(\mathbf{I} - \boldsymbol{QQ}^*)\boldsymbol{B}(\mathbf{I} - \boldsymbol{QQ}^*)\omega \approx \mathrm{tr}((\mathbf{I} - \boldsymbol{QQ}^*)\boldsymbol{B}(\mathbf{I} - \boldsymbol{QQ}^*)).$$

One could also use resphering at this step, but we will postone this refinement to simplify the presentation. Using the Monte Carlo estimate for the residual, we obtain the following unbiased trace estimate:

$$\widehat{\mathrm{tr}} := \mathrm{tr}(\boldsymbol{Q}^*\boldsymbol{BQ}) + \omega^*(\mathbf{I} - \boldsymbol{QQ}^*)\boldsymbol{B}(\mathbf{I} - \boldsymbol{QQ}^*)\omega. \tag{14.1}$$

Step 4: Downdate the low-rank approximation by recomputing it with a test vector removed, and use the left-out test vector as the random vector in step 3. So far,

the estimate (14.1) is nothing special, basically just a lopsided version of Hutch++ where only a single vector ω is used to estimate the residual trace $\mathrm{tr}(\boldsymbol{B} - \widehat{\boldsymbol{B}})$. Now, we employ the core device of the leave-one-out approach. Pick an index $1 \le i \le k$, and leave out the ith column of the test matrix $\boldsymbol{\Omega}$, resulting in a downdated low-rank approximation

$$\widehat{\boldsymbol{B}}_{(i)} \coloneqq \boldsymbol{Q}_{(i)}\boldsymbol{Q}^*_{(i)}\boldsymbol{B} \quad \text{where } \boldsymbol{Q}_{(i)} \coloneqq \mathbf{Orth}(\boldsymbol{B}\boldsymbol{\Omega}_{-i}).$$

Here and elsewhere, $\boldsymbol{\Omega}_{-i}$ denotes $\boldsymbol{\Omega}$ without its ith column, and ω_i denotes the ith column of $\boldsymbol{\Omega}$. The downdated low-rank approximation $\widehat{\boldsymbol{B}}_{(i)}$ is somewhat less accurate because it has smaller rank $k - 1$, but we have freed up an isotropic vector ω_i that is *independent* of the low-rank approximation $\widehat{\boldsymbol{B}}_{(i)}$ and the orthonormal basis $\boldsymbol{Q}_{(i)}$. Introducing the downdated approximation $\widehat{\boldsymbol{B}}_{(i)}$ and the left-out vector vector ω_i in the trace estimate (14.1) from step 3, we obtain a new basic trace estimate:

$$\widehat{\mathrm{tr}}_i \coloneqq \mathrm{tr}(\boldsymbol{Q}^*_{(i)}\boldsymbol{B}\boldsymbol{Q}_{(i)}) + \omega_i^*(\mathbf{I} - \boldsymbol{Q}_{(i)}\boldsymbol{Q}^*_{(i)})\boldsymbol{B}(\mathbf{I} - \boldsymbol{Q}_{(i)}\boldsymbol{Q}^*_{(i)})\omega_i. \tag{14.2}$$

We call $\widehat{\mathrm{tr}}_i$ the *ith basic XTrace estimator*. It is an unbiased estimate for $\mathrm{tr}(\boldsymbol{B})$, owing to the independence of ω_i from $\boldsymbol{Q}_{(i)}$.

Step 5: Average the estimator from step 4 over all choices of vectors to leave out.
Each basic XTrace estimator $\widehat{\mathrm{tr}}_i$ can individually be a poor estimate of the trace, but we can reduce variance by averaging all of them:

$$\widehat{\mathrm{tr}}_X \coloneqq \frac{1}{k}\sum_{i=1}^{k}\widehat{\mathrm{tr}}_i. \tag{14.3}$$

We call $\widehat{\mathrm{tr}}_X$ the (full) *XTrace estimator*, short for the e**X**changeable **Trace** estimator.

Discussion. While it is not obvious yet, the XTrace estimator (14.3) can be computed using only $2k$ matvecs (k to compute $\boldsymbol{A}\boldsymbol{\Omega}$, k to compute $\boldsymbol{A}\boldsymbol{Q}$). Equivalently, a fixed budget of s matvecs can accommodate a rank of $k = \lfloor s/2 \rfloor$. (Henceforth, we will assume s is even for simplicity.) Thus, the XTrace estimator takes the form

$$\widehat{\mathrm{tr}}_X = \frac{1}{s/2}\sum_{i=1}^{s/2}\left[\mathrm{tr}(\boldsymbol{Q}^*_{(i)}\boldsymbol{B}\boldsymbol{Q}_{(i)}) + \omega_i^*(\mathbf{I} - \boldsymbol{Q}_{(i)}\boldsymbol{Q}^*_{(i)})\boldsymbol{B}(\mathbf{I} - \boldsymbol{Q}_{(i)}\boldsymbol{Q}^*_{(i)})\omega_i\right]. \tag{14.4}$$

Let us compare XTrace to Hutch++. With its budget of s matvecs, Hutch++ dedicates $2s/3$ matvecs to generate a low-rank approximation $\widehat{\boldsymbol{B}} = \boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{B}$ and $s/3$ matvecs to forming an estimate of the residual trace $\mathrm{tr}(\boldsymbol{B} - \widehat{\boldsymbol{B}})$. XTrace, in

effect, uses the same pool of $s/2$ matvecs both to generate a *family* of low-rank approximations $\widehat{\boldsymbol{B}}_{(i)}$ and to estimate all of the residual traces $\operatorname{tr}(\boldsymbol{B} - \widehat{\boldsymbol{B}}_{(i)})$. In this sense, *XTrace uses all of its multipications with isotropic random test vectors both for low-rank approximation and for Monte Carlo estimation.* XTrace also expends an additional $s/2$ matvecs to compute $\boldsymbol{BQ}$, which is necessary to evaluate the expressions $\operatorname{tr}(\boldsymbol{Q}_{(i)}^* \boldsymbol{BQ}_{(i)})$.

XTrace fixes both weaknesses of Hutch++ that we identified in Section 13.3. First, XTrace is exchangeable, being a symmetric function of all the matvecs $\boldsymbol{B\omega}_i$ collected in the first phase of the algorithm. Second, and more importantly, XTrace is based on a rank-$(s/2)$ approximation to $\boldsymbol{B}$, which is more powerful than Hutch++'s rank-$(s/3)$ approximation. Thus, XTrace can be significantly more accurate than Hutch++ for matrices $\boldsymbol{B}$ with rapid singular value decay, while maintaining accuracy for matrices with slow singular value decay. Moreover, XTrace achieves this increased accuracy automatically through an exchangeable, leave-one-out design; it does not require manually or adaptively apportioning matvecs between the low-rank approximation and Monte Carlo estimation roles, as in the adaptive Hutch++ algorithm [PCK22]. (See Section 15.5 for discussion.)

14.2 Leave-one-out formula for the randomized SVD

To implement XTrace, we need a way of efficiently evaluating the XTrace estimator (14.4). For the purpose of error estimation (Section 15.3), we will also need to form each of the individual estimators $\widehat{\operatorname{tr}}_i$ defined in (14.2). To develop efficient implementations, we can use the following *downdating formula* for the randomized SVD:

Theorem 14.1 (Downdating the randomized SVD)**.** *Let* $\boldsymbol{B} \in \mathbb{K}^{n \times n}$ *and* $\boldsymbol{\Omega} \in \mathbb{K}^{n \times k}$ *be matrices, and assume* $\boldsymbol{B\Omega}$ *is full-rank. Compute the matrix* $\boldsymbol{Q}$ *defining the randomized SVD approximation* $\widehat{\boldsymbol{B}} = \boldsymbol{QQ}^*\boldsymbol{B}$ *by an economy-size* QR *decomposition* $\boldsymbol{B\Omega} = \boldsymbol{QR}$. *The downdated matrices* $\boldsymbol{Q}_{(i)} = \mathbf{Orth}(\boldsymbol{B\Omega}_{-i})$ *admit the representation*

$$\boldsymbol{Q}_{(i)}\boldsymbol{Q}_{(i)}^* = \boldsymbol{Q}(\mathbf{I} - \boldsymbol{s}_i\boldsymbol{s}_i^*)\boldsymbol{Q}^* \quad for\ i = 1, 2, \ldots, k. \tag{14.5}$$

The vectors $\boldsymbol{s}_i$ *are equal to the columns of* $\boldsymbol{R}^{-*}$, *scaled to have unit norm.*

This result was developed by myself and collaborators in [ET24; ETW24].

I find this result to be quite surprising. To compute the randomized SVD, we must orthogonalize the matrix $\boldsymbol{B\Omega}$, which is conventionally accomplished using

a QR decomposition $\boldsymbol{B\Omega} = \boldsymbol{QR}$. Theorem 14.1 demonstrates that the matrix $\boldsymbol{R}$, a useless byproduct in most randomized SVD implementations, contains all the information needed to extract all k downdated approximations $\widehat{\boldsymbol{B}}_{(i)} = \boldsymbol{Q}_{(i)}\boldsymbol{Q}_{(i)}^*\boldsymbol{B}$. These downdated approximations can be represented implicitly using the formula (14.5), and the arithmetic cost is only $O(k^3)$ operations to compute the $\boldsymbol{s}_i$ vectors. In particular, the cost is independent of both dimensions of the original matrix $\boldsymbol{B}$!

Proof of Theorem 14.1. Fix an index i. To compute $\boldsymbol{Q}_{(i)}$, we to form a QR decomposition of $\boldsymbol{B\Omega}_{-i}$. The decomposition $\boldsymbol{B\Omega}_{-i} = \boldsymbol{QR}_{-i}$ is nearly a QR decomposition, but the matrix $\boldsymbol{R}_{-i}$ is no longer triangular when $i < k$ because its ith column has been deleted. To restore triangularity, we take a (full) QR decomposition of $\boldsymbol{R}_{-i}$, which we partition as

$$\boldsymbol{R}_{-i} = \begin{bmatrix} \widetilde{\boldsymbol{Q}} & \widetilde{\boldsymbol{q}} \end{bmatrix} \begin{bmatrix} \widetilde{\boldsymbol{R}} \\ \boldsymbol{0} \end{bmatrix} \quad \text{for } \widetilde{\boldsymbol{Q}} \in \mathbb{K}^{k\times(k-1)}, \widetilde{\boldsymbol{R}} \in \mathbb{K}^{(k-1)\times(k-1)}, \widetilde{\boldsymbol{q}} \in \mathbb{K}^k.$$

Using this factorization, we obtain a QR decomposition of $\boldsymbol{B\Omega}_{-i}$, namely

$$\boldsymbol{B\Omega}_{-i} = \boldsymbol{QR}_{-i} = (\boldsymbol{Q}\widetilde{\boldsymbol{Q}})\widetilde{\boldsymbol{R}}.$$

In particular, $\boldsymbol{Q}_{(i)} := \boldsymbol{Q}\widetilde{\boldsymbol{Q}}$ is an orthonormal basis for the column space of $\boldsymbol{B\Omega}_{-i}$. Therefore, the outer product of $\boldsymbol{Q}_{(i)}$ with itself is

$$\boldsymbol{Q}_{(i)}\boldsymbol{Q}_{(i)}^* = \boldsymbol{Q}(\widetilde{\boldsymbol{Q}}\widetilde{\boldsymbol{Q}}^*)\boldsymbol{Q}^*.$$

Since $\begin{bmatrix} \widetilde{\boldsymbol{Q}} & \widetilde{\boldsymbol{q}} \end{bmatrix}$ is unitary, the parenthesized term may be written as

$$\widetilde{\boldsymbol{Q}}\widetilde{\boldsymbol{Q}}^* = \begin{bmatrix} \widetilde{\boldsymbol{Q}} & \widetilde{\boldsymbol{q}} \end{bmatrix} \begin{bmatrix} \widetilde{\boldsymbol{Q}} & \widetilde{\boldsymbol{q}} \end{bmatrix}^* - \widetilde{\boldsymbol{q}}\widetilde{\boldsymbol{q}}^* = \mathbf{I} - \widetilde{\boldsymbol{q}}\widetilde{\boldsymbol{q}}^*.$$

Combining the two previous displays yields

$$\boldsymbol{Q}_{(i)}\boldsymbol{Q}_{(i)}^* = \boldsymbol{Q}(\mathbf{I} - \widetilde{\boldsymbol{q}}\widetilde{\boldsymbol{q}}^*)\boldsymbol{Q}^*.$$

To establish the desired result, it remains to show that the unit vector $\widetilde{\boldsymbol{q}}$ is proportional to the ith column of $\boldsymbol{R}^{-*}$. By construction, $\widetilde{\boldsymbol{q}}$ is orthogonal to the column space of $\boldsymbol{R}_{-i}$. That is, $\boldsymbol{R}_{-i}^*\widetilde{\boldsymbol{q}} = \boldsymbol{0}$. Restoring the deleted ith column to $\boldsymbol{R}$ and recalling that $\boldsymbol{R}$ is nonsingular, we conclude that $\boldsymbol{R}^*\widetilde{\boldsymbol{q}} = \alpha \cdot \mathbf{e}_i$ for some nonzero scalar α. Therefore, $\widetilde{\boldsymbol{q}} = \alpha \cdot \boldsymbol{R}^{-*}\mathbf{e}_i$ is proportional to the ith column of $\boldsymbol{R}^{-*}$. $\qquad\square$

14.3 Implementing XTRACE efficiently

Having established Theorem 14.1, we can use it to calculate the basic XTRACE estimates $\widehat{\mathrm{tr}}_i$, defined in (14.2). This section begins with a derivation for an efficiently computable formula for the $\widehat{\mathrm{tr}}_i$'s, after which implementation is discussed.

Formula for the basic XTRACE estimates: Derivation

The easiest way to derive efficient formulas for a leave-one-out algorithm, in my experience, is to proceed methodically and introduce variables to represent intermediate matrices that arise during the derivation. Let me demonstrate as we derive an efficient XTRACE implementation.

We begin with the randomized SVD step. First, generate a random matrix $\boldsymbol{\Omega} \in \mathbb{K}^{n \times (s/2)}$ with isotropic random columns. Next, form the product

$$Y := B\boldsymbol{\Omega} \tag{14.6}$$

and obtain its (economy-size) QR decomposition

$$Y = QR. \tag{14.7}$$

Then, to use the randomized SVD downdating formula (14.5), generate the matrix S (with columns s_i) by building R^{-*} and recaling each of its columns to have norm one. Finally, compute

$$Z := BQ. \tag{14.8}$$

The matrices Y, Q, S, and Q will be used later in our derivation.

Next, we turn to the basic XTRACE estimators $\widehat{\mathrm{tr}}_i$, which were defined in (14.2). To use the randomized SVD downdating formula (14.5), invoke the cyclic property of the trace and write:

$$\widehat{\mathrm{tr}}_i = \mathrm{tr}(BQ_{(i)}Q_{(i)}^*) + \omega_i^*(\mathbf{I} - Q_{(i)}Q_{(i)}^*)B(\mathbf{I} - Q_{(i)}Q_{(i)}^*)\omega_i.$$

Now, invoke the downdating formula (14.5):

$$\widehat{\mathrm{tr}}_i = \underbrace{\mathrm{tr}(BQ(\mathbf{I} - s_is_i^*)Q^*)}_{\text{Ⓐ}} + \underbrace{\omega_i^*(\mathbf{I} - Q(\mathbf{I} - s_is_i^*)Q^*)B(\mathbf{I} - Q(\mathbf{I} - s_is_i^*)Q^*)\omega_i}_{\text{Ⓑ}}.$$

$$\tag{14.9}$$

The result is a sum of two terms, Ⓐ and Ⓑ, which we will treat separately.

Begin with Ⓐ. Use the cyclic property of the trace and (14.8) to rewrite

$$\text{Ⓐ} = \mathrm{tr}(Q^*BQ(\mathbf{I} - s_is_i^*)) = \mathrm{tr}(Q^*Z(\mathbf{I} - s_is_i^*)).$$

To evaluate this expression, compute

$$\boldsymbol{H} := \boldsymbol{Q}^* \boldsymbol{Z}. \tag{14.10}$$

The expression Ⓐ now simplifies as

$$Ⓐ = \operatorname{tr}(\boldsymbol{H}) - \boldsymbol{s}_i^* \boldsymbol{H} \boldsymbol{s}_i. \tag{14.11}$$

This formula constitutes our final expression for Ⓐ.

Now, we treat Ⓑ. Begin by forming the matrix

$$\boldsymbol{W} := \boldsymbol{Q}^* \boldsymbol{\Omega}, \tag{14.12}$$

from which we may define a matrix $\boldsymbol{X} \in \mathbb{K}^{(s/2) \times (s/2)}$ with columns

$$\boldsymbol{x}_i := (\mathbf{I} - \boldsymbol{s}_i \boldsymbol{s}_i^*) \boldsymbol{Q}^* \boldsymbol{\omega}_i = \boldsymbol{w}_i - \boldsymbol{s}_i \cdot \boldsymbol{s}_i^* \boldsymbol{w}_i. \tag{14.13}$$

With this definition, Ⓑ can be written as

$$
\begin{aligned}
Ⓑ &= \boldsymbol{\omega}_i^* (\mathbf{I} - \boldsymbol{Q} \boldsymbol{Q}^* + \boldsymbol{Q} \boldsymbol{s}_i \boldsymbol{s}_i^* \boldsymbol{Q}^*) \boldsymbol{B} (\boldsymbol{\omega}_i - \boldsymbol{Q} \boldsymbol{x}_i) \\
&= (\boldsymbol{\omega}_i^* (\mathbf{I} - \boldsymbol{Q} \boldsymbol{Q}^*) + \boldsymbol{w}_i^* \boldsymbol{s}_i \cdot \boldsymbol{s}_i^* \boldsymbol{Q}^*)(\boldsymbol{y}_i - \boldsymbol{Z} \boldsymbol{x}_i).
\end{aligned}
$$

Since $\boldsymbol{Y} = \boldsymbol{B} \boldsymbol{\Omega} = \boldsymbol{Q} \boldsymbol{R}$ is a QR decomposition,

$$(\mathbf{I} - \boldsymbol{Q} \boldsymbol{Q}^*) \boldsymbol{y}_i = \mathbf{0} \quad \text{and} \quad \boldsymbol{Q}^* \boldsymbol{y}_i = \boldsymbol{r}_i.$$

Therefore,

$$Ⓑ = -\boldsymbol{\omega}_i^* (\mathbf{I} - \boldsymbol{Q} \boldsymbol{Q}^*) \boldsymbol{Z} \boldsymbol{x}_i + \boldsymbol{w}_i^* \boldsymbol{s}_i \cdot \boldsymbol{s}_i^* (\boldsymbol{r}_i - \boldsymbol{H} \boldsymbol{x}_i).$$

To simplify further, introduce and form the matrix

$$\boldsymbol{T} := \boldsymbol{Z}^* \boldsymbol{\Omega}. \tag{14.14}$$

Now, we may simplify Ⓑ as

$$Ⓑ = -\boldsymbol{t}_i^* \boldsymbol{x}_i + \boldsymbol{w}_i^* \boldsymbol{H} \boldsymbol{x}_i + \boldsymbol{w}_i^* \boldsymbol{s}_i \cdot \boldsymbol{s}_i^* (\boldsymbol{r}_i - \boldsymbol{H} \boldsymbol{x}_i).$$

Finally, using the definition (14.13) of the vectors $\boldsymbol{x}_i$, we simplify

$$Ⓑ = -\boldsymbol{t}_i^* \boldsymbol{x}_i + \boldsymbol{x}_i^* \boldsymbol{H} \boldsymbol{x}_i + \boldsymbol{w}_i^* \boldsymbol{s}_i \cdot \boldsymbol{s}_i^* \boldsymbol{r}_i.$$

Combining this expression for Ⓑ with the expression (14.11) for Ⓐ, we obtain our final expression for $\widehat{\operatorname{tr}}_i$:

$$\widehat{\operatorname{tr}}_i = \operatorname{tr}(\boldsymbol{H}) - \boldsymbol{s}_i^* \boldsymbol{H} \boldsymbol{s}_i - \boldsymbol{t}_i^* \boldsymbol{x}_i + \boldsymbol{x}_i^* \boldsymbol{H} \boldsymbol{x}_i + \boldsymbol{w}_i^* \boldsymbol{s}_i \cdot \boldsymbol{s}_i^* \boldsymbol{r}_i. \tag{14.15}$$

Program 14.1 `diagprod.m`. Efficient implementation of the **diagprod** operation (14.16).

```
function d = diagprod(F,G)
% Input:  Matrices F and G of the same size
% Output: The diagonal of the product d = diag(F'*G)

d = sum(conj(F).*G,1).';

end
```

Formula for the basic XTRACE estimates: Implementation

We now discuss a few implementation details for evaluating the formula (14.15) for the basic trace estimates $\widehat{\text{tr}}_i$.

Diagonal entries of products. The formula (14.15) contains many expressions such as $f_i^* g_i$, which constitute the diagonal entries of the matrix product $F^* G$ for $F, G \in \mathbb{K}^{d_1 \times d_2}$. We denote the vector of all $f_i^* g_i$ using the **diagprod** operation:

$$\textbf{diagprod}(F, G) := \textbf{diag}(F^* G) = (f_i^* g_i : 1 \le i \le d_2). \quad (14.16)$$

One way of evaluating the **diagprod** operation would be to form the matrix product $F^* G$ and extract its diagonal; this approach expends $O(d_1^2 d_2)$ operations. However, computing these expressions directly is cheaper, requiring just just $O(d_1 d_2)$ operations to evaluate $f_i^* g_i$ for each i. Code is given in Program 14.1

The vectors x_i. The vectors x_i defined in (14.13) can be packaged into a matrix X with formula

$$X = W - S \cdot \textbf{Diag}(\textbf{diagprod}(S, W)). \quad (14.17)$$

Recall that $\textbf{Diag}(a)$ denotes the diagonal matrix with diagonal entries a_i. In MATLAB, expression $S \cdot \textbf{Diag}(a)$ can be evaluated rapidly as `S .* a.'`.

XTRACE implementation. Using all of the formulas we've developed, the vector $\widehat{\textbf{tr}} = (\widehat{\text{tr}}_i : 1 \le i \le s/2)$ of basic XTRACE estimators (14.15) is

$$\widehat{\textbf{tr}} = \text{tr}(H)\textbf{1} - \textbf{diagprod}(S, HS) - \textbf{diagprod}(T, X) + \textbf{diagprod}(X, HX)$$
$$+ \textbf{diagprod}(W, S) \odot \textbf{diagprod}(S, R). \quad (14.18)$$

The XTRACE estimator (14.4) is merely the mean of the entries of this vector; the variance of the entries will be used for error estimation in Section 15.3. To evaluate $\widehat{\textbf{tr}}$, we evaluate the equations (14.6) to (14.8), (14.10), (14.12), (14.14), (14.17)

Program 14.2 `xtrace.m`. Efficient implementation of the XTRACE estimator. Subroutines `diagprod`, `random_signs`, and `cnormc` appear in Programs 14.1, F.1 and F.2.

```
function [tr, est] = xtrace(B,n,s)
% Input:  Function B() computing matrix products B(X) = B*X, number
%         of rows n, and number of matvecs s
% Output: Estimate tr of trace(B), estimate est of the error
%         abs(tr - trace(B))

% Define test matrix
k = floor(s/2);         % Approximation rank is s/2
Om = random_signs(n,k); % Matrix of random signs

% Randomized SVD and downdate
Y = B(Om);              % Collect matvecs
[Q,R] = qr(Y,"econ");   % Randomized SVD
S = cnormc(inv(R'));    % Downdate RSVD, cnormc normalizes columns

% Compute other necessary matrices
Z = B(Q);               % Collect matvecs
H = Q'*Z;
W = Q'*Om;
T = Z'*Om;
X = W - S .* diagprod(W,S).';

% Compute estimator, output
tr_vec = trace(H) * ones(k,1) - diagprod(S,H*S) - diagprod(T,X)...
            + diagprod(X,H*X) + diagprod(W, S) .* diagprod(S, R);
tr = mean(tr_vec);           % Trace estimate
est = std(tr_vec) / sqrt(k); % Error estimate

end
```

and (14.18). Stringing these formulas together produces a daunting and mysterious looking program, but this program is nothing complicated—just matrix algebra.

An implementation of XTRACE is provided in Program 14.2. This implementation outputs the trace estimator $\widehat{\mathrm{tr}}_X$ as first output `tr`; it also outputs an error estimate `est`, which will be discussed in Section 15.3.

14.4 XNYSTRACE: Trace estimation for psd matrices

XTRACE is an effective trace estimator for general square matrices, but it can be made even more efficient for psd matrices. This section will derive XNYSTRACE, an optimized trace estimator for psd matrices using the leave-one-out approach. We

uncover the algorithm by following the five-step process for deriving a leave-one-out randomized matrix algorithm, similar to our derivation of XTRACE in Section 14.1. Sections 14.5 and 14.6 will discuss efficient implementation. Throughout this section, and the following sections, $A \in \mathbb{K}^{n \times n}$ denotes a psd matrix.

Step 1: Compute a low-rank approximation to the input matrix by multiplying it against a collection of test vectors. Since the matrix A is psd, we have a more rich set of low-rank approximation algorithms available to us. Here, we will employ the single pass Nyström approximation

$$\widehat{A} := A\langle \Omega \rangle$$

associated with a test matrix $\Omega \in \mathbb{K}^{n \times s}$ with independent, isotropic random columns (Section 2.5). An advantage of the single-pass Nyström approximation is that we can form a rank-s approximation using only s matvecs, whereas s matvecs only allow us to obtan a rank-$(s/2)$ approximation with the randomized SVD.

Step 2: Decompose the quantity of interest into known piece depending on the low-rank approximation plus a residual. We decompose the trace by employing the simplest possible approach

$$\mathrm{tr}(A) = \mathrm{tr}(\widehat{A}) + \mathrm{tr}(A - \widehat{A}).$$

Step 3: Construct a Monte Carlo estimate of the residual using a single random vector. To estimate the residual trace $\mathrm{tr}(A - \widehat{A})$ using a single random vector, we employ the single-vector Girard–Hutchinson estimator

$$\omega^*(A - \widehat{A})\omega \approx \mathrm{tr}(A - \widehat{A}),$$

which leads to the trace estimate

$$\widehat{\mathrm{tr}} := \mathrm{tr}(\widehat{A}) + \omega^*(A - \widehat{A})\omega. \tag{14.19}$$

Step 4: Downdate the low-rank approximation by recomputing it with a test vector removed, and use the left-out test vector as the random vector in step 3. Now, we invoke the leave-one-out trick. Define the downdated the Nyström approximation by leaving out column i of the test matrix Ω,

$$\widehat{A}_{(i)} := A\langle \Omega_{-i} \rangle.$$

Using the left-out vector ω_i as the test vector ω in (14.19) gives the family of basic XNYSTRACE estimators

$$\widehat{\mathrm{tr}}_i := \mathrm{tr}(\widehat{A}_{(i)}) + \omega_i^*(A - \widehat{A}_{(i)})\omega_i \quad \text{for } i = 1, \ldots, s. \tag{14.20}$$

Step 5: Average the estimator from step 4 over all choices of vectors to leave out.
Averaging the basic XNYSTRACE estimators $\widehat{\mathrm{tr}}_i$ gives the full XNYSTRACE estimator

$$\widehat{\mathrm{tr}}_{\mathrm{XN}} := \frac{1}{s}\sum_{i=1}^{s}\widehat{\mathrm{tr}}_i = \frac{1}{s}\sum_{i=1}^{s}\left[\mathrm{tr}(\widehat{A}_{(i)}) + \omega_i^*(A - \widehat{A}_{(i)})\omega_i\right]. \tag{14.21}$$

Discussion. As we will show in Sections 14.5 and 14.6, the estimator XNYSTRACE estimator (14.21) can be computed using only the s matvecs composing $A\Omega$. XNYS-TRACE is exchangeable and dedicates all matvecs both to low-rank approximation and to Monte Carlo estimation of the residual trace.

The use of Nyström approximation for variance reduction in trace estimation was originally proposed by Persson, Cortinovis, and Kressner [PCK22], who developed a HUTCH++-style algorithm using Nyström approximation called NYSTRÖM++. The XNYSTRACE algorithm improves on NYSTRÖM++ by using an exchangeable, variance-reduced design.

The main difference between HUTCH++, XTRACE, and XNYSTRACE is the rank of the matrix approximation. Given a fixed budget of s matvecs, HUTCH++ uses a rank-$(s/3)$ approximation, XTRACE uses a rank-$(s/2)$ approximation, and XNYSTRACE uses a rank-s approximation. This disparity results in significant differences among these estimators *when applied to a matrix with rapidly decaying eigenvalues*. As an example, consider a psd matrix A whose eigenvalues decay at an exponential rate $\lambda_i(A) \le \alpha^i$ for $\alpha \in (0, 1)$. As we will show in Section 15.1, HUTCH++, XTRACE, and XNYSTRACE satisfy error bounds of the form

$$\begin{aligned}
\left(\mathbb{E}\left|\widehat{\mathrm{tr}}_{\mathrm{H++}} - \mathrm{tr}(A)\right|^2\right)^{1/2} &\le & C_1(\alpha)\,\alpha^{s/3}; \\
\left(\mathbb{E}\left|\widehat{\mathrm{tr}}_{\mathrm{X}} - \mathrm{tr}(A)\right|^2\right)^{1/2} &\le & \sqrt{s}\,C_2(\alpha)\,\alpha^{s/2}; \\
\left(\mathbb{E}\left|\widehat{\mathrm{tr}}_{\mathrm{XN}} - \mathrm{tr}(A)\right|^2\right)^{1/2} &\le & s\,C_3(\alpha)\,\alpha^{s}.
\end{aligned} \tag{14.22}$$

Here, $C_i(\alpha)$ denote prefactors depending only on α. For this matrix, the rate of convergence of XTRACE is $3/2\times$ faster than HUTCH++, and the convergence rate of XNYSTRACE is $3\times$ faster than HUTCH++.

14.5 Leave-one-out formula for randomized Nyström approximation

Just as the randomized SVD downdating formula (Theorem 14.1) was the main ingredient in making an efficient XTrace implementation, a fast Nyström approximation downdating formula will support a fast XNysTrace implementation. This formula will also be useful in deriving other leave-one-out algorithms based on randomized Nyström approximation.

Theorem 14.2 (Downdating randomized Nyström approximation). *Let $A \in \mathbb{K}^{n \times n}$ be psd and let $\mathbf{\Omega} \in \mathbb{K}^{n \times k}$ be a matrix, and assume $A\mathbf{\Omega}$ is full-rank. Denote $H := \mathbf{\Omega}^*(A\mathbf{\Omega})$. Then, the downdated Nyström approximation $\widehat{A}_{(i)} := A\langle \mathbf{\Omega}_{-i} \rangle$ has the representation*

$$\widehat{A}_{(i)} = (A\mathbf{\Omega}) \left(H^{-1} - \frac{H^{-1}(:,i)H^{-1}(i,:)}{H^{-1}(i,i)} \right) (A\mathbf{\Omega})^* \tag{14.23a}$$

$$= \widehat{A} - \frac{(A\mathbf{\Omega}H^{-1}(:,i))(A\mathbf{\Omega}H^{-1}(:,i))^*}{H^{-1}(i,i)}. \tag{14.23b}$$

The proof of this result relies on a consequence of the Banachiewicz inversion formula. Here, we restate the version of this formula given in [PS05, eq. (0.7.2)].

Fact 14.3 (Banachiewicz inversion formula). *Let $P \in \mathbb{K}^{d_1 \times d_1}, Q \in \mathbb{K}^{d_1 \times d_2}, R \in \mathbb{K}^{d_2 \times d_1}, S \in \mathbb{K}^{d_2 \times d_2}$ with P nonsingular. Then the block matrix*

$$M := \begin{bmatrix} P & Q \\ R & S \end{bmatrix}$$

is invertible if and only if the Schur complement $M/P := S - RP^{-1}Q$ *is invertible, in which case*

$$M^{-1} = \begin{bmatrix} P & Q \\ R & S \end{bmatrix}^{-1} = \begin{bmatrix} P^{-1} + P^{-1}Q(M/P)^{-1}RP^{-1} & -P^{-1}Q(M/P)^{-1} \\ -(M/P)^{-1}RP^{-1} & (M/P)^{-1} \end{bmatrix} \tag{14.24a}$$

$$= \begin{bmatrix} P^{-1} & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} -P^{-1}Q \\ I \end{bmatrix} (M/P)^{-1} \begin{bmatrix} -RP^{-1} & I \end{bmatrix}. \tag{14.24b}$$

As a consequence, we obtain a formula relating the inverse of a matrix to the inverse of a submatrix.

Corollary 14.4 (Downdating the inverse). *Instate the notation and assumptions of Fact 14.3, and denote $\mathsf{E} := \{d_1 + 1, \dots, d_1 + d_2\}$. Then*

$$\begin{bmatrix} P^{-1} & 0 \\ 0 & 0 \end{bmatrix} = M^{-1} - M^{-1}(:,\mathsf{E})[M^{-1}(\mathsf{E},\mathsf{E})]^{-1}M^{-1}(\mathsf{E},:).$$

Proof. Rewrite (14.24b) as

$$M^{-1} = \begin{bmatrix} P^{-1} & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} -P^{-1}Q(M/P)^{-1} \\ (M/P)^{-1} \end{bmatrix} (M/P) \begin{bmatrix} -(M/P)^{-1}RP^{-1} & (M/P)^{-1} \end{bmatrix},$$

and use (14.24a) to recognize the factors of the second term as submatrices of M^{-1}:

$$\begin{bmatrix} -P^{-1}Q(M/P)^{-1} \\ (M/P)^{-1} \end{bmatrix} = M^{-1}(:, \mathsf{E}), \quad \begin{bmatrix} -(M/P)^{-1}RP^{-1} & (M/P)^{-1} \end{bmatrix} = M^{-1}(\mathsf{E}, :).$$

Finally observe that $M/P = [M^{-1}(\mathsf{E}, \mathsf{E})]^{-1}$ is the inverse of the block-$(2, 2)$ entry of M^{-1}. Combining these observations yields the desired result. □

With this formula and its corollary in hand, the proof of Theorem 14.2 is immediate.

Proof of Theorem 14.2. The Nyström approximation $A\langle\Omega\rangle$ is invariant to permutation of the columns of Ω (Proposition 2.9(b)). As such, we can assume without loss of generality that $i = k$ by permuting the ith column of Ω to appear last.

The downdated Nyström approximation $\widehat{A}_{(k)}$ takes the form

$$\widehat{A}_{(k)} = A\Omega_{-k}(H_{(k)})^{-1}\Omega^*_{-k}A = A\Omega \begin{bmatrix} H^{-1}_{(k)} & 0 \\ 0^* & 0 \end{bmatrix} \Omega^*A.$$

Invoking Corollary 14.4 and repackaging $\Omega^*A = (A\Omega)^*$ yields the stated result. □

Using the formula (14.23)

The Nyström downdating formula (14.23) can be combined with the stable Nyström implementation from (2.11). We treat the shift μ as zero for the following discussion. In practice, the shift μ given by (2.11a) should be used.

We compute the Nyström approximation in outer product form $\widehat{A} = FF^*$ using the stable implementation in (2.11). First, we compute the matrix product $Y = A\Omega$. Then, we obtain a Cholesky decomposition $\Omega^*A\Omega = R^*R$. Finally, we construct the factor matrix $F = YR^{-1}$. Using these matrices, the downdating formula (14.23) may be written

$$\widehat{A}_{(i)} = FF^* - z_i z_i^* \quad \text{for } i = 1, 2, \dots, k \tag{14.25a}$$

where

$$Z := FR^{-*} \cdot \mathbf{Diag}\left(\mathbf{srn}(R^{-1})\right)^{-1/2}. \tag{14.25b}$$

The matrix Z contains all of the information needed to do Nyström downdating.

14.6 Implementing XNysTrace efficiently

We can use Theorem 14.2 to form the XNysTrace estimator rapidly. This section begins with a derivation of an efficient formula for the basic XNysTrace estimates (14.20). Then, we will discuss how to implement this formula.

Formula for the basic XNysTrace estimates: Derivation

Begin by generating a matrix $\boldsymbol{\Omega}$ with isotropic columns, compute the Nyström $\widehat{\boldsymbol{A}} = \boldsymbol{FF}^*$ via (2.11), and form $\boldsymbol{Z}$ from (14.25b). Then, substitute the Nyström downdating formula (14.23a) in the definition (14.20) of the basic XNysTrace estimators:

$$\widehat{\mathrm{tr}}_i = \mathrm{tr}(\boldsymbol{FF}^* - \boldsymbol{z}_i\boldsymbol{z}_i^*) + \boldsymbol{\omega}_i^*(\boldsymbol{A} - \widehat{\boldsymbol{A}} + \boldsymbol{z}_i\boldsymbol{z}_i^*)\boldsymbol{\omega}_i. \tag{14.26}$$

By Proposition 2.9(d), the Nyström approximation $\widehat{\boldsymbol{A}}$ satisfies the *interpolation condition* $\widehat{\boldsymbol{A}}\boldsymbol{\omega}_i = \boldsymbol{A}\boldsymbol{\omega}_i$. Using this interpolation property, the cyclic property of the trace, and the identity $\|\boldsymbol{F}\|_{\mathrm{F}}^2 = \mathrm{tr}(\boldsymbol{F}^*\boldsymbol{F})$, the formula (14.26) simplifies as

$$\widehat{\mathrm{tr}}_i = \|\boldsymbol{F}\|_{\mathrm{F}}^2 - \|\boldsymbol{z}_i\|^2 + |\boldsymbol{z}_i^*\boldsymbol{\omega}_i|^2.$$

More concisely, the vector of trace estimates $\mathbf{tr} = (\widehat{\mathrm{tr}}_i : 1 \leq i \leq s)$ is

$$\widehat{\mathbf{tr}} = \|\boldsymbol{F}\|_{\mathrm{F}}^2 \cdot \mathbf{1} - \mathbf{srn}(\boldsymbol{Z}) + |\,\mathbf{diagprod}(\boldsymbol{Z}, \boldsymbol{\Omega})\,|^2.$$

Formula for the basic XNysTrace estimates: Implementation

In practice, we use the shift μ given by formula (2.11a) to ensure numerical stability and success of the Cholesky decomposition (2.11d). As a result, the XNysTrace estimator produces an unbiased estimate of the *shifted matrix* $\boldsymbol{A} + \mu\mathbf{I}$. To correct for the shift, we remove the trace of the correction $\mathrm{tr}(\mu\mathbf{I}) = n\mu$ from each trace estimate, resulting in the alternative formula

$$\widehat{\mathbf{tr}} = \|\boldsymbol{F}\|_{\mathrm{F}}^2 \cdot \mathbf{1} - \mathbf{srn}(\boldsymbol{Z}) + |\,\mathbf{diagprod}(\boldsymbol{Z}, \boldsymbol{\Omega})\,|^2 - n\mu \cdot \mathbf{1}.$$

We use this formula in our code. Code for XNysTrace is provided in Program 14.3, which outputs the XNysTrace estimator $\widehat{\mathrm{tr}}_{\mathrm{XN}}$ as `tr` and an error estimate `est`; see Section 15.3 for discussion.

Remark 14.5 (Comparison to XNysTrace implementation in [ETW24]). Our original paper on XNysTrace [ETW24] uses a distnctive implementation based on a QR decomposition of $\boldsymbol{Y}$. The implementation in Program 14.3 is significantly faster. When applied to a problem of dimension $n = 10^5$ with $s = 10^3$ matvecs, the processing time (that is, the total runtime minus the time required to perform matvecs) for Program 14.3 was 7× faster than the implementation give in [ETW24]. ◇

Program 14.3 `xnystrace.m`. Efficient and stable implementation of XNYSTRACE estimator. Subroutines `nystrom` and `diagprod` appear in Programs 2.3 and 14.1.

```matlab
function [tr,est] = xnystrace(A,n,s)
% Input:  Function A() computing matrix products A(X) = A*X, number
%         of rows n, and number of matvecs s
% Output: Estimate tr of trace(A), estimate est of the error
%         abs(tr - trace(A))

[F,mu,Om,R] = nystrom(A,n,s); % Nystrom approx in outer product form
Z = (F/R') .* (sqrownorms(inv(R)) .^ (-1/2))'; % Downdate it

% Compute vector of estimates
tr_vec = norm(F,"fro")^2 * ones(s,1) - sqcolnorms(Z) ...
    + abs(diagprod(Z,Om)) .^ 2 - mu * n * ones(s,1);
tr = mean(tr_vec);          % Trace estimate
est = std(tr_vec) / sqrt(s); % Error estimate

end
```

14.7 Synthetic Experiments

To compare the XTRACE and XNYSTRACE algorithms to HUTCH++ and the Girard–Hutchinson estimator, we evaluate on four test matrices with different spectra:

$$\texttt{flat} \coloneqq U_1 \, \mathbf{Diag}(\texttt{linspace}(1,3,n))U_1^*, \tag{14.27a}$$

$$\texttt{poly} \coloneqq U_2 \, \mathbf{Diag}(i^{-2} : i = 1, \ldots, n)U_2^*, \tag{14.27b}$$

$$\texttt{exp} \coloneqq U_3 \, \mathbf{Diag}(0.7^i : i = 0, \ldots, n-1)U_3^*, \tag{14.27c}$$

$$\texttt{step} \coloneqq U_4 \, \mathbf{Diag}(\underbrace{1, \ldots, 1}_{50 \text{ times}}, \underbrace{10^{-3}, \ldots, 10^{-3}}_{n-50 \text{ times}})U_4^*. \tag{14.27d}$$

Here, $U_i \in \mathbb{R}^{n \times n}$ denote Haar-random orthogonal matrices, $\texttt{linspace}(1,3,n)$ denotes equally spaced entries between 1 and 3, and $n \coloneqq 10^3$. Each matrix is real and positive definite, although the `exp` matrix is singular up to numerical precision.

Results are shown in Fig. 14.1. Here are my conclusions:

With spectral decay, XNYSTRACE > XTRACE > HUTCH++ ≫ Girard–Hutchinson. For matrices with eigenvalues decaying at a steady, and sufficiently rapid, rate (e.g., the `poly` and `exp` examples), the ranking of methods is clear: XNYSTRACE, XTRACE, HUTCH++, then Girard–Hutchinson. This disparity is most visible with the `exp` examples, where XNYSTRACE converges at a 3× faster exponential rate than HUTCH++ and XTRACE converges 1.5× faster. The Girard–Hutchinson estimator, by contrast, converges at the much slower Monte Carlo rate $\Theta(s^{-1/2})$.
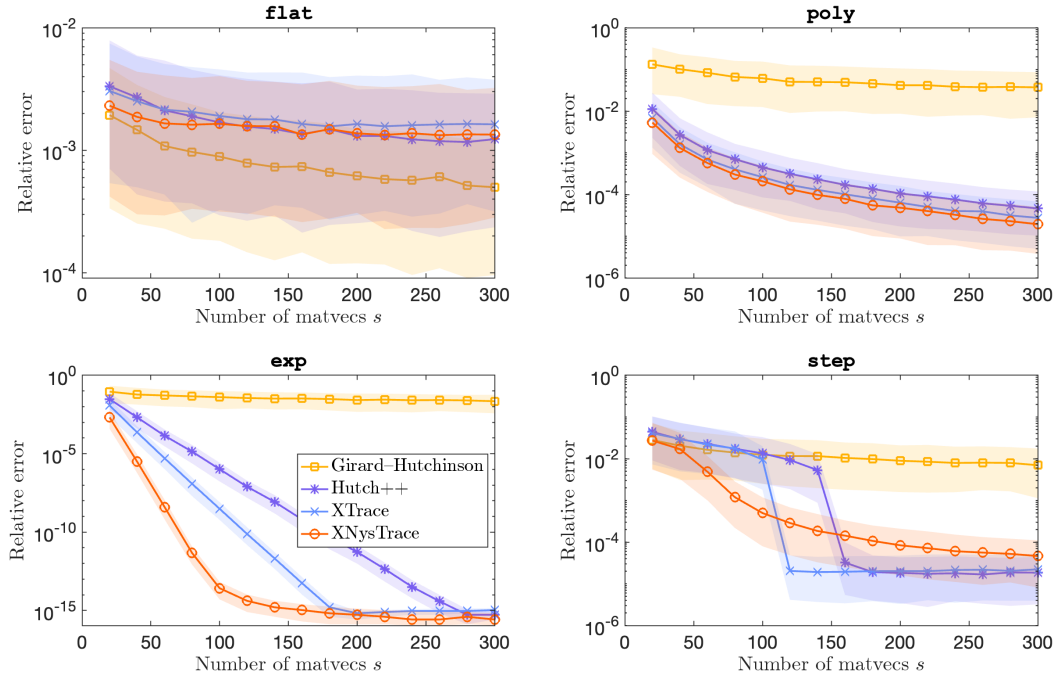
Figure 14.1: Relative error of trace estimates by Girard–Hutchinson estimator (yellow squares), Hᴜᴛᴄʜ++ (purple asterisks), XTʀᴀᴄᴇ (blue crosses), and XNʏsTʀᴀᴄᴇ (orange circles) for the four test matrices (14.27) for different numbers of matvecs s. Lines show median of 1000 trials and error bars show 10% and 90% quantiles.

Without spectral decay, variance reduction by low-rank approximation doesn't help. The Hᴜᴛᴄʜ++, XTʀᴀᴄᴇ, and XNʏsTʀᴀᴄᴇ estimators depend on reducing the variance of the trace estimate by using a low-rank approximation of the matrix as a control variate. This technique does not provide any benefit, and in fact slightly harms the quality of the estimate, when the matrix has a flat spectrum. This effect is visible with the `flat` matrix, where the Girard–Hutchinson estimator beats the Hᴜᴛᴄʜ++, XTʀᴀᴄᴇ, and XNʏsTʀᴀᴄᴇ estimators by a small multiple. (Though, resphering can help improve the variance-reduced estimators on problems like this, as in Fig. 14.3 below.)

Weakness of XNʏsTʀᴀᴄᴇ on `step` matrix. Perhaps the most interesting example of these four test matrices is the `step` matrix. This matrix has $k = 50$ large eigenvalues, with the remaining $n - 50$ eigenvalues being much smaller. It takes Hᴜᴛᴄʜ++ about $s = 3k = 150$ matvecs to produce a low-rank approximation capturing these dominant eigenvalues, and it takes XTʀᴀᴄᴇ about $s = 2k = 100$ matvecs. After hitting this number of matvecs, the error of Hᴜᴛᴄʜ++ and XTʀᴀᴄᴇ drop about three orders of magnitude (and even more with resphering, see Fig. 14.3 below). By contrast, XNʏsTʀᴀᴄᴇ begins reaping the benefits of low-rank approximation

at about $s = 50$ matvecs. Yet, the convergence of XNYSTRACE for $s > 50$ is more gradual than for either HUTCH++ and XTRACE. Consequently, HUTCH++ and XTRACE achieve lower error than XNYSTRACE for sufficiently large s.

An explanation for this behavior is visible in the error bounds for the randomized SVD and the single-pass Nyström approximation provided by Facts 2.5 and 2.11. For a psd matrix, the error of the single-pass Nyström approximation depends on the *sum* of the tail eigenvalues $\sum_{i \gtrsim k} \lambda_i$, whereas the randomized SVD error depends on the ℓ_2 *norm* of the tail eigenvalues $(\sum_{i \gtrsim k} \lambda_i^2)^{1/2}$. On this example, the sum of tail eigenvalues is larger than the ℓ_2 norm of the tail eigenvalues by a factor of about $\sqrt{n}$. Thus, the randomized SVD-based XTRACE and HUTCH++ estimators can approximate the `step` matrix better than the Nyström-based XNYSTRACE estimator.

Which method should I use? Unfortunately, the pattern of results in Fig. 14.1 defies a truly simple conclusion. Still, I think there are some pretty clear recommendations that can be gleaned from these experiments.

One can imagine two distinct settings for trace estimation. In the first setting, one is writing general-purpose software, and the trace estimator must be designed to handle arbitrary input matrices. In the second setting, one is interested in a specific application, and the trace estimator needs only work well for matrices appearing in that application. For the former setting, I would make the following recommendation:

> For general-purpose use, I would recommend XTRACE or, for psd matrices, either XTRACE or XNYSTRACE. Both should be implemented with resphering (Section 14.8).

While XTRACE and XNYSTRACE are not the best trace estimators for every single problem, the benefits over HUTCH++ and the Girard–Hutchinson estimator can be substantial on some problems. For problems with slow spectral decay (like the `flat` matrix), the resphering step (Section 14.8) can substantially improve XNYSTRACE.

In a specific application, the choice of trace estimator can be determined by profiling. As a rule of thumb, XNYSTRACE is the best estimator for psd matrices with *consistent* spectral decay, XTRACE is the best estimate for general matrices with at least some singular value decay, and the Girard–Hutchinson estimator is the best estimator (by a small margin) on problems with very little spectral decay. For additional approaches
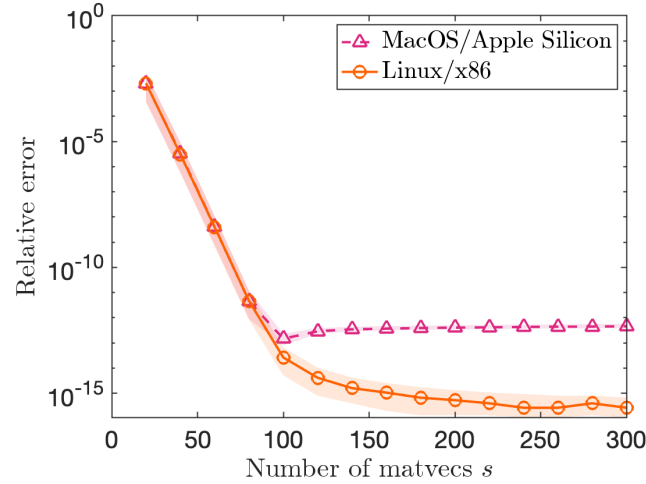
Figure 14.2: Comparison of the error of XNʏsTʀᴀᴄᴇ on `exp` matrix (14.27c) on Linux x86 machine (MATLAB Online, orange solid circles) and MacOS Apple Silicon (Macbook Pro with M3 ARM chip, pink dashed triangles). The error on the Linux system saturates at 2.7e-16 and the Mac system saturates at 4.6e-13, 1700× higher. Lines show median of 1000 trials and error bars show 10% and 90% quantiles.

to trace estimation, see Section 15.5. The paper [ETW24] for comparisons of XTʀᴀᴄᴇ with the Nʏsᴛʀöᴍ++ and adaptive Hᴜᴛᴄʜ++ algorithms of [PCK22].

Remark 14.6 (Platform dependence). Curiously, during the numerical experiments for this thesis, I observed that the numerical errors for XNʏsTʀᴀᴄᴇ were many orders of magnitude higher for matrices than in [ETW24]. In [ETW24], the results show XNʏsTʀᴀᴄᴇ achieving machine precision, whereas the new results showed the error saturating roughly $10^3\times$ higher. Eventually, I isolated the discrepancy to the platform; the original experiments in [ETW24] were performed on an Mac computer with an Intel x86 chip, and the new experiments werre performed on a Mac with an Apple Silicon ARM chip. I was able to reproduce the numerical behavior in the original paper [ETW24] on MATLAB Online, which uses x86 Linux machines. See Fig. 14.2 for a comparison.

I am not aware for an underlying reason for the platform dependence on the numerical accuracy. The distinction between x86 and Apple ARM systems persisted in every numerical experiment I ran across multiple machines, including multiple MATLAB versions up to 2024b. To show the best performance for the algorithm, the experiments in Fig. 14.1 were performed on MATLAB Online (x86).   ◇

14.8 Resphering XTrace and XNysTrace

We can improve the XTrace and XNysTrace algorithms by using the resphering technique, discussed at the end of Section 13.1.

Resphering XTrace

To resphere XTrace, we replace ω_i by

$$\nu_i := \sqrt{n - s/2 + 1} \cdot \frac{(\mathbf{I} - \boldsymbol{Q}_{(i)}\boldsymbol{Q}^*_{(i)})\omega_i}{\|(\mathbf{I} - \boldsymbol{Q}_{(i)}\boldsymbol{Q}^*_{(i)})\omega_i\|}.$$

The effect of this substitution is to scale the factor ⑧ in (14.9) by

$$\alpha_i := \frac{n - s/2 + 1}{\|(\mathbf{I} - \boldsymbol{Q}_{(i)}\boldsymbol{Q}^*_{(i)})\omega_i\|^2}.$$

Using the X matrix defined in (14.17), we evaluate the denominator to be

$$\|(\mathbf{I} - \boldsymbol{Q}_{(i)}\boldsymbol{Q}^*_{(i)})\omega_i\|^2 = \omega_i^*(\mathbf{I} - \boldsymbol{Q}(\mathbf{I} - s_i s_i^*)\boldsymbol{Q}^*)\omega_i = \|\omega_i\|^2 - \|\boldsymbol{x}_i\|^2.$$

Consequently, the vector of scaling factors is

$$\alpha = \frac{n - s/2 + 1}{\mathbf{scn}(\boldsymbol{\Omega}) - \mathbf{scn}(X)}.$$

As usual, division is performed elementwise. With resphering, the vector of XTrace vectors is given by

$$\widehat{\mathbf{tr}} = \text{tr}(\boldsymbol{H})\mathbf{1} - \mathbf{diagprod}(S, HS) + \alpha \odot [-\mathbf{diagprod}(T, X)$$
$$+ \mathbf{diagprod}(X, HX) + \mathbf{diagprod}(W, S) \odot \mathbf{diagprod}(S, R)].$$

An implementation of XTrace with resphering is provided in Program 14.4.

Resphering XNysTrace

To resphere XNysTrace requires a bit more thought. Remember that the basic XNysTrace estimators take the form

$$\widehat{\text{tr}}_i = \text{tr}(\widehat{\boldsymbol{A}}_{(i)}) + \omega_i^*(\boldsymbol{A} - \widehat{\boldsymbol{A}}^{(i)})\omega_i \quad \text{where } \widehat{\boldsymbol{A}}_{(i)} := \boldsymbol{A}\langle\boldsymbol{\Omega}_{-i}\rangle.$$

To resphere XNysTrace, we need to identify a matrix $\boldsymbol{G}$ such that $(\boldsymbol{A} - \widehat{\boldsymbol{A}}^{(i)})\boldsymbol{G} = \mathbf{0}$. Such a matrix is furnished the the *interpolatory property* of Nyström approximation (Proposition 2.9(d)), which shows that $\boldsymbol{G} = \boldsymbol{\Omega}_{-i}$ has this feature:

$$(\boldsymbol{A} - \widehat{\boldsymbol{A}}^{(i)})\boldsymbol{\Omega}_{-i} = \mathbf{0}.$$

Program 14.4 `xtrace_resphere.m`. Efficient implementation of XTRACE algorithm with resphering. Subroutines `diagprod` and `sqcolnorms` are provided in Programs 14.1 and F.4.

```matlab
function [tr, est] = xtrace_resphere(B,n,s)
% Input:  Function B() computing matrix products B(X) = B*X, number
%         of rows n, and number of matvecs s
% Output: Estimate tr of trace(B), estimate est of the error
%         abs(tr - trace(B))

% Define test matrix
k = floor(s/2);          % Approximation rank is s/2
Om = randn(n,k);         % Gaussian matrix

% Randomized SVD and downdate
Y = B(Om);               % Collect matvecs
[Q,R] = qr(Y,"econ");    % Randomized SVD
S = cnormc(inv(R'));     % Downdate RSVD, cnormc normalizes columns

% Compute other necessary matrices
Z = B(Q);                % Collect matvecs
H = Q'*Z;
W = Q'*Om;
T = Z'*Om;
X = W - S .* diagprod(W,S).';

% Scaling factor
alpha = (n - k + 1) ./ (sqcolnorms(Om) - sqcolnorms(X));

% Compute estimator, output
tr_vec = trace(H) * ones(k,1) - diagprod(S,H*S)...
    + alpha .* (-diagprod(T,X) + diagprod(X,H*X)...
              + diagprod(W, S) .* diagprod(S, R));
tr = mean(tr_vec);               % Trace estimate
est = std(tr_vec) / sqrt(k);     % Error estimate

end
```

Thus, we can resphere XNYSTRACE by replacing ω_i by

$$\nu_i := \sqrt{n - s + 1} \cdot \frac{(\mathbf{I} - \mathbf{Q}_{(i)}\mathbf{Q}_{(i)}^*)\omega_i}{\|(\mathbf{I} - \mathbf{Q}_{(i)}\mathbf{Q}_{(i)}^*)\omega_i\|} \quad \text{for } \mathbf{Q}_{(i)} := \mathbf{Orth}(\mathbf{\Omega}_{-i}).$$

To compute the vectors ν_i, we need to orthonormalize the columns of the matrix $\mathbf{\Omega}$ after every possible column deletion. Fortunately, the randomized SVD downdating formulas (Theorem 14.1) are exactly what we need to perform this computation.

Begin by computing a QR decomposition of $\boldsymbol{\Omega}$,

$$\boldsymbol{\Omega} = \boldsymbol{QT}, \tag{14.28}$$

and form the downdating matrix $\boldsymbol{S}$ by normalizing the columns of $\boldsymbol{T}^{-*}$. The downdated orthonormal matrices $\boldsymbol{Q}_{(i)}$ then admit the relation

$$\boldsymbol{Q}_{(i)}\boldsymbol{Q}_{(i)}^* = \boldsymbol{Q}(\mathbf{I} - \boldsymbol{s}_i \boldsymbol{s}_i^*)\boldsymbol{Q}^*.$$

Using this display, a short computation shows the resphered XNYsTRACE estimators with shifting are

$$\widehat{\mathbf{tr}} = \|\boldsymbol{F}\|_{\mathrm{F}}^2 \cdot \mathbf{1} - \mathbf{srn}(\boldsymbol{Z}) + \alpha \odot |\mathbf{diagprod}(\boldsymbol{Z}, \boldsymbol{\Omega})|^2 - n\mu \cdot \mathbf{1} \tag{14.29a}$$

where

$$\alpha := \frac{n - s + 1}{\mathbf{scn}(\boldsymbol{\Omega}) - \mathbf{scn}(\boldsymbol{X})} \quad \text{with } \boldsymbol{X} := \boldsymbol{T} - \boldsymbol{T}^{-*} \cdot \mathbf{Diag}(\mathbf{scn}(\boldsymbol{T}^{-*}))^{-1}. \tag{14.29b}$$

An interesting observation is that the resphered XNYsTRACE estimator (14.29) depends only on the triangular factor of the QR decomposition (14.28). As such, we can develop an implementation of the resphered XNYsTRACE estimator that avoids QR decomposition entirely, which is beneficial since QR decomposition is expensive. Indeed, we can instead compute a Cholesky decomposition

$$\boldsymbol{\Omega}^*\boldsymbol{\Omega} = \boldsymbol{T}^*\boldsymbol{T}$$

of the Gram matrix $\boldsymbol{\Omega}^*\boldsymbol{\Omega}$. Forming the Gram matrix is highly discouraged as a general practice in matrix computations [Hig22], but it is benign here because Gaussian random matrices are very well-conditioned (at least if $n \geq s/2$) [Tro21, §11.2 and p. 166]. An implementation of the resphered XNYsTRACE estimator is provided in Program 14.5.

Remark 14.7 (Improved resphered XNYsTRACE implementation). The publicly available code from [ETW24] uses a QR-based implementation of the resphered XNYsTRACE estimator. We have improved it here by introducing the faster Cholesky-based implementation. ◇

Experiments

Figure 14.3 compares the HUTCH++, XTRACE and XNYsTRACE algorithms and their resphered versions on the matrices `flat` and `step` defined in (14.27). We see that resphering significantly improves the performance of all estimators on the `flat` example and the XTRACE and HUTCH++ estimators on the `step` example.

Program 14.5 `xnystrace_resphere.m`. Efficient implementation of XNᴙꜱTʀᴀᴄᴇ algorithm with resphering. Subroutines `diagprod`, `sqcolnorms`, and `sqrownorms` are provided in Programs 14.1, F.3 and F.4.

```matlab
function [tr,est] = xnystrace_resphere(A,n,s)
% Input:  Function A() computing matrix products A(X) = A*X, number
%         of rows n, and number of matvecs s
% Output: Estimate tr of trace(A), estimate est of the error
%         abs(tr - trace(A))

Om = randn(n,s);                  % Gaussian random test matrix
Y = A(Om);                        % Matrix product Y = A*Om
mu = eps*norm(Y,"fro")/sqrt(n);   % Compute shift
Y = Y + mu * Om;                  % Apply shift to Y
H = Om'*Y;
R = chol((H+H')/2);               % Explicitly symmetrize H to be safe
F = Y/R;                          % Triangular substitution

% Downdating
Z = (F/R') .* (sqrownorms(inv(R)) .^ (-1/2))';

% Resphering
T = chol(Om'*Om);
Tinv = inv(T);
X = T - Tinv' / diag(sqcolnorms(Tinv'));
alpha = (n-s+1) ./ (sqcolnorms(Om) - sqcolnorms(X));

% Compute vector of estimates
tr_vec = norm(F,"fro")^2 * ones(s,1) - sqcolnorms(Z) ...
    + alpha .* abs(diagprod(Z,Om)) .^ 2 - mu * n * ones(s,1);
tr = mean(tr_vec);          % Trace estimate
est = std(tr_vec) / sqrt(s); % Error estimate

end
```

14.9 Application: Estrada index

As a running example throughout this part of the thesis, we will apply trace and diagonal estimators to problems in network science. Given a graph with adjacency matrix M, the *Estrada index* [Est22] is defined as the trace-exponential of the adjacency matrix:

$$\text{estr} := \text{tr}(\exp(M)).$$

The Estrada index is a measure of *centralization* for a graph, that is, how "clustered" or "spread out" its nodes are. This quantity is an ideal candidate for variance-reduced trace estimators like XTʀᴀᴄᴇ, as the exponential function promotes spectral decay
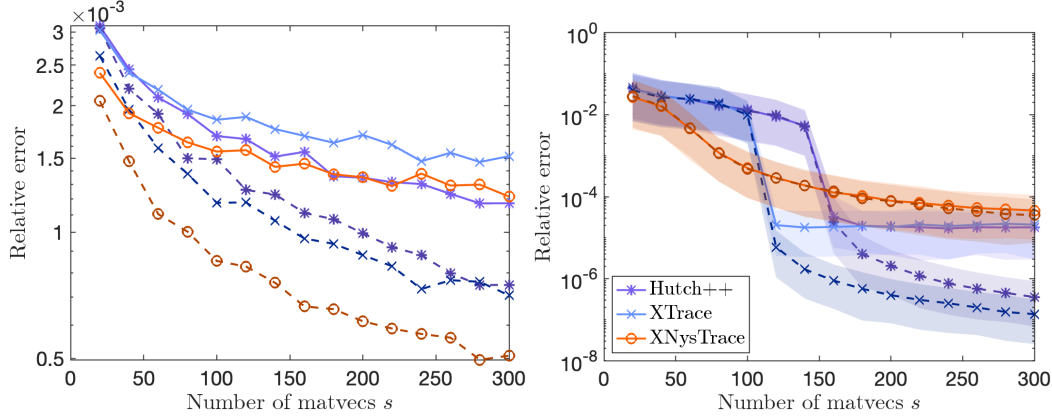
Figure 14.3: Comparison of Hutch++, XTrace and XNysTrace algorithms (light solid lines) and their resphered versions (dark dashed lines) on the `flat` ((14.27a), *left*) and `step` matrices ((14.27d), *right*). Lines show median of 100 trials, and error bars on the right panel show 10% and 90% quantiles. (Error bars are omitted on the left panel for clarity.)
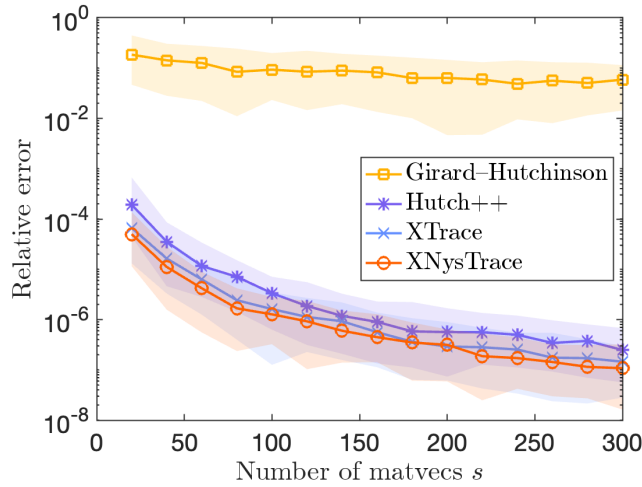
Figure 14.4: Relative error of trace estimates by Girard–Hutchinson estimator (yellow squares), Hutch++ (purple asterisks), XTrace (blue crosses), and XNysTrace (orange circles) for Estrada index problem as a function of the number of matvecs s. Lines show median of 100 trials, and error bars show 10% and 90% quantiles.

in the matrix $A := \exp(M)$. The matrix A is psd, allowing us to apply estimators designed for psd matrices like XNysTrace.

Figure 14.4 provides a demonstration of various trace estimators applied to the Estrada index. I test on the `yeast` network $M \in \mathbb{R}^{2361 \times 2361}$ from [BZCX+03]; for this small example, the exact values of the Estrada index can be computed for reference. To obtain matvecs with $A = \exp(M)$, I employ forty steps of the Lanczos algorithm [Che24, Ch. 6], which is sufficient to compute matvecs

to relative error 10^{-11}. I implemented all trace estimators with resphering. The conclusions are much the same as in Figs. 14.1 and 14.3, with ranking from best-to-worst XN*ys*Trace, XTrace, Hutch++, Girard–Hutchinson. On this example, the difference between XNysTrace, XTrace, and Hutch++ is comparatively small, and all of these variance-reduced methods leading to substantial accuracy improvements over the Girard–Hutchinson estimator.

14.10 The leave-one-out approach: Summary

This chapter described the leave-one-out approach to designing randomized algorithms for matrix attribute estimation. At a high level, this approach proceeds in five steps:

1. Compute a low-rank approximation to the input matrix by multiplying it against a collection of test vectors.

2. Decompose the quantity of interest into known piece depending on the low-rank approximation plus a residual.

3. Construct a Monte Carlo estimate of the residual using a single random vector.

4. Downdate the low-rank approximation by recomputing it with a test vector removed, and use the left-out test vector as the random vector in step 3.

5. Average the estimator from step 4 over all choices of vectors to leave out.

We employed this approach to derive the XTrace and XNysTrace estimators, which improve on Hutch++ by using an exchangeable design. These algorithms all the matrix–vector products *both* for low-rank approximation *and* residual trace estimation. We will see more leave-one-out randomized matrix algorithms in upcoming chapters.

The leave-one-out approach is simple, but deriving efficient implementations using it required some effort. To derive fast algorithms, our main tools were the downdating formulas for the randomized SVD (Theorem 14.1) and randomized Nyström approximation (Theorem 14.2). The resphering technique further improves leave-one-out estimators, at the cost of some additional complexity.

Chapter 15

MORE ON TRACE ESTIMATION

> *From 1987 to 2020, an algorithm called Hutchinson's Estimator was the state-of-the-art for the trace estimation problem, with analysis giving matching upper and lower bounds at $\Theta(1/\varepsilon^2)$ Matrix–Vector products. Notably, this $\Omega(1/\varepsilon^2)$ lower bound was known only for this estimator, and was not known to hold in general [WWZ14] We tried to generalize that lower bound to hold for arbitrary Matrix-Vector algorithms, but that didn't work. While looking into why, we instead found an algorithm that only used $O(1/\varepsilon)$ Matrix-Vector products, called HUTCH++.*

> Raphael A. Meyer

The last chapter introduced the leave-one-out approach for designing randomized algorithms for matrix computations. As a demonstration of that technique, we saw the XTRACE and XNYSTRACE algorithms for trace estimation and developed efficient implementations using downdating formulas for the randomized SVD and Nyström approximation.

In this chapter, we will continue our discussion of trace estimation by discussing *a priori* error analysis, *a posteriori* error estimation, adaptive determination of parameters, and alternatives to XTRACE and XNYSTRACE.

Sources. This chapter is based on the XTRACE paper

Ethan N. Epperly, Joel A. Tropp, and Robert J. Webber. "XTrace: Making the Most of Every Sample in Stochastic Trace Estimation". In: *SIAM Journal on Matrix Analysis and Applications* (Mar. 2024), pp. 1–23. DOI: 10.1137/23M1548323.

Several aspects of the presentation have been significantly expanded, including the discussion of how to interpret the XTRACE error bounds in Section 15.2, the discussion of adaptivity in Section 15.4, and the discussion of trace estimation alternatives in Section 15.5.

Outline. Section 15.1 presents *a priori* error bounds for XTRACE and XNYSTRACE, and Section 15.2 discusses how to interpret them. Section 15.3 describes how to estimate the error of XTRACE and XNYSTRACE *a posteriori*, and Section 15.4 explains how these error estimates can be used to implement XTRACE and XNYSTRACE

adaptively to meet an error tolerance. Finally, Section 15.5 presents alternatives to XTRACE and XNYSTRACE for trace estimation.

15.1 *A priori* error bounds

In this section, we introduce *a priori* error bounds for XTRACE and XNYSTRACE, analogous to Theorem 13.5 for HUTCH++. Versions of these bounds originally appeared as [ETW24, Thm. 1.1]. The first result is the following bound for XTRACE:

Theorem 15.1 (XTRACE: mean-squared error bound). *Let $B \in \mathbb{R}^{n \times n}$ be a* real *square matrix, and consider the XTRACE estimator with test vectors $\omega_1, \ldots, \omega_{s/2}$ drawn iid from the standard normal $\text{NORMAL}_{\mathbb{R}}(\mathbf{0}, \mathbf{I}_n)$ or sphere $\sqrt{n} \cdot \mathbb{S}(\mathbb{R}^n)$. This estimator $\widehat{\text{tr}}_X$ is an unbiased estimate for the trace $\text{tr}(B)$ with mean-squared error*

$$\text{Var}(\widehat{\text{tr}}_X) \leq \min_{r \leq s/2-4} \left[\frac{4e^2 s}{(s/2 - r - 3)^2} \|B - [\![B]\!]_r\|_F^2 + \frac{4s}{s/2 - r - 3} \|B - [\![B]\!]_r\|^2 \right]. \tag{15.1}$$

In particular, the root-mean-squared error decreases at a rate of at least $O(s^{-1})$:

$$[\mathbb{E}(\widehat{\text{tr}}_X - \text{tr}(B))^2]^{1/2} \leq \frac{\text{const}}{s} \cdot \|B\|_*. \tag{15.2}$$

An analogous result holds for XNYSTRACE [ETW24, Thm. 1.1]:

Theorem 15.2 (XNYSTRACE: mean-squared error bound). *Let $A \in \mathbb{R}^{n \times n}$ be a* real *psd matrix and consider the XNYSTRACE estimator with test vectors $\omega_1, \ldots, \omega_{s/2}$ drawn iid from the standard normal $\text{NORMAL}_{\mathbb{R}}(\mathbf{0}, \mathbf{I}_n)$ or sphere $\sqrt{n} \cdot \mathbb{S}(\mathbb{R}^n)$ distributions. Then the XNYSTRACE estimator $\widehat{\text{tr}}_{XN}$ provicdes an unbiased estimate for $\text{tr}(A)$ with root-mean-squared error*

$$\text{Var}(\widehat{\text{tr}}_{XN})^{1/2} \leq s \min_{r \leq s-6} \left[\frac{5e^2}{(s - r - 5)^2} \|A - [\![A]\!]_r\|_* \right.$$
$$\left. + \frac{\sqrt{2}}{(s - r - 5)^{3/2}} \|A - [\![A]\!]_r\|_F + \frac{\sqrt{8}}{s - r - 5} \|A - [\![A]\!]_r\| \right]. \tag{15.3}$$

In particular, the root-mean-squared error decreases at a rate of at least $O(s^{-1})$:

$$[\mathbb{E}(\widehat{\text{tr}}_{XN} - \text{tr}(A)^2]^{1/2} \leq \frac{\text{const}}{s} \cdot \text{tr}(A). \tag{15.4}$$

These bounds, particularly the HUTCH++-style $O(1/s)$ "convergence rates" of XTRACE (15.2) and XNYSTRACE (15.4) are easy to misinterpret; we will elaborate more on what these bounds do and do not mean in Section 15.2.

For now, let us focus on the main error bounds (15.1) and (15.3), which are more informative than the $O(1/s)$ convergence rate but perhaps harder to interpret. To help make things simpler, we remind the reader of the comparison (14.22) above. Let A be a psd matrix with eigenvalues decaying at an exponential rate $\lambda_i(A) \leq \alpha^i$ for $\alpha \in (0, 1)$. Then the XTrace (Theorem 15.1, XNysTrace (Theorem 15.2), and Hutch++ (Theorem 13.5) theorems imply that the root-mean-squared errors decay at rates

$$
\begin{aligned}
\left[\mathbb{E}\big(\widehat{\mathrm{tr}}_{\mathrm{H}++} - \mathrm{tr}(A)\big)^2\right]^{1/22} &\leq & c_1(\alpha)\, \alpha^{s/3}; \\
\left[\mathbb{E}\big(\widehat{\mathrm{tr}}_{\mathrm{X}} - \mathrm{tr}(A)\big)^2\right]^{1/2} &\leq s^{1/2}\, c_2(\alpha)\, \alpha^{s/2}; \\
\left[\mathbb{E}\big(\widehat{\mathrm{tr}}_{\mathrm{XN}} - \mathrm{tr}(A)\big)^2\right]^{1/2} &\leq & s\, c_3(\alpha)\, \alpha^s.
\end{aligned}
\tag{15.5}
$$

Here, the prefactors $c_1(\alpha), c_2(\alpha), c_3(\alpha) > 0$ depend only on α. We see that XNysTrace and XTrace converge at 3× and 1.5× the rate of Hutch++, respectively. These theoretically predicted convergence rates are mirrored in the `exp` example in Fig. 14.1.

We will prove the XTrace bound (Theorem 15.1) and omit the proof of the XNys-Trace bound (Theorem 15.2), which is similar. The essence of the theorem is contained in the following structural bound.

Lemma 15.3 (XTrace: Structural bound). *Import the setting of Theorem 15.1. The XTrace estimator is unbiased* $\mathbb{E}[\widehat{\mathrm{tr}}_{\mathrm{X}}] = \mathrm{tr}(B)$ *and the variance admits the bound*

$$
\mathrm{Var}(\widehat{\mathrm{tr}}_{\mathrm{X}}) \leq \frac{4}{s} \cdot \mathbb{E}\left\|(\mathbf{I} - \boldsymbol{Q}_{(1)}\boldsymbol{Q}_{(1)}^*)B\right\|_{\mathrm{F}}^2 + 4\,\mathbb{E}\left\|(\mathbf{I} - \boldsymbol{Q}_{(12)}\boldsymbol{Q}_{(12)}^*)B\right\|^2.
$$

Here, $\boldsymbol{Q}_{(1)} = \mathbf{Orth}(B\boldsymbol{\Omega}_{-1})$ *and* $\boldsymbol{Q}_{(12)} = \mathbf{Orth}(B\boldsymbol{\Omega}(:, 3 : s/2))$.

Before we prove this lemma, let us use it to establish Theorem 15.1.

Proof of Theorem 15.1. Combining the XTrace structural bound Lemma 15.3 and the randomized SVD bound (Fact 2.5) gives

$$
\begin{aligned}
\mathrm{Var}(\widehat{\mathrm{tr}}_{\mathrm{X}}) &\leq \frac{4}{s} \min_{r \leq s/2-3} \frac{s/2 - 2}{s/2 - r - 2}\big\|B - [\![B]\!]_r\big\|_{\mathrm{F}}^2 \\
&\quad + 4 \min_{r \leq s/2-4} \frac{s/2 + r - 3}{s/2 - r - 3}\left(\big\|B - [\![B]\!]_r\big\|^2 + \frac{\mathrm{e}^2}{s/2 - r - 2}\big\|B - [\![B]\!]_r\big\|_{\mathrm{F}}^2\right).
\end{aligned}
$$

Simplifying, we obtain (15.1). To prove (15.2), we set $r \coloneqq \lfloor s/4 \rfloor - 4$ and invoke Fact 13.6. $\qquad\square$

We now prove the lemma:

Proof of Lemma 15.3. The unbiasedness of the XTRACE estimator follows form its construction; see [ETW24, p. 18] for a detailed proof.

By bilinearity of the variance and since every pair $(\widehat{\text{tr}}_i, \widehat{\text{tr}}_j)$ has the same distribution, we have

$$\text{Var}(\widehat{\text{tr}}_X) = \frac{2}{s} \cdot \text{Var}(\widehat{\text{tr}}_1) + \left(1 - \frac{2}{s}\right) \text{Cov}(\widehat{\text{tr}}_1, \widehat{\text{tr}}_2). \tag{15.6}$$

We will bound the variance $\text{Var}(\widehat{\text{tr}}_1)$ and the covariance $\text{Cov}(\widehat{\text{tr}}_1, \widehat{\text{tr}}_2)$ separately.

To evaluate $\text{Var}(\widehat{\text{tr}}_1)$, use the chain rule for the variance:

$$\text{Var}(\widehat{\text{tr}}_1) = \mathbb{E}[\text{Var}(\widehat{\text{tr}}_1 \mid \mathbf{\Omega}_{-1})] + \text{Var}(\mathbb{E}[\widehat{\text{tr}}_1 \mid \mathbf{\Omega}_{-1}]). \tag{15.7}$$

To compute the second term in (15.7), we observe that $\mathbb{E}[\widehat{\text{tr}}_1 \mid \mathbf{\Omega}_{-1}] = \text{tr}(\boldsymbol{B})$. That is, even conditional on the other test vectors $\omega_2, \ldots, \omega_{s/2}$, the first basic XTRACE estimator $\widehat{\text{tr}}_1$ is unbiased. Thus, the second term in (15.7) is zero. To evaluate the first term of (15.7), we invoke Fact 13.2 to obtain

$$\begin{aligned} \text{Var}(\widehat{\text{tr}}_1 \mid \mathbf{\Omega}_{-1}) &= \text{Var}(\omega_i^*(\mathbf{I} - \boldsymbol{Q}_{(i)}\boldsymbol{Q}_{(i)}^*)\boldsymbol{B}(\mathbf{I} - \boldsymbol{Q}_{(i)}\boldsymbol{Q}_{(i)}^*)\omega_i \mid \mathbf{\Omega}_{-1}) \\ &\leq 2\left\|(\mathbf{I} - \boldsymbol{Q}_{(i)}\boldsymbol{Q}_{(i)}^*)\boldsymbol{B}(\mathbf{I} - \boldsymbol{Q}_{(i)}\boldsymbol{Q}_{(i)}^*)\right\|_F^2. \end{aligned}$$

Thus, we have

$$\text{Var}(\widehat{\text{tr}}_1) = \mathbb{E}[\text{Var}(\widehat{\text{tr}}_1 \mid \mathbf{\Omega}_{-1})] \leq 2\left\|(\mathbf{I} - \boldsymbol{Q}_{(i)}\boldsymbol{Q}_{(i)}^*)\boldsymbol{B}\right\|_F^2. \tag{15.8}$$

We have dropped the second projector $\mathbf{I} - \boldsymbol{Q}_{(i)}\boldsymbol{Q}_{(i)}^*$, which can only increase the Frobenius norm.

Now, we bound $\text{Cov}(\widehat{\text{tr}}_1, \widehat{\text{tr}}_2)$. Introduce

$$\boldsymbol{X} \coloneqq (\mathbf{I} - \boldsymbol{Q}_{(12)}\boldsymbol{Q}_{(12)}^*)\boldsymbol{B}(\mathbf{I} - \boldsymbol{Q}_{(12)}\boldsymbol{Q}_{(12)}^*). \tag{15.9}$$

This matrix is independent of both ω_1 and ω_2 by design. Therefore, we may write

$$\begin{aligned} \text{Cov}(\widehat{\text{tr}}_1, \widehat{\text{tr}}_2) &= \mathbb{E}[(\widehat{\text{tr}}_1 - \text{tr}(\boldsymbol{B}))(\widehat{\text{tr}}_2 - \text{tr}(\boldsymbol{B}))] \\ &= \mathbb{E}[(\widehat{\text{tr}}_1 - \text{tr}(\boldsymbol{B}) + \text{tr}(\boldsymbol{X}) - \omega_1^*\boldsymbol{X}\omega_1)(\widehat{\text{tr}}_2 - \text{tr}(\boldsymbol{B}))] \\ &= \mathbb{E}[(\widehat{\text{tr}}_1 - \text{tr}(\boldsymbol{B}) + \text{tr}(\boldsymbol{X}) - \omega_1^*\boldsymbol{X}\omega_1)(\widehat{\text{tr}}_2 - \text{tr}(\boldsymbol{B}) + \text{tr}(\boldsymbol{X}) - \omega_2^*\boldsymbol{X}\omega_2)]. \end{aligned}$$

The second line follows because $\widehat{\text{tr}}_2$ is unbiased approximation to $\boldsymbol{B}$, even conditional on $(\omega_i : i \neq 2)$. The third line follows because the first factor is mean zero, even

conditionally on $(\omega_i : i \neq 1)$. Now, invoke Cauchy–Schwarz and the fact that the two factors are identifically distributed to obtain

$$\mathrm{Cov}(\widehat{\mathrm{tr}}_1, \widehat{\mathrm{tr}}_2) \leq \mathbb{E}\left|\widehat{\mathrm{tr}}_1 - \mathrm{tr}(\boldsymbol{B}) + \mathrm{tr}(\boldsymbol{X}) - \omega_1^* \boldsymbol{X} \omega_1\right|^2 \tag{15.10}$$

Conditional on $(\omega_i : i \neq 1)$, the terms $\mathrm{tr}(\boldsymbol{B})$ and $\mathrm{tr}(\boldsymbol{X})$ are constants and the quantity in the absolute value signs is mean-zero. Furthermore, the first term of

$$\widehat{\mathrm{tr}}_1 = \mathrm{tr}(\boldsymbol{Q}_{(1)}^* \boldsymbol{B} \boldsymbol{Q}_{(1)}) + \omega_1^*(\mathbf{I} - \boldsymbol{Q}_{(1)}\boldsymbol{Q}_{(1)}^*)\boldsymbol{B}(\mathbf{I} - \boldsymbol{Q}_{(1)}\boldsymbol{Q}_{(1)}^*)\omega_1,$$

is also conditionally constant. Therefore, we obtain,

$$\mathrm{Cov}(\widehat{\mathrm{tr}}_1, \widehat{\mathrm{tr}}_2) \leq \mathbb{E}\left[\mathrm{Var}\left(\omega_1^*[(\mathbf{I} - \boldsymbol{Q}_{(1)}\boldsymbol{Q}_{(1)}^*)\boldsymbol{B}(\mathbf{I} - \boldsymbol{Q}_{(1)}\boldsymbol{Q}_{(1)}^*) - \boldsymbol{X}]\omega_1 \mid \boldsymbol{\Omega}_{-1}\right)\right].$$

Using Fact 13.2, we bound the right-hand side as

$$\mathrm{Cov}(\widehat{\mathrm{tr}}_1, \widehat{\mathrm{tr}}_2) \leq 2\,\mathbb{E}\left\|(\mathbf{I} - \boldsymbol{Q}_{(1)}\boldsymbol{Q}_{(1)}^*)\boldsymbol{B}(\mathbf{I} - \boldsymbol{Q}_{(1)}\boldsymbol{Q}_{(1)}^*) - \boldsymbol{X}\right\|_{\mathrm{F}}^2. \tag{15.11}$$

We now do some wrangling to bound the right-hand side of (15.11). Introduce the projectors $\boldsymbol{\Pi}_1 := \boldsymbol{Q}_{(1)}\boldsymbol{Q}_{(1)}^*$ and $\boldsymbol{\Pi}_{12} := \boldsymbol{Q}_{(12)}\boldsymbol{Q}_{(12)}^*$, and note the identity

$$\mathbf{I} - \boldsymbol{\Pi}_1 = (\mathbf{I} - \boldsymbol{\Pi}_1)(\mathbf{I} - \boldsymbol{\Pi}_{12}) = (\mathbf{I} - \boldsymbol{\Pi}_{12})(\mathbf{I} - \boldsymbol{\Pi}_1). \tag{15.12}$$

We compute

$$
\begin{aligned}
\|(\mathbf{I} - \boldsymbol{\Pi}_1)\boldsymbol{B}(\mathbf{I} - \boldsymbol{\Pi}_1) - \boldsymbol{X}\|_{\mathrm{F}}^2 &= \|(\mathbf{I} - \boldsymbol{\Pi}_1)(\mathbf{I} - \boldsymbol{\Pi}_{12})\boldsymbol{B}(\mathbf{I} - \boldsymbol{\Pi}_{12})(\mathbf{I} - \boldsymbol{\Pi}_1) - \boldsymbol{X}\|_{\mathrm{F}}^2 \\
&= \|(\mathbf{I} - \boldsymbol{\Pi}_1)\boldsymbol{X}(\mathbf{I} - \boldsymbol{\Pi}_1) - \boldsymbol{X}\|_{\mathrm{F}}^2 \\
&= \|(\mathbf{I} - \boldsymbol{\Pi}_1)\boldsymbol{X}(\mathbf{I} - \boldsymbol{\Pi}_1) - (\mathbf{I} - \boldsymbol{\Pi}_1)\boldsymbol{X}\mathbf{I} - \boldsymbol{\Pi}_1\boldsymbol{X}\|_{\mathrm{F}}^2 \\
&= \|-(\mathbf{I} - \boldsymbol{\Pi}_1)\boldsymbol{X}\boldsymbol{\Pi}_1 - \boldsymbol{\Pi}_1\boldsymbol{X}\|_{\mathrm{F}}^2 \\
&= \|(\mathbf{I} - \boldsymbol{\Pi}_1)\boldsymbol{X}\boldsymbol{\Pi}_1\|_{\mathrm{F}}^2 + \|\boldsymbol{\Pi}_1\boldsymbol{X}\|_{\mathrm{F}}^2 \\
&\leq \|\boldsymbol{X}\boldsymbol{\Pi}_1\|_{\mathrm{F}}^2 + \|\boldsymbol{\Pi}_1\boldsymbol{X}\|_{\mathrm{F}}^2 \\
&= \|\boldsymbol{X}(\boldsymbol{\Pi}_1 - \boldsymbol{\Pi}_{12})\|_{\mathrm{F}}^2 + \|(\boldsymbol{\Pi}_1 - \boldsymbol{\Pi}_{12})\boldsymbol{X}\|_{\mathrm{F}}^2 \\
&= \|\boldsymbol{X}(\boldsymbol{\Pi}_1 - \boldsymbol{\Pi}_{12})\|^2 + \|(\boldsymbol{\Pi}_1 - \boldsymbol{\Pi}_{12})\boldsymbol{X}\|^2 \\
&\leq 2\|\boldsymbol{X}\|^2.
\end{aligned}
$$

The first line is the identity (15.12), the second line is the definition of $\boldsymbol{X}$ (15.9), the fifth line is the Pythagorean theorem, the sixth line is the nonexpansiveness of the orthoprojector $\mathbf{I} - \boldsymbol{\Pi}_1$, the seventh line invokes the identity $\boldsymbol{X}\boldsymbol{\Pi}_{12} = \boldsymbol{0}$, the eighth line observes that $\boldsymbol{X}(\boldsymbol{\Pi}_1 - \boldsymbol{\Pi}_{12})$ is a rank-one matrix, and the final line holds again

because the orthoprojector $\mathbf{\Pi}_1 - \mathbf{\Pi}_{12}$ is nonexpansive. Combining this result with (15.11) and using the definition of X, we obtain

$$\mathrm{Cov}(\widehat{\mathrm{tr}}_1, \widehat{\mathrm{tr}}_2) \leq 4 \, \mathbb{E} \left\| (\mathbf{I} - \boldsymbol{Q}_{(12)} \boldsymbol{Q}^*_{(12)}) \boldsymbol{B} (\mathbf{I} - \boldsymbol{Q}_{(12)} \boldsymbol{Q}^*_{(12)}) \right\|^2.$$

Substituting this covariance bound and the variance bound (15.8) into the variance decomposition (15.6) yields the stated result. $\square$

15.2 How to interpret the HUTCH++ and XTRACE error bounds

When the HUTCH++ paper was released, it generated a large amount of interest, both among specialists in randomized matrix computations and researchers in adjacent areas such as machine learning, quantum information, and scientific computing. In my view, part of this interest can be attributed to the the HUTCH++ paper showing that its method is *optimal*. Speaking precisely, the paper [MMMW21b] shows that HUTCH++ is optimal in the following sense:

> **Optimality of HUTCH++.** HUTCH++ produces an approximation to the trace of a *psd* matrix A with relative error ε using $s = O(1/\varepsilon)$ matvecs. Conversely, there is no algorithm that produces a trace estimate up to relative error ε *for an arbitrary psd input matrix* in fewer than $s = O(1/[\varepsilon \log(1/\varepsilon)])$ matvecs.

This lower bound was later improved to $s = O(1/\varepsilon)$ [Mey24], establishing that the $s = O(1/\varepsilon)$ rate of HUTCH++ is optimal *even up to logarithmic factors*. I consider the matching $O(1/\varepsilon)$ complexity upper and lower bounds for trace estimation to be one of the most beautiful results in the field of randomized matrix computations.

Given these facts, it is very tempting to invert the relation $s \sim 1/\varepsilon$ to claim that the *convergence rate* for HUTCH++ is $\varepsilon \sim 1/s$. Based on this analysis, we would expect that plotting the error versus number of matvecs for HUTCH++ on a log–log plot should result in a curve with a slope of -1. Figure 15.1 tests this proposition. Here, we evaluate HUTCH++ (*left*) and XTRACE (*right*) on a family of psd matrices of dimension $n = 1000$ with eigenvalues $\lambda_i(A) = i^{-\alpha}$ for each fixed $\alpha = 0, 0.5, \ldots, 3$. Darker curves show lower values of α and lighter curves show higher values. We report the average relative error over 100 trials. For reference, the curve $1/s$ is marked as a black dashed line. Perhaps surprisingly, the error-versus-matvec curves for both HUTCH++ show a number of convergence rates that are both faster *and*
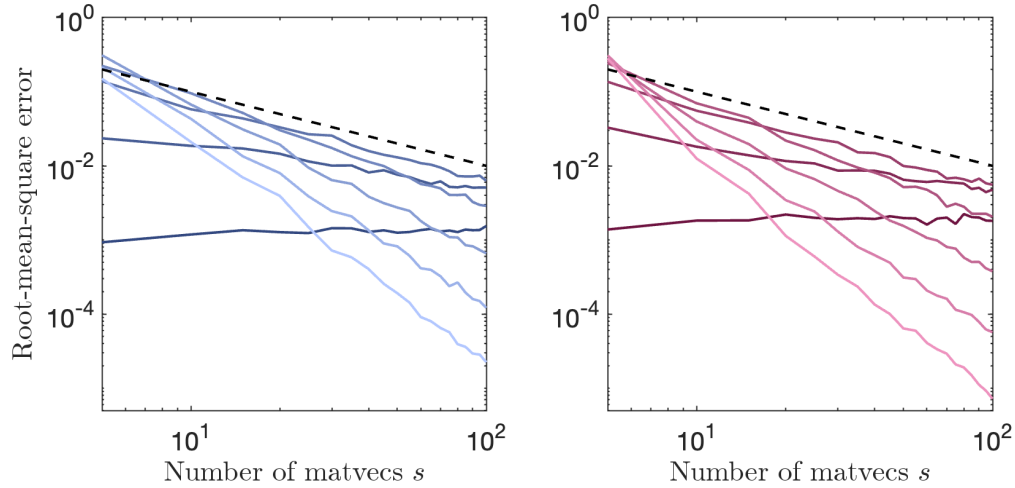
Figure 15.1: Root-mean-squared errors for HUTCH++ (*left*) and XTRACE (*right*) on the family of psd matrices with eigenvalues $\lambda_i(A) = i^{-\alpha}$ for $\alpha = 0, 0.5, \ldots, 3$. Higher values of α are indicated using lighter shades, and the the dashed black line shows $\varepsilon = 1/s$. Root-mean-squared errors are estimated using 100 trials.

slower than the error $\sim 1/s$ convergence rate we predicted from HUTCH++ analysis. What's going on?

Part of the issue is a cultural difference between how different researchers talk about the complexity or rate of convergence of an algorithm.

- Under one view, which might be labeled the ***computational mathematics view***, the performance of an algorithm is determined by its *convergence rate*. In the context of trace estimate, the convergence rate is the number β such that the error ε scales like s^β (or better). Equivalently, the convergence rate β is gives upper bound on the slope of a log–log plot of the error plotted against the number of matvecs. Under this interpretation, an algorithm has a convergence rate of $1/s$ only if doubling s halves the error (or better). As Fig. 15.1 shows, the convergence rate of HUTCH++ and XTRACE can be both slower and faster than $1/s$ for different input matrices.

- A second perspective, which we might label as the ***theoretical computer science view***, is concerned with the amount of computational resources (in this case, matvecs s) required to achieve an of error ε for a *worst-case input matrix*. Under this viewpoint, the complexity of an algorithm is $s = O(1/\varepsilon)$ if the error-versus-matvec curve is *bounded above* by a curve of the form "const/s" for all $s \le n$. It is in this second sense in which HUTCH++ is optimal and has a complexity of $s = O(1/\varepsilon)$.

Having understood these two viewpoints, we reconcile the perhaps surprising results in Fig. 15.1 . The error analysis of HUTCH++ and XTRACE does *not* say that their *convergence rate* is $O(1/s)$, just that the error-versus-matvec curve is bounded above by a curve of the form "const/s" for all $s \le n$. Indeed, this is exactly the case in Fig. 15.1: The error curves for both HUTCH++ and XTRACE are bounded above by $3/s$ for each of the seven test matrices.

This understanding of the HUTCH++ and XTRACE error bounds delivers both good news and bad news about how these algorithms perform.

The good: Spectral convergence. On the positive side, HUTCH++ and XTRACE often converge *much faster* than the $O(1/s)$ analysis would suggest. In fact, they converge *spectrally*, with the trace errors decreasing at a rate proportional to the error of the best rank-$O(s)$ approximation, measured in an appropriate norm. This phenomenon was demonstrated theoretically in (15.5), which shows that the variance-reduced trace estimators converge *geometrically* when the eigenvalues decay geometrically. Spectral convergence is also demonstrated empirically in Fig. 15.1, where several curves are seen to decrease significantly more rapidly then the dashed $O(1/s)$ rate. This spectral convergence means that stochastic trace estimators are very effective at estimating the trace of a matrix with rapidly decaying eigenvalues, such a matrix of the form $\exp(-\beta A)$ arising from partition function calculations in quantum physics.

The bad: HUTCH++ and XTRACE are not always better than Girard–Hutchinson. The Girard–Hutchinson *converges at* a rate of $O(1/s^{1/2})$, in both the "computational math" and "theoretical computer science" senses. Given the $s = O(1/\varepsilon)$ complexity of HUTCH++/XTRACE and the $s = O(1/\varepsilon^2)$ complex of the Girard–Hutchinson estimator, one might surmise that HUTCH++ or XTRACE will always achieve lower error than the Girard–Hutchinson estimator, except possibly for very small values of s. Unfortunately, this is not the case.

> For problems with sufficiently slow spectral decay, the Girard–Hutchinson estimator will be (slightly) more accurate than HUTCH++ and XTRACE.

Let us justify this claim by applying both HUTCH++ and the Girard–Hutchinson estimator to the identity matrix $A = I$. To yield generalizable conclusions, we use *Gaussian* test vectors $\omega \sim \text{NORMAL}_{\mathbb{R}}(\mathbf{0}, \mathbf{I})$—these are a bad choice for computation in practice, but they suffice to illustrate the conceptual point. By Fact 13.2, the

variance of the Girard–Hutchinson estimator for this problem is

$$\text{Var}(\widehat{\text{tr}}_{\text{GH}}) = \frac{2}{s}\|\mathbf{I}\|_{\text{F}}^2 = \frac{2n}{s}.$$

Consequently, the root-mean-squared *relative* error is

$$\left[\mathbb{E}\left(\frac{\text{tr}(\mathbf{I}) - \widehat{\text{tr}}_{\text{GH}}}{\text{tr}(\mathbf{I})}\right)^2\right]^{1/2} = \frac{\sqrt{2}}{s^{1/2}} \cdot \frac{1}{n^{1/2}}.$$

Now, let us turn to Hutch++. For the matrix $\mathbf{Q}$ produced by the Hutch++ algorithm, the matrix $(\mathbf{I} - \mathbf{Q}\mathbf{Q}^*)\mathbf{I}(\mathbf{I} - \mathbf{Q}\mathbf{Q}^*)$ is a compression of the identity matrix to a uniformly random subspace of dimension $n - s/3$. Thus,

$$\text{Var}(\widehat{\text{tr}}_{\text{H++}}) = \frac{2}{s/3}\|(\mathbf{I} - \mathbf{Q}\mathbf{Q}^*)\mathbf{I}(\mathbf{I} - \mathbf{Q}\mathbf{Q}^*)\|_{\text{F}}^2 = \frac{6(n - s/3)}{s}.$$

Ergo, the root-mean-squared relative error for Hutch++ is

$$\left[\mathbb{E}\left(\frac{\text{tr}(\mathbf{I}) - \widehat{\text{tr}}_{\text{H++}}}{\text{tr}(\mathbf{I})}\right)^2\right]^{1/2} = \frac{\sqrt{6}}{s^{1/2}} \cdot \frac{(n - s/3)^{1/2}}{n}.$$

We observe two things: First, the *convergence rate* of Hutch++ on this example is $1/s^{1/2}$, the same as the Girard–Hutchinson estimator. Second, except for large values $s \approx 3n$, the root-mean-squared error for Hutch++ is larger than the Girard–Hutchinson estimator by about $\sqrt{3}$. This simple computation demonstrates that, for problems with slow spectral decay, the Girard–Hutchinson estimator can be better than Hutch++ by a constant factor. The same qualitative conclusions hold for XTrace and XNysTrace.

15.3 Posterior error estimation

In the past sections, we studied *a priori* bounds on the error for trace estimation algorithms. Sharp forms of these error bounds are typically inaccessible in applications, as they depend on the singular values of the matrix $\mathbf{B}$. As such, it can be helpful to have *posterior* bounds on the error for trace estimation algorithms. This section develops posterior error estiamtes for the XTrace and XNysTrace algorithms.

Our approach to error estimation is straightforward. The XTrace and XNysTrace algorithms generate a *family* of trace estimates $\widehat{\text{tr}}_i$ for $i = 1, \ldots, \ell$ with $\ell = s/2$ (XTrace) or $\ell = s$ (XNysTrace). The scaled standard deviation of these estimates serves as a natural posterior estimate for the error $|\widehat{\text{tr}}_{\text{X}} - \text{tr}(\mathbf{B})|$ or $|\widehat{\text{tr}}_{\text{XN}} - \text{tr}(\mathbf{A})|$. Specifically, we define the error estimate

$$\widehat{\text{Err}}^2 := \frac{1}{\ell(\ell - 1)} \sum_{i=1}^{\ell} |\widehat{\text{tr}}_i - \widehat{\text{tr}}|^2 \quad \text{where } \widehat{\text{tr}} = \frac{1}{\ell} \sum_{i=1}^{\ell} \widehat{\text{tr}}_i. \tag{15.13}$$

We have the following result:

Proposition 15.4 (XTRACE and XNYSTRACE: Posterior error estimate). *For both algorithms, the posterior error estimate* (15.13) *satisfies*

$$\mathbb{E}\left[\widehat{\mathrm{Err}}^2\right] = \frac{1 - \mathrm{Cor}(\widehat{\mathrm{tr}}_1, \widehat{\mathrm{tr}}_2)}{1 + (\ell - 1)\,\mathrm{Cor}(\widehat{\mathrm{tr}}_1, \widehat{\mathrm{tr}}_2)} \cdot \mathbb{E}\left(\mathrm{tr}(\boldsymbol{B}) - \widehat{\mathrm{tr}}\right)^2,$$

where Cor *is the correlation. Moreover, if* $\boldsymbol{B}$ *is a* real *symmetric matrix, the columns* $\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_\ell$ *are iid standard Gaussian vectors, and the algorithm is XTRACE, then*

$$\mathrm{Cor}(\widehat{\mathrm{tr}}_1, \widehat{\mathrm{tr}}_2) \le 2\frac{\mathbb{E}\|(\boldsymbol{I} - \boldsymbol{Q}_{(12)}\boldsymbol{Q}_{(12)}^*)\boldsymbol{B}(\boldsymbol{I} - \boldsymbol{Q}_{(12)}\boldsymbol{Q}_{(12)}^*)\|^2}{\mathbb{E}\|(\boldsymbol{I} - \boldsymbol{Q}_{(1)}\boldsymbol{Q}_{(1)}^*)\boldsymbol{B}(\boldsymbol{I} - \boldsymbol{Q}_{(1)}\boldsymbol{Q}_{(1)}^*)\|_{\mathrm{F}}^2}. \tag{15.14}$$

Here, $\boldsymbol{Q}_{(1)}$ *and* $\boldsymbol{Q}_{(12)}$ *were defined in Lemma 15.3.*

See [ETW24, §5.6] for a proof.

In practice, we find that the XTRACE and XNYSTRACE estimates have a small positive correlation. For example, [ETW24, §5.6] reports empirical correlations of at most 0.06 on the `exp` example (14.27c). In this typical setting, Proposition 15.4 shows that $\widehat{\mathrm{Err}}$ provides a slight *underestimate* of the error, on average.

The second result of Proposition 15.4 provides some control on the size of these correlations in the case where $\boldsymbol{B}$ is real and symmetric and the test vectors are Gaussian. In particular, if the matrix has a *slow* rate of spectral decay, then the Frobenius norm in the denominator of (15.14) denominates the spectral norm in the numerator, so the correlations are small.

Figure 15.2 compares the actual error to the error estimate for the XTRACE algorithm applied to the four test matrices (14.27). Across each test matrix for most numbers of matvecs s, the error estimate closely tracks the true error, agreeing up to a factor of 1.5×. There are two situations where the error estimate performs somewhat less effectively: The first is at the value of $s = 100$ matvecs for the step example. This example confounds the error estimator because there is a large gap between the eigenvalues $\lambda_{s/2}(\boldsymbol{A})$ and $\lambda_{s/2+1}(\boldsymbol{A})$, causing the basic XTRACE estimators to become more correlated, underestimating the error by a factor of about 0.3. The second example occurs on the `exp` example for large values of s. Here, the trace estimates are so accurate that the size of rounding errors comes into play, and the error estimate is less quantitatively sharp. This is a non-issue in practice (in double precision computation, at least), because the error estimate correctly diagnoses that the trace has been computed up to "machine accuracy".
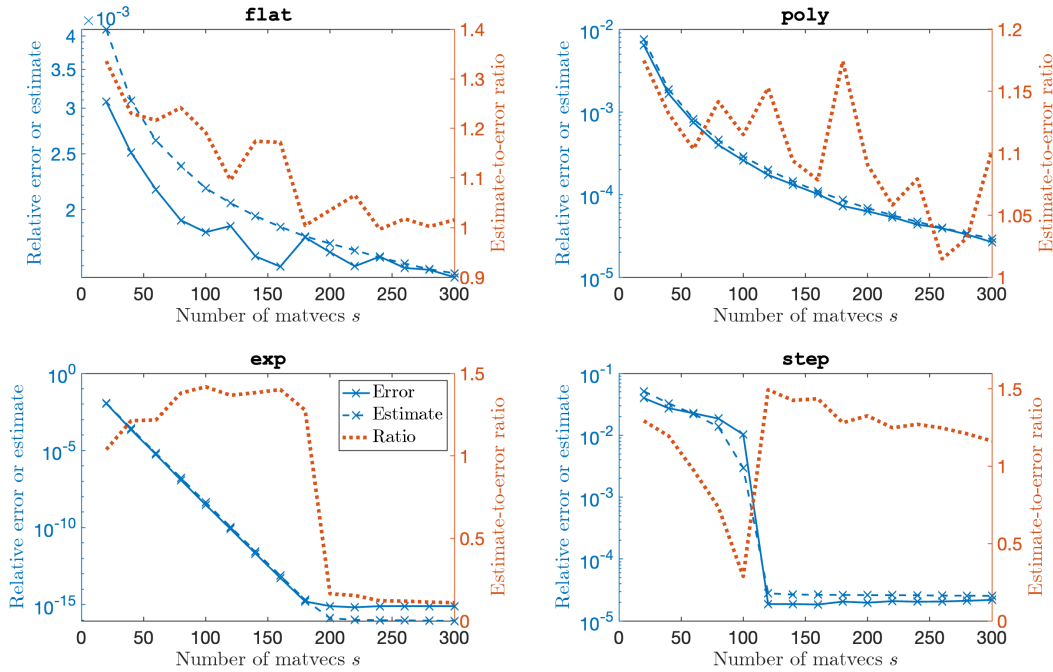
Figure 15.2: Left-hand axes on each subplot show the error (blue solid crosses) and the median error estimate (blue dashed crosses) for XTRACE on the four test matrices (14.27). Right-hand axes show the ratio of the median error estimate to the median error (red dotted). Medians are computed using 1000 trials.

15.4 Adaptivity

The XTRACE or XNYSTRACE algorithms can be implemented to adaptively determine the number of matvecs required to achieve an error tolerance. The basic scheme is simple. Begin by running the XTRACE or XNYSTRACE algorithm with some initial number of matvecs s and compute the error estimate (15.13). If the error estimate is below the tolerance, output the estimated trace. Otherwise, collect an additional Δs matvecs, set $s \leftarrow s + \Delta s$, and update estimates with the newly collected vectors, repeating as many times as necessary to achieve the error tolerance.

A basic version of this scheme with $\Delta s \coloneqq s$ was proposed in [ETW24, §3.2]. With this approach, the number of matvecs is doubled every time the error tolerance is met. This approach can be wasteful, as it may collect roughly twice as many matvecs as needed. On the other hand, it maintains the postprocessing cost of $O(s^2 n)$ of the original algorithm, since

$$s^2 n + (s/2)^2 n + (s/4)^2 n + \cdots = O(s^2 n).$$

A more economical approach is to use a constant increment Δs, say, $\Delta s = 10$. Implemented directly, this approach requires a post-processing cost of $O(s^3 n)$ operations,

since

$$s^2 n + (s - 10)^2 n + (s - 20)^2 n + \cdots = O(s^3 n).$$

However, the cost of the algorithm can be reduced to $O(s^2 n)$ by carefully updating the matrix factorizations as new matvecs are collected. In particular, XTRACE requires an incremental version of the QR decomposition, discussed in Appendix A. An efficient adaptive version of XTRACE (using the `scipy` routine `qr_update` for the incremental QR functionality) was developed by Matt Piekenbrock and is implemented in the `primate` software [Pie24].

15.5 Alternatives to XTRACE and XNYSTRACE

XTRACE and XNYSTRACE are very effective trace estimators for *truly black-box trace estimation* of matrices enjoying *rapidly decaying singular values*. However, in many applications, we have additional structure or lack decaying singular values. In such cases, there may be more effective trace estimators, which we briefly review in this section.

Adaptive HUTCH++

The adaptive HUTCH++ algorithm was introduced by Persson, Cortinovis, and Kressner [PCK22] as a way of both solving the *apportionment* problem of HUTCH++ (how should matvecs be allocated between low-rank approximation and residual trace estimation?) and determining when to stop the algorithm. The core of their idea is that the number of matvecs needed to perform trace estimation on the residual matrix $(\mathbf{I} - \mathbf{Q}\mathbf{Q}^*)\mathbf{A}(\mathbf{I} - \mathbf{Q}\mathbf{Q}^*)$ is determined by the Frobenius norm of that matrix (in view of Fact 13.2 and related results). Thus, the number of additional residual trace estimation matvecs can be estimated at runtime using a randomized norm estimator.

The adaptive HUTCH++ algorithm gives a natural strategy for improving the HUTCH++ algorithm, and it is more straightforward to implement than adaptive versions of XTRACE. However, in experiments reported in [ETW24], we find that adaptive HUTCH++ produces less accurate results than XTRACE does for a fixed budget of matvecs, with a few exceptions.

Stochastic Lanczos quadrature

Consider the task of computing $\operatorname{tr} f(\mathbf{A})$ for a Hermitian matrix $\mathbf{A}$. The starting point for stochastic Lanczos quadrature is the observation that the trace is an integral

$$\operatorname{tr}(f(\mathbf{A})) = \int_{\mathbb{R}} f(x) \, \mathrm{d}\mu_A(x),$$

where $\mu_A = \sum_{i=1}^n \delta_{\lambda_i(A)}$ denotes the (unnormalized) *spectral measure* of A. Thus, computing trace-functions of A can be reformulated as a quadrature problem to approximate integral with respect to the measure μ_A.

Our first move to solve this problem will be to introduce a stochastic approximation for μ_A. For a vector $\omega \in \mathbb{K}^n$, we define the *eigenvector spectral measure* of A and ω to be

$$\mu_{A,\omega} = \sum_{i=1}^n |u_i^* \omega|^2 \, \delta_{\lambda_i(A)} \quad \text{where } A = \sum_{i=1}^n \lambda_i(A) u_i u_i^* \text{ is a spectral decomposition.}$$

The eigenvector spectral measure of A and ω has the property that

$$\omega^* f(A)\omega = \int_{\mathbb{R}} f(x) \, \mathrm{d}\mu_{A,\omega}(x).$$

Stochastic Lanczos quadrature is based on the following observation: If ω is an *isotropic* random vector, then the random measure $\mu_{A,\omega}$ is an unbiased approximation to the spectral measure μ_A, $\mathbb{E}[\mu_{A,\omega}] = \mu_A$. (In general, defining the expectation of a random measure requires care; no issues arise here because the measures $\mu_{A,\omega}$ and μ_A are both supported on the spectrum of A, a finite set.) Thus, given approximate eigenvector spectral measures $\widehat{\mu}_{A,\omega_i} \approx \mu_{A,\omega_i}$ for iid isotropic vectors $\omega_1, \ldots, \omega_k$, we can fashion a Monte Carlo approximation to the spectral measure μ_A by averaging, $\mu_A \approx k^{-1} \sum_{i=1}^k \mu_{A,\omega_i}$.

To approximate the eigenvector spectral measure $\mu_{A,\omega}$ of the pair A and ω, we can deploy the Lanczos algorithm [Lan50; Che24]. Given an initial vector ω, the Lanczos algorithm computes an orthonormal basis $Q \in \mathbb{K}^{n \times t}$ for the *Krylov subspace*

$$\mathsf{K}_t(A, \omega) := \mathrm{span}\{\omega, A\omega, A^2\omega, \ldots, A^{t-1}\omega\}$$

and a Hermitian tridiagonal matrix T such that

$$Q^* A Q = T,$$

where $Q(:, 1) = \omega / \|\omega\|$. We have the following amazing fact:

Fact 15.5 (Lanczos and quadrature). *Introduce* $\widehat{\mu}_{A,\omega} := \|\omega\|^2 \cdot \mu_{T,\mathbf{e}_1}$, *where T is the tridiagonal matrix produced by t steps of the Lanczos procedure. Then $\widehat{\mu}_{A,\omega}$ is the unique t-point Gaussian quadrature measure for the eigenvector spectral measure $\mu_{A,\omega}$. That is,*

$$\omega^* p(A)\omega = \int_{\mathbb{R}} p(x) \, \mathrm{d}\mu_{A,\omega}(x) = \int_{\mathbb{R}} p(x) \, \mathrm{d}\widehat{\mu}_{A,\omega}(x)$$

for all polynomial p of degree $< 2t$.

See [Che24, Ch. 2–3], [Tro20, §3], and [GM10] for more details on the Lanczos method and the proof of this result.

Stochastic Lanczos quadrature consists of two levels of approximation

$$\mu_A \approx \frac{1}{k} \sum_{i=1}^{k} \mu_{A,\omega_i} \approx \widehat{\mu}_A := \frac{1}{k} \sum_{i=1}^{k} \mu_{T_i, \mathbf{e}_1}.$$

First, we approximate the spectral measure μ_A by an average of eigenvector spectral measures μ_{A,ω_i} generated by iid isotropic vectors ω_i. Second, we approximate each μ_{A,ω_i} by a quadrature approximation $\mu_{T_i, \mathbf{e}_1}$ generated from the tridiagonal matrix T_i obtained from applying the Lanczos method to each ω_i. Once formed, the *stochastic Lanczos quadrature* approximation

$$\widehat{\mu}_A := \frac{1}{k} \sum_{i=1}^{k} \mu_{T_i, \mathbf{e}_1} = \sum_{i=1}^{k} \sum_{j=1}^{t} \frac{|\mathbf{u}_{i,j}^* \omega_i|^2}{k} \delta_{\lambda_j(T_i)} \approx \mu_A$$

can be use to compute the trace of *any* trace-function of A at a minimal additional cost of $O(k \cdot t)$ operations:

$$\mathrm{tr}(f(A)) \approx \int_{\mathbb{R}} f(x) \, \mathrm{d}\widehat{\mu}_A(x) = \sum_{i=1}^{k} \sum_{j=1}^{t} \frac{|\mathbf{u}_{i,j}^* \omega_i|^2}{k} f(\lambda_j(T_i)).$$

Here, $T_i = \sum_{j=1}^{t} \lambda_j(T_i) \mathbf{u}_{i,j} \mathbf{u}_{i,j}^*$ denotes a spectral decomposition of each T_i.

To refine the stochastic Lanczos quadrature approximation, one must increase both the number of isotropic vectors k and the number of Lanczos steps t. The rate of convergence is Monte Carlo in k, and the rate of convergence in t depends on the regularity of the function f. For analytic functions, the convergence is geometric [CTU25, §3.2].

Stochastic Lanczos quadrature has advantages and disadvantages for computing $\mathrm{tr}(f(A))$ over variance-reduced trace estimators like Hutch++ and XTrace. As a disadvantage, the (root-mean-square) error of stochastic Lanczos quadrature converges at the un-accelerated Monte Carlo rate $O(k^{-1/2})$, the same as the ordinary non-variance-reduced Girard–Hutchinson estimator. For problems with decaying eigenvalues, Hutch++ and XTrace can converge must faster. As an advantage, once the stochastic Lanczos quadrature rule has been computed, one can estimate $\mathrm{tr}(f(A))$ for as many different functions f as desired. This feature contrasts with Hutch++ and XTrace, which must be applied separately for each function f.

Krylov-aware stochastic trace estimation

The virtues of Hutch++ and stochastic Lanczos quadrature are combined in the *Krylov-aware trace estimator* of Chen and Hallman [CH23]. As with stochastic Lanczos quadrature, their algorithm applies to functions $f(A)$ of a *Hermitian* matrix A. Fix a block size b, number of Lanczos steps t, rank k, and number of Girard–Hutchinson steps m. The Krylov-aware trace estimator begins by applying (roughly) $\lceil k/b \rceil + t$ *steps* of the *block* Lanczos method [GU77] to a matrix $\boldsymbol{\Omega}$ with iid standard Gaussian columns to build an approximation

$$f(A) \approx Q f(T) Q^*.$$

This approximation is then truncated to rank k by symmetrically projecting onto the matrix $\boldsymbol{Q}_k := \boldsymbol{Q}(:, 1:k)$ containing the first k columns of $\boldsymbol{Q}$:

$$f(A) \approx M := Q_k Q_k^* \cdot Q f(T) Q^* \cdot Q_k Q_k^* = Q_k [f(T)](1:k, 1:k) Q_k^*.$$

Then, they estimate the residual trace $\mathrm{tr}(f(A) - M)$ using the resphered Girard–Hutchinson estimator, resulting in the estimate

$$\widehat{\mathrm{tr}}_{\mathrm{KA}} = \mathrm{tr}(M) + \frac{n-k}{m} \sum_{i=1}^{m} \frac{\boldsymbol{v}_i^*(f(A)\boldsymbol{v}_i)}{\boldsymbol{v}_i^* \boldsymbol{v}_i} \text{ for } \boldsymbol{v}_i = (\mathbf{I} - \boldsymbol{Q}_k \boldsymbol{Q}_k^*)\boldsymbol{\gamma}_i, \ \boldsymbol{\gamma}_i \overset{\text{iid}}{\sim} \mathrm{NORMAL}_{\mathbb{R}}(\mathbf{0}, \mathbf{I}).$$

Matvecs $f(A)\boldsymbol{v}_i$ in this display are performed using t steps of the Lanczos algorithm. See [CH23, §3] for details and [PCM25] for analysis of the approximation M.

The Krylov-aware trace estimator and XTrace accelerate the Hutch++ method in different ways. The Krylov-aware estimator consolidates the computation of matrix–vector products $f(A)\boldsymbol{\Omega}$ and $f(A)\boldsymbol{Q}$ into a single invocation of the block Lanczos algorithm, but retains the need to perform matrix–vector products $f(A)\boldsymbol{v}_i$ for residual trace estimation. XTrace still needs two batches $f(A)\boldsymbol{\Omega}$ and $f(A)\boldsymbol{Q}$ of matvecs to form a low-rank approximation of $f(A)$ but removes the need for additional matvecs for residual trace estimation.

The main advantage of the Krylov-aware trace estimator over XTrace and XNys-Trace is that it can be used to evaluate $\mathrm{tr}(f(A))$ for a family of functions f at minimal additional effort. For this reason, the Krylov-aware trace estimator is often the best algorithm for the $\mathrm{tr}(f(A))$ problem.

Probing methods

Another family of methods for trace estimation are based on *probing*. Here, is one version of the idea. We know that the Girard–Hutchinson estimator with random

sign vectors has a variance depending only on the sum of the off-diagonal entries:

$$\mathrm{Var}(\omega^* A \omega) = 2 \sum_{i \neq j} a_{ij}^2 \quad \text{for } A \in \mathbb{R}^{n \times n} \text{ symmetric and } \omega \sim \mathrm{U}_{\mathrm{NIF}}\{\pm 1\}^n.$$

Let us use this fact to our advantage. Suppose that we are able to partition the index set $\{1, \ldots, n\}$ as a disjoint union of a small number of sets $\mathsf{S}_1, \ldots, \mathsf{S}_m$ with the property that the off-diagonal entries of each submatrix $A(\mathsf{S}_i, \mathsf{S}_i)$ are small. Using such an approximation, we can obtain an accurate approximation to the trace by applying the Girard–Hutchinson estimator separately to each submatrix $A(\mathsf{S}_i, \mathsf{S}_i)$ where the random vectors ω are supported on each partition S_i. In particular, we can define the *stochastic probing estimator*:

$$\widehat{\mathrm{tr}}_{\mathrm{SP}} := \frac{1}{t} \sum_{i=1}^{m} \sum_{j=1}^{t} \omega_{ij}^* A \omega_{ij} \quad \text{with} \quad \begin{cases} \omega_{i1}(\mathsf{S}_i), \ldots, \omega_{it}(\mathsf{S}_i) \overset{\text{iid}}{\sim} \mathrm{U}_{\mathrm{NIF}}\{\pm 1\}^{\mathsf{S}_i}, \\ \omega_{i1}(\overline{\mathsf{S}_i}) = \cdots = \omega_{it}(\overline{\mathsf{S}_i}) = \mathbf{0}. \end{cases}$$

Here, t denotes the number of matvecs per set S_i, and $\overline{\mathsf{S}_i} := \{1, \ldots, n\} \setminus \mathsf{S}_i$ denotes the complement of each S_i.

Stochastic probing methods were initially developed in quantum chromodynamics [MBFJ+11; BBCF+12], and they reached computational mathematics shortly thereafter [ASE14; GSO17]. There are also deterministic versions of probing [TS12]. See the paper [FRS25] for analysis of stochastic probing methods and a literature review on probing.

Probing methods require a good partition $\mathsf{S}_1, \ldots, \mathsf{S}_m$, which demands prior information about the matrix A. In a typical application of probing, one is interested in evaluating the trace (or diagonal) of $A = f(H)$ for a sparse matrix H. In this setting, one can construct a partitioning $\mathsf{S}_1, \ldots, \mathsf{S}_m$ from a (distance-d) graph coloring of the adjacency graph of H, with the hope that $f(H)$ possesses decaying entries away from the sparsity pattern of H.

Access to a good partitioning is a nontrivial requirement for probing methods. However, when such an a partitioning is known, probing methods can be very effective [FRS25, Fig. 3].

C h a p t e r   16

DIAGONAL ESTIMATION

Extracting [the diagonal of the inverse of A] is considered a challenging task, in part because the problem cannot be easily expressed in the form of a system of equations. The problem is usually harder to solve than a linear system with the same matrix A. So far, not much literature has been devoted to this topic, whereas the related problem of estimating the trace of B [= A^{-1}] received much more attention.

Jok M. Tang and Yousef Saad, *A probing method for computing the diagonal of a matrix inverse* [TS12]

Chapter 14 presented the basic leave-one-out methodology and applied it to the problem of trace estimation. This section will use the same approach to develop diagonal estimators, XDɪᴀɢ and XNʏsDɪᴀɢ, for general and psd matrices. We will also encounter another approach to matrix attribute estimation based on unbiased low-rank approximation.

Sources. The XDɪᴀɢ estimator was developed in the XTʀᴀᴄᴇ paper:

Ethan N. Epperly, Joel A. Tropp, and Robert J. Webber. "XTrace: Making the Most of Every Sample in Stochastic Trace Estimation". In: *SIAM Journal on Matrix Analysis and Applications* (Mar. 2024), pp. 1–23. ᴅᴏɪ: 10.1137/23M1548323.

The XNʏsDɪᴀɢ algorithm and the unbiased randomized SVD algorithms are newly developed in this thesis.

Outline. Section 16.1 introduces existing approaches to diagonal estimation not using the leave-one-out approach, Section 16.2 presents the XDɪᴀɢ estimator for the diagonal of a general matrix, Section 16.3 develops the XNʏsDɪᴀɢ estimator for the diagonal of a psd matrix, and Sections 16.4 and 16.5 provides an experimental comparison. We conclude with an alternative approach to matrix attribute estimation based on *unbiased low-rank approximation* in Section 16.6. The XDɪᴀɢ estimator can also be derived using this approach.

16.1 The BKS and Diag++ diagonal estimators

To begin this chapter, we briefly discuss existing approaches to the diagonal estimation problem and propose a few modifications. Let $\boldsymbol{B} \in \mathbb{K}^{n \times n}$ be a general square matrix.

In Section 13.1, we reviewed the BKS diagonal estimator (13.4) for $\mathbf{diag}(\boldsymbol{B})$. Given iid isotropic vectors $\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_s$, we defined

$$\widehat{\mathbf{diag}}_{\mathrm{BKS}} = \frac{1}{s} \sum_{i=1}^{s} (\boldsymbol{B}\boldsymbol{\omega}_i) \odot \overline{\boldsymbol{\omega}_i}. \tag{16.1}$$

The BKS estimator is an unbiased Monte Carlo estimator for $\mathbf{diag}(\boldsymbol{B})$. Given its similarity to the Girard–Hutchinson trace estimator, this estimator is also sometimes called *Hutchinson's diagonal estimator*. Analysis appears in [BN22; DM23a].

Remark 16.1 (Original BKS estimator). The original work [BKS07] of Bekas, Kokiopoulou, and Saad treated the real case $\boldsymbol{\omega}_i \in \mathbb{R}^n$ and considered the estimator

$$\widehat{\mathbf{diag}}_{\mathrm{BKS,orig}} = \frac{\sum_{i=1}^{s} (\boldsymbol{B}\boldsymbol{\omega}_i) \odot \boldsymbol{\omega}_i}{\sum_{i=1}^{s} \boldsymbol{\omega}_i \odot \boldsymbol{\omega}_i}. \tag{16.2}$$

Division is performed entrywise. The original BKS estimator (16.2) and simplified BKS estimator (16.1) coincide when the vectors $\boldsymbol{\omega}_i$ are drawn iid from the random sign distribution $\mathrm{U}_{\mathrm{NIF}}\{\pm 1\}^n$. This thesis will work exclusively with the simplified BKS estimator $\widehat{\mathbf{diag}}_{\mathrm{BKS}}$ given in (16.1). $\diamond$

The Hutch++ variance reduction framework extends naturally to the diagonal estimation problem. Given a budget of s matvecs, begin by computing the randomized SVD approximation of rank $s/3$. In detail, generate a test matrix $\boldsymbol{\Omega} \in \mathbb{K}^{n \times (s/3)}$ with isotropic random columns and form $\boldsymbol{Q} = \mathbf{Orth}(\boldsymbol{B}\boldsymbol{\Omega})$. As usual, $\boldsymbol{Q}$ defines a low-rank approximation $\widehat{\boldsymbol{B}} = \boldsymbol{Q}(\boldsymbol{Q}^*\boldsymbol{B})$. Now, we decompose $\mathbf{diag}(\boldsymbol{B})$ as

$$\mathbf{diag}(\boldsymbol{B}) = \mathbf{diag}(\widehat{\boldsymbol{B}}) + \mathbf{diag}(\boldsymbol{B} - \widehat{\boldsymbol{B}}),$$

and apply the BKS estimator to estimate the residual diagonal. The resulting *unbiased Diag++ estimator* is

$$\widehat{\mathbf{diag}}_{\mathrm{UD++}} := \mathbf{diag}(\boldsymbol{Q}(\boldsymbol{Q}^*\boldsymbol{B})) + \frac{1}{s/3} \sum_{i=1}^{s/3} [(\mathbf{I} - \boldsymbol{Q}\boldsymbol{Q}^*)\boldsymbol{B}\boldsymbol{\gamma}_i] \odot \overline{\boldsymbol{\gamma}_i}. \tag{16.3}$$

The vectors $\boldsymbol{\gamma}_1, \ldots, \boldsymbol{\gamma}_{s/3}$ are chosen to be isotropic and iid. The computational cost of the unbiased Diag++ estimator is $2s/3$ matvecs with $\boldsymbol{B}$, $s/3$ matvecs with $\boldsymbol{B}^*$, and $O(s^2 n)$ additional arithmetic operations. See Program 16.1 for code.

Program 16.1 `udiagpp.m`. Unbiased DIAG++ estimator for the diagonal of a matrix. Subroutine `random_signs` is provided in Program F.2.

```
function d = udiagpp(B,Bt,n,s)
% Input:  Functions B() and Bt() computing matrix products B(X) =
%           B*X and Bt(X) = B'*X, number of rows n, and number of
%           matvecs s
% Output: Estimate d of the diagonal of B

% Randomized SVD
k = floor(s/3);         % Approximation rank is s/3
Om = random_signs(n,k); % Matrix of random signs
BOm = B(Om);            % Matvecs with B
[Q,~] = qr(BOm,"econ"); % Randomized SVD

% Unbiased Diag++ estimator
BtQ = Bt(Q);            % Matvecs with B'
k = s - 2*k;            % Remaining matvecs
Ga = random_signs(n,k); % More random signs
BGa = B(Ga);            % Matvecs with B
d = diagprod(Q',BtQ') + mean((BGa - Q*(Q'*BGa)) .* Ga,2);

end
```

In the original paper on variance-reduced diagonal estimation [BN22], Baston and Nakatsukasa use the following alternative definition of the DIAG++ estimator:

$$\widehat{\mathbf{diag}}_{\mathrm{D++}} := \mathbf{diag}(QQ^*(BQ)Q^*) + \frac{1}{s/3}\sum_{i=1}^{s/3}[(I-QQ^*)B(I-QQ^*)\omega_i]\odot\overline{\omega_i}. \quad (16.4)$$

This estimator requires only matvecs with B and not its adjoint B^*, an advantage over the unbiased DIAG++ estimator. In general, the original DIAG++ estimator is biased, although this bias may be small relative to the diagonal if the matrix B has rapid singular value decay. To distinguish between the two estimators (16.3) and (16.4), "the DIAG++ estimator" will always refer to the original DIAG++ estimator (16.4).

16.2 XDIAG: A leave-one-out diagonal estimator for general matrices

The idea of XDIAG is similar to XTRACE. We proceed with a derivation.

Step 1: Compute a low-rank approximation to the input matrix by multiplying it against a collection of test vectors. We begin by computing a low-rank approximation to B using the randomized SVD. Let $\Omega \in \mathbb{K}^{n\times s/2}$ be a random matrix with isotropic columns, form the product $B\Omega$, and orthonormalize $Q = \mathbf{Orth}(B\Omega)$. We

obtain the approximation

$$\widehat{B} := QQ^*B.$$

Step 2: Decompose the quantity of interest into known piece depending on the low-rank approximation plus a residual. The diagonal operation is linear. As such,

$$\mathbf{diag}(B) = \mathbf{diag}(\widehat{B}) + \mathbf{diag}(B - \widehat{B}) = \mathbf{diag}(QQ^*B) + \mathbf{diag}((I - QQ^*)B).$$

Step 3: Construct a Monte Carlo estimate of the residual using a single random vector. To approximate the residual diagonal $\mathbf{diag}((I - QQ^*)B)$, we employ the single-vector BKS estimator:

$$\mathbf{diag}((I - QQ^*)B) \approx (I - QQ^*)B\omega \odot \overline{\omega}.$$

This step leads to the unbiased diagonal estimate

$$\widehat{\mathbf{diag}} := \mathbf{diag}(QQ^*B) + (I - QQ^*)B\omega \odot \overline{\omega}.$$

Step 4: Downdate the low-rank approximation by recomputing it with a test vector removed, and use the left-out test vector as the random vector in step 3. Now, rather than using a fresh vector ω, we downdate the randomized SVD, obtaining a new matrix $Q_{(i)} = \mathbf{Orth}(B\Omega_{-i})$ that is independent from the ith test vector ω_i. This process defines a family of basic XDɪᴀɢ estimators

$$\widehat{\mathbf{diag}}_i := \mathbf{diag}(Q_{(i)}Q_{(i)}^*B) + (I - Q_{(i)}Q_{(i)}^*)B\omega_i \odot \overline{\omega_i} \quad \text{for } i = 1, \dots, s/2.$$

Step 5: Average over all choices of vectors to leave out. Finally, we average over the index i to obtain the full XDɪᴀɢ estimate

$$\widehat{\mathbf{diag}}_{\mathrm{X}} := \frac{1}{s/2} \sum_{i=1}^{s/2} \left[\mathbf{diag}(Q_{(i)}Q_{(i)}^*B) + (I - Q_{(i)}Q_{(i)}^*)B\omega_i \odot \overline{\omega_i} \right]. \tag{16.5}$$

Efficient formula and implementation. To implement XDɪᴀɢ efficiently, we employ the randomized SVD downdating formula (14.5). In particular, we form $Y = B\Omega$, compute the QR decomposition $Y = QR$, and obtain the downdating matrix S by normalizing the columns of R^{-*}. The basic XDɪᴀɢ estimator is then

$$\widehat{\mathbf{diag}}_i = \mathbf{diag}(Q(I - s_i s_i^*)Q^*B) + (I - QQ^* + Qs_i s_i^*Q^*)B\omega_i \odot \overline{\omega_i}.$$

We can simplify this equation is several ways. First, define and compute

$$Z := B^*Q.$$

Program 16.2 `xdiag.m`. Efficient implementation of XDɪᴀɢ. Subroutines `cnormc` and `diagprod` are provided in Programs 14.1 and F.1.

```matlab
function d = xdiag(B,Bt,n,s)
% Input:  Functions B() and Bt() computing matrix products B(X) =
%         B*X and Bt(X) = B'*X, number of rows n, and number of
%         matvecs s
% Output: Estimate d of the diagonal of B

% Randomized SVD
k = floor(s/2);          % Approximation rank is s/2
Om = random_signs(n,k);  % Matrix of random signs
Y = B(Om);               % Matvecs with B
[Q,R] = qr(Y,"econ");    % Randomized SVD
S = cnormc(inv(R'));     % Downdate RSVD, cnormc normalizes cols

% Form estimator
Z = Bt(Q);               % Matvecs with B'
d = diagprod(Q',Z') + mean((Q*S)...                   % XDiag
    .* (-conj(Z*S) + conj(Om) .* diagprod(S,R).'),2);

end
```

Second, observe that $\mathbf{I} - \boldsymbol{Q}\boldsymbol{Q}^*$ annihilates $\boldsymbol{B}\omega_i = \boldsymbol{y}_i$ and $\boldsymbol{Q}^*\boldsymbol{B}\omega_i = \boldsymbol{Q}^*\boldsymbol{y}_i = \boldsymbol{r}_i$. Introducing $\boldsymbol{Z}$ and these observations, the basic XDɪᴀɢ estimators simplify as

$$\widehat{\mathbf{diag}}_i = \mathbf{diagprod}(\boldsymbol{Q}^*, \boldsymbol{Z}^*) + \boldsymbol{Q}\boldsymbol{s}_i \odot (-\overline{\boldsymbol{Z}\boldsymbol{s}_i} + \boldsymbol{s}_i^*\boldsymbol{r}_i \cdot \overline{\omega}_i).$$

Averaging, we obtain a formula for the full XDɪᴀɢ estimator.

$$\widehat{\mathbf{diag}}_X = \mathbf{diagprod}(\boldsymbol{Q}^*, \boldsymbol{Z}^*) + \frac{1}{s/2}[\boldsymbol{Q}\boldsymbol{S} \odot (-\overline{\boldsymbol{Z}\boldsymbol{S}} + \overline{\boldsymbol{\Omega}} \cdot \mathbf{Diag}(\mathbf{diagprod}(\boldsymbol{S}, \boldsymbol{R})))]\boldsymbol{1}.$$

Code is provided in Program 16.2.

Remark 16.2 (Original XDɪᴀɢ estimator). In our original paper [ETW24], my coauthors and I used the original BKS estimator (16.2) for residual trace estimation in XDɪᴀɢ. However, all of the experiments used the random sign distribution and were thus insensitive to the choice between the original and unbiased BKS estimators. I now prefer the version of XDɪᴀɢ presented in this thesis, because it is unbiased for any choice of isotropic test vectors. The publicly released code for the paper [ETW24]—available at `https://github.com/eepperly/XTrace`—implements the same version of XDɪᴀɢ that appears in this thesis. ◇

16.3   XN$\textsc{ys}$D$\textsc{iag}$: A leave-one-out diagonal estimator for psd matrices

We can develop an improved estimator XN$\textsc{ys}$D$\textsc{iag}$ for the diagonal of a psd matrix by using Nyström approximation in place of the randomized SVD. The relationship between XN$\textsc{ys}$D$\textsc{iag}$ and XD$\textsc{iag}$ is analogous to the relationship between XN$\textsc{ys}$T$\textsc{race}$ and XT$\textsc{race}$.

Having presented many detailed algorithm derivations in the leave-one-out framework, we now proceed more quickly. Let $A \in \mathbb{K}^{n \times n}$ be a psd matrix. Begin by drawing a matrix $\boldsymbol{\Omega} \in \mathbb{K}^{n \times s}$ with iid isotropic columns, and define the Nyström approximation

$$\widehat{A} := A\langle \boldsymbol{\Omega} \rangle.$$

Leaving out each column of $\boldsymbol{\Omega}$ in turn gives downdated approximations

$$\widehat{A}_{(i)} := A\langle \boldsymbol{\Omega}_{-i} \rangle \quad \text{for } i = 1, 2, \ldots, s.$$

For each i, the diagonal may be decomposed as

$$\mathbf{diag}(A) = \mathbf{diag}(\widehat{A}_{(i)}) + \mathbf{diag}(A - \widehat{A}_{(i)}).$$

Estimating the second term using the single-vector BKS diagonal estimator yields the basic XN$\textsc{ys}$D$\textsc{iag}$ estimators

$$\widehat{\mathbf{diag}}_i := \mathbf{diag}(\widehat{A}_{(i)}) + (A - \widehat{A}_{(i)})\omega_i \odot \overline{\omega}_i \quad \text{for } i = 1, 2, \ldots, s.$$

Averaging over the index i yields the full XN$\textsc{ys}$D$\textsc{iag}$ estimate

$$\widehat{\mathbf{diag}}_{\mathrm{XN}} = \frac{1}{s} \sum_{i=1}^{s} \widehat{\mathbf{diag}}_i.$$

Efficient formula and implementation. To implement XN$\textsc{ys}$D$\textsc{iag}$ efficiently, we compute the Nyström approximation $\widehat{A} = FF^*$ using (2.11) and form the matrix Z in (14.25b). The basic XN$\textsc{ys}$D$\textsc{iag}$ estimators are

$$\widehat{\mathbf{diag}}_i = \mathbf{diag}(FF^* - z_i z_i^*) + (A - FF^* + z_i z_i^*)\omega_i \odot \overline{\omega}_i$$

In view of the interpolatory property Proposition 2.9(d), we have the identity $A\omega_i = (FF^*)\omega_i$, so this formula simplifies to

$$\widehat{\mathbf{diag}}_i = \mathbf{srn}(F) - |z_i|^2 + (z_i \odot \overline{\omega}_i) \cdot z_i^* \omega_i.$$

Averaging over the index i yields a formula for the full XN$\textsc{ys}$D$\textsc{iag}$ estimator:

$$\widehat{\mathbf{diag}}_{\mathrm{XN}} = \mathbf{srn}(F) - \frac{1}{s}\mathbf{srn}(Z) + \frac{1}{s}(Z \odot \overline{\boldsymbol{\Omega}}) \cdot \mathbf{Diag}(\mathbf{diagprod}(Z, \boldsymbol{\Omega})) \cdot \mathbf{1}.$$

Program 16.3 `xnysdiag.m`. Efficient and stable implementation of XNysDiag estimator. Subroutine `nystrom` is provided in Program 2.3.

```
function d = xnysdiag(A,n,s)
% Input:  Function A() computing matrix products A(X) = A*X, number
%         of rows n, and number of matvecs s
% Output: Estimate d of diag(A)

[F,mu,Om,R] = nystrom(A,n,s);                    % Nystrom approx
Z = (F/R') .* (sqrownorms(inv(R)) .^ (-1/2))';   % Downdate it
d = sqrownorms(F) + (-sqrownorms(Z) + sum((Z...  % Compute estimator
    .* conj(Om)) .* diagprod(Z,Om).',2))/s ...
    - mu * ones(n,1);

end
```

For stability, we apply XNysDiag to $A + \mu\mathbf{I}$ with the shift from (2.11a). We correct the shift by subtracting $\mu\mathbf{1}$ from the diagonal estimate. Code is provided in Program 16.3.

Remark 16.3 (XNysDiag and XNysTrace). Observe that the XNysTrace estimator is the sum of the entries of the XNysDiag estimator. In contrast, the XTrace estimate does not equal to the sum of the entries of the XDiag estimate. ◇

16.4 Synthetic Experiments

We compare diagonal estimators on two sets of synthetic examples. First, we test with the four synthetic test matrices from (14.27) with varying spectral properties. Second, we test on two matrices with similar spectra but very disparate distributions of diagonal entries.

Tests on matrices with different spectra

Figure 16.1 provides a comparison of the BKS, unbiased Diag++, XDiag, and XNysDiag algorithms on the synthetic matrices from (14.27). To quantify the discrepancy between a diagonal approximation $\widehat{\mathbf{diag}}$ and the true diagonal $\mathbf{diag}$, we use the maximum relative error:

$$\text{maxrelerr} = \max_{1\leq i\leq n} \frac{|\widehat{\text{diag}}_i - \text{diag}_i|}{|\text{diag}_i|}. \tag{16.6}$$

The comparison of diagonal estimators in Fig. 16.1 is similar with the comparison of trace estimators in Fig. 14.1: The XNysDiag and XDiag estimators outperform unbiased XDiag and the BKS estimator on problems with spectral decay (i.e., the
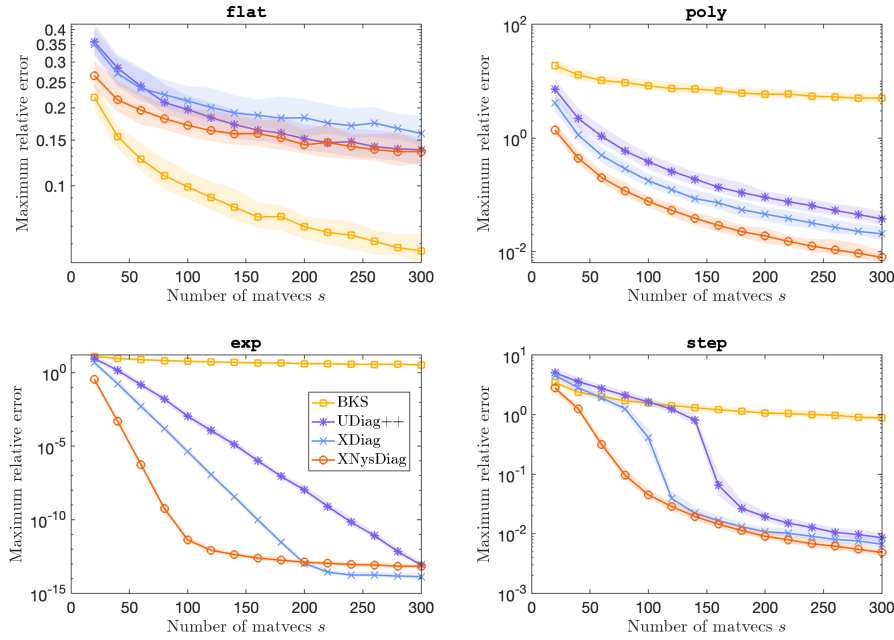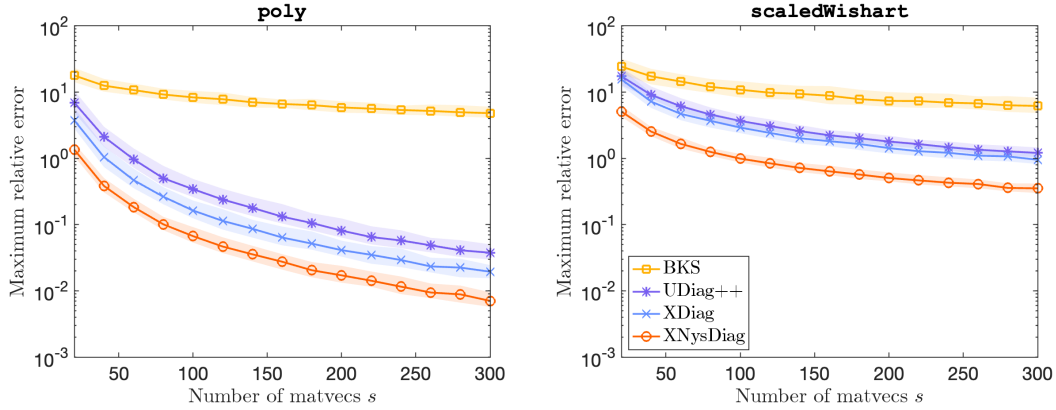
Figure 16.1: Maximum relative error of diagonal estimates by the BKS estimator (yellow squares), unbiased DIAG++ (purple asterisks), XDIAG (blue crosses), and XNYSDIAG (orange circles) for the four test matrices (14.27) as a function of the numbers of matvecs s. The error metric is the maximum relative error, defined in (16.6). Lines show the median of 100 trials, and shaded regions range from the 10% to the 90% quantile.

poly and exp matrices), and the BKS estimator slightly outperforms the other estimators when the spectrum is flat (i.e., the flat example). The step example demonstrates a dissimilarity between the diagonal estimation and trace estimation problems: For diagonal estimation, the XNYSDIAG estimator outperforms XDIAG on this example for any number of matvecs s, whereas XNYSTRACE does worse than XTRACE on this example for sufficiently large values of s.

Tests on matrices with similar spectra but different diagonals

A key distinction between diagonal estimation and trace estimation is that the diagonal entries may vary across many orders of magnitude. In such cases, it is harder to estimate all of the diagonal entries up to a small relative error. To demonstrate this point, we compare the diagonal estimators on two matrices with similar spectra but very different diagonals: the poly matrix (14.27b) defined above and the following scaledWishart matrix

$$\texttt{scaledWishart} := D^{1/2} G^* G D^{-1/2} \tag{16.7}$$

Figure 16.2: Maximum relative error of diagonal estimates by the BKS estimator (yellow squares), unbiased DIAG++ (purple asterisks), XDIAG (blue crosses), and XNYSDIAG (orange circles) on the `poly` matrix ((14.27b), *left*) and `scaledWishart` matrix ((16.7), *right*). The error metric is the maximum relative error, defined in (16.6). Lines show the median of 100 trials, and shaded regions show the 10% and 90% quantiles.

where $G \in \mathbb{R}^{2n \times n}$ is a matrix with iid (real) standard Gaussian entries and $D :=$ $\mathbf{Diag}(i^{-2} : i = 1, \ldots, n)$ is a diagonal matrix with entries decaying at an algebraic rate. We again set $n := 10^3$. These two matrices have the following properties:

- ***Similar eigenvalue decay.*** Both `poly` and `scaledWishart` have polynomially decreasing eigenvalues spanning a similar range: $\lambda_{\max}/\lambda_{\min}$ is 1.0e6 for `poly` and 5.0e6 for `scaledWishart`.

- ***Dustubct diagonal values.*** Despite their spectral similarities, `poly` and `scaledWishart` have very different diagonals. The diagonal entries of `poly` are comparable, with $\max a_{ii}/\min a_{ii} = 6.9\text{e}1$, while the diagonal entries of `scaledWishart` span many orders of magnitude, with $\max a_{ii}/\min a_{ii} = 1.0\text{e}6$.

Results appear in Fig. 16.2. We see that while the *ordinal ranking* of the algorithms remains unchanged (XNYSDIAG > XDIAG > unbiased DIAG++ > BKS), the maximum relative error is significantly higher for the `scaledWishart` example than the `poly` examples. These experiments demonstrate that accurately approximating the small diagonal entries of a matrix whose diagonal entries span multiple orders of magnitude can be challenging for variance-reduced diagonal estimators. This behavior makes sense: Unbiased DIAG++, XDIAG, and XNYSDIAG all work by computing a
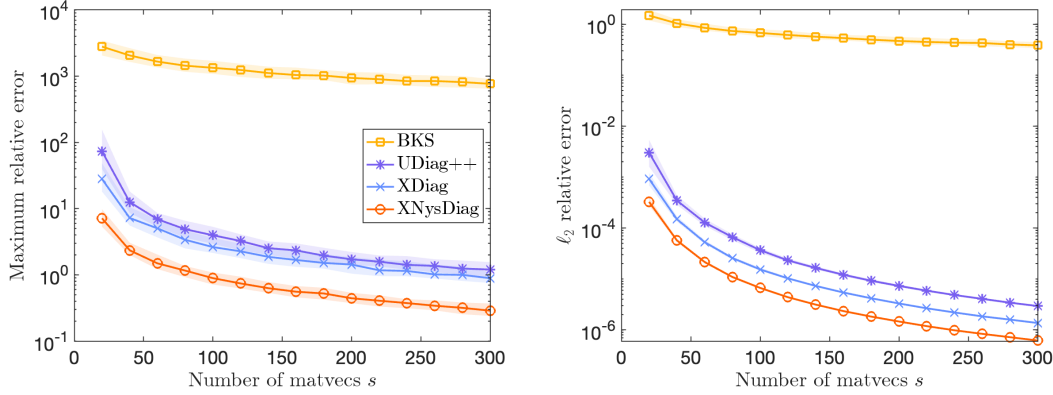
Figure 16.3: Maximum relative error of diagonal estimates by the BKS estimator (yellow squares), unbiased DIAG++ (purple asterisks), XDIAG (blue crosses), and XNYSDIAG (orange circles) for estimating the subgraph centralities. We show both the the maximum relative error (*left*, (16.6)) and the relative ℓ_2 error (*right*). Lines show median of 100 trials, and error bars show 10% and 90% quantiles.

low-rank approximation of the input matrix, which naturally capture large matrix entries better than smaller ones.

16.5 Application: Subgraph centralities

In Section 14.9, we used stochastic trace estimators to compute the Estrada index of a graph with adjacency matrix M, which is defined as

$$\text{estr} := \text{tr}(\exp(M)).$$

The Estrada index serves as a *global* measure of how "centralized" or "clustered" the nodes of a graph are. The *subgraph centralities* [Est22] provide an analogous notion of how "central" each node is to a graph. The vector $\mathbf{sc}$ of subgraph centralities is the diagonal of $A = \exp(M)$:

$$\mathbf{sc} := \mathbf{diag}(\exp(M)).$$

Consequently, the sum of the subgraph centralities is the Estrada index.

Figure 16.3 shows estimates of the the subgraph centralities calculated with different diagonal estimation algorithms. As in Fig. 14.4, we use the `yeast` dataset [BZCX+03] and perform matvecs with $A = \exp(M)$ by forty steps of the Lanczos algorithm [Che24, Ch. 6]. The left panel shows the maximum relative error (16.6) and the right panel shows the ℓ_2 error $\|\widehat{\mathbf{diag}} - \mathbf{diag}\|/\|\mathbf{diag}\|$. We see that XNYS-DIAG clearly outperforms XDIAG and unbiased DIAG++, which in turn outperform the BKS estimator. In absolute terms, the maximum relative error for all methods

is fairly large, with the XNʏsDɪᴀɢ estimator achieving a maximum relative error of 0.3 with $s = 300$ matvecs. The ℓ_2 errors are much smaller for all methods. The reason for the large discrepancy between the maximum relative error and ℓ_2 error is that the diagonal entries of A span many orders of magnitude, similar to the scaled Wishart example (16.7). Thus, the variance-reduced methods compute the largest diagonal entries to high-accuracy but struggle to accurately approximate the smaller entries. We will explore a partial solution to this "small diagonal entries" problem in Sections 17.2 and 17.7.

16.6   Another view: Unbiased low-rank approximation

In this section, we will develop another way of deriving the XDɪᴀɢ estimator. This strategy extends naturally to construct estimators for other matrix entries or linear functionals. So far, we have approached the problem of matrix attribute estimation by designing purpose-built unbiased estimators that target each particular matrix attribute. But one could instead form an unbiased estimate $\widehat{B}$ of the input matrix $B \in \mathbb{K}^{m \times n}$ itself. Given such a $\widehat{B}$, one can obtain unbiased estimates for any *linear* matrix attribute $L(B)$ by computing $L(\widehat{B})$. Here, $L : \mathbb{K}^{m \times n} \to \mathsf{V}$ denotes an arbitrary linear map. Trace estimation and diagonal estimation are special cases of this framework with $L = \mathrm{tr}$ and $L = \mathbf{diag}$. Other special cases include $L(B) = b_{ij}$ (computing a specified entry) or $L(B) = \mathrm{tr}(C^*B)$ (computing the inner product with another matrix C).

We can use the leave-one-out approach to develop unbiased low-rank approximation primitives. We will illustrate by developing an unbiased version of the randomized SVD. Consider the task of computing an unbiased rank-k approximation to a general matrix $B \in \mathbb{K}^{m \times n}$. We follow the leave-one-out approach:

Step 1: Compute a low-rank approximation to the input matrix by multiplying it against a collection of test vectors. Begin by generating a matrix $\Omega \in \mathbb{K}^{n \times k}$ with iid isotropic columns, and form the randomized SVD approximation: $Q = \mathbf{Orth}(B\Omega)$ and $QQ^*B \approx B$.

Step 2: Decompose the quantity of interest into known piece depending on the low-rank approximation plus a residual. The matrix B can decomposed as

$$B = QQ^*B + (I - QQ^*)B.$$

Step 3: Construct a Monte Carlo estimate of the residual using a single random vector. To approximate the residual $(I - QQ^*)B$, introduce a copy of the identity

matrix and introduce the unbiased stochastic approximation $\mathbf{I} \approx \omega\omega^*$ furnished by an isotropic vector $\omega \in \mathbb{K}^n$:

$$(\mathbf{I} - QQ^*)B = (\mathbf{I} - QQ^*)B \cdot \mathbf{I} \approx (\mathbf{I} - QQ^*)B \cdot \omega\omega^*.$$

This results in a stochastic approximation to B,

$$B \approx QQ^*B + (\mathbf{I} - QQ^*)B\omega\omega^*.$$

Step 4: Downdate the low-rank approximation by recomputing it with a test vector removed, and use the left-out test vector as the random vector in step 3. For each i, downdate the randomized SVD approximation to produce a downdated low-rank approximation $Q_{(i)}Q_{(i)}^*B$ that is independent of ω_i. For each i, this procedure yields an unbiased low-rank approximation

$$\widehat{B}_i := Q_{(i)}Q_{(i)}^*B + (\mathbf{I} - Q_{(i)}Q_{(i)}^*)B\omega_i\omega_i^*. \tag{16.8}$$

Step 5: Average over all choices of vectors to leave out. Finally, average (16.8) over the index i to obtain the unbiased randomized SVD low-rank approximation

$$\widehat{B} := \frac{1}{k}\sum_{i=1}^{k}\left[Q_{(i)}Q_{(i)}^*B + (\mathbf{I} - Q_{(i)}Q_{(i)}^*)B\omega_i\omega_i^*\right]. \tag{16.9}$$

Implementation. To develop an efficient unbiased randomized SVD implementation, we substitute the downdating formula (14.5) in (16.9) and simplify algebraically, à la Section 14.3. As usual, the derivation is somewhat complicated; we omit the details. Here is the procedure: First, compute $Y := B\Omega$, the (economy-size) QR decomposition $Y = QR$, the downdating matrix S in (14.5), and the product $C := B^*Q$. The basic unbiased randomized SVD estimators (16.8) are

$$\widehat{B}_i = QC^* + Qs_i(-s_i^*C^* + s_i^*r_i \cdot \omega_i^*), \tag{16.10}$$

and their mean is

$$\widehat{B} = Q\left[\left(\mathbf{I} - \frac{SS^*}{k}\right)C^* + \frac{1}{k} \cdot S \cdot \mathbf{Diag}(\mathbf{diagprod}(S, R)) \cdot \Omega^*\right].$$

An implementation appears in Program 16.4.

Deriving the XDIAG estimator from the unbiased randomized SVD. Comparing the XDIAG estimator (16.5) to the unbiased randomized SVD approximation (16.9),

Program 16.4 `usvd.m`. Unbiased randomized SVD algorithm for unbiased low-rank approximation and matrix attribute estimation. Subroutines `diagprod`, `random_signs`, and `cnormc` are provided in Programs 14.1, F.1 and F.2.

```
function [U,S,V] = usvd(B,Bt,n,k)
% Input:  Functions B() and Bt() computing matrix products B(X) =
%         B*X and Bt(X) = B'*X, number of rows n, and number of
%         matvecs s
% Output: Unbiased low-rank approximation Bhat to B, presented as an
%         economy-size SVD Bhat = U*S*V'

% Define test matrix
Om = random_signs(n,k); % Matrix of random signs
Y = B(Om);              % Matvecs with B
[Q,R] = qr(Y,"econ");   % Randomized SVD
S = cnormc(inv(R'));    % Downdate randomized SVD
C = Bt(Q);              % Matvecs with B'

% Compute unbiased randomized SVD
F = (eye(k) - S*S'/k) * C' + (S .* diagprod(S,R).') * Om' / k;
[UU,S,V] = svd(F,"econ");
U = Q*UU;

end
```

we see the XDɪᴀɢ estimator is the diagonal of the unbiased randomized SVD approximation. In this way, the unbiased randomized SVD algorithm (Program 16.4) gives another way of implementing XDɪᴀɢ: Simply run the unbiased randomized SVD and form XDɪᴀɢ estimator by the formula $\widehat{\mathbf{diag}} = \mathbf{diagprod}(\boldsymbol{U}^*, \boldsymbol{\Sigma V}^*)$. This approach has the advantage that one can reuse the same unbiased randomized SVD approximation to estimate other entries of the matrix on an as-needed basis via the formula

$$b_{ij} \approx \widehat{b}_{ij} = \boldsymbol{U}(i,:)\boldsymbol{\Sigma V}(j,:)^*.$$

Chapter 17

ROW-NORM ESTIMATION

I often say that when you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meagre and unsatisfactory kind; it may be the beginning of knowledge, but you have scarcely, in your thoughts, advanced to the stage of science, whatever the matter may be.

Lord Kelvin [Kel89]

Among matrix attribute estimation problems, the trace estimation and diagonal estimation problems have received the lion's share of the attention. But there is another problem of fundamental interest that also merits study: the problem of estimating the *row norms* of a matrix B.

This chapter uses the leave-one-out approach to develop algorithms for estimating the row norms of an implicit matrix. The resulting XRowNorm and XSymRowNorm estimators are applicable to general and Hermitian matrices, respectively.

Sources. This chapter is based on original research that has not yet been published. It uses the leave-one-out approach developed in the XTrace paper [ETW24].

Outline. The squared row norms of B comprise the diagonal of BB^*, so it is natural to ask: Should we just use diagonal estimators for row-norm estimation? We answer this question with a definitive "no" in Section 17.1. In fact, as we discuss in Section 17.2, sometimes one can (and should) use row-norm estimators to solve diagonal estimation problems! With the importance of purpose-built row-norm estimators established, Section 17.3 describes existing variance-reduced but non-leave-on-out row-norm estimators, Section 17.4 develops the XRowNorm algorithm for general matrices, and Section 17.5 introduces the XSymRowNorm algorithm for Hermitian matrices. Sections 17.6 and 17.7 provide experimental comparisons.

17.1 Is row-norm estimation just diagonal estimation?

Viewed one way, the row-norm estimation problem is a special case of the diagonal estimation problem, owing to the identity $\mathbf{srn}(B) = \mathbf{diag}(BB^*)$. Therefore, the

row-norm estimation problem can be solved by applying a diagonal estimator, such as XNysDiag, to BB^*.

However, the most accurate algorithms for the row-norm estimation problem do not proceed via diagonal estimation. To see why, let us compare two estimates of $\mathbf{srn}(B) = \mathbf{diag}(BB^*)$: The BKS estimator

$$\widehat{\mathbf{srn}}_{\mathrm{BKS}} = \frac{1}{k} \sum_{i=1}^{k} B(B^*\omega_i) \odot \overline{\omega_i} \tag{17.1}$$

versus the Johnson–Lindenstrauss row-norm estimator

$$\widehat{\mathbf{srn}}_{\mathrm{JL}} = \frac{1}{k} \sum_{i=1}^{k} |B\omega_i|^2 = \frac{1}{k} \sum_{i=1}^{k} (B\omega_i) \odot \overline{(B\omega_i)}. \tag{17.2}$$

The first difference between these estimators is their computational cost; to form a k-term estimate, the BKS estimator $\widehat{\mathbf{srn}}_{\mathrm{BKS}}$ requires k matvecs with B and k matvecs with B^*. The Johnson–Lindenstrauss estimate $\widehat{\mathbf{srn}}_{\mathrm{JL}}$ requires just k matvecs with B. Perhaps a more significant difference emerges from the error analysis of these two estimators.

Theorem 17.1 (Comparison of Monte Carlo row norm estimates)**.** *Let $B \in \mathbb{R}^{m \times n}$ be a* real *matrix with rows β_i^*, and consider the squared row norm estimates* (17.1) *and* (17.2) *using iid vectors on the sphere $\sqrt{n}\mathbb{S}(\mathbb{K}^n)$. Then, for each $1 \le i \le m$, we have the variance bounds*

$$\mathbb{E}\left|\widehat{\mathbf{srn}}_{\mathrm{JL}}(i) - \|\beta_i\|^2\right|^2 = \frac{1}{k} \cdot \frac{n-1}{n} K_{\mathbb{K}} \cdot \|\beta_i\|^4, \tag{17.3}$$

$$\mathbb{E}\left|\widehat{\mathbf{srn}}_{\mathrm{BKS}}(i) - \|\beta_i\|^2\right|^2 \le \frac{1}{k} \cdot K_{\mathbb{K}} \cdot \sum_{j=1}^{m} |\beta_i^*\beta_j|^2. \tag{17.4}$$

The constants are $K_{\mathbb{R}} = 2n/(n+2)$ and $K_{\mathbb{C}} = n/(n+1)$.

This result suggests that the Johnson–Lindenstrauss row-norm estimator (17.2) is superior to the BKS diagonal estimator applied to BB^* (17.1). The error of the Johnson–Lindenstrauss estimator of the ith row norm depends only on the ith row norm of B; by contrast, the bound on the error of the BKS estimator depends on the magnitude of the inner products of the ith row of B against all of the other rows. For this example $B = \mathbf{1}\beta^*$, mean-squared error for the Monte Carlo row-norm estimator (17.3) is roughly a factor of m smaller than the bound (17.4) on the mean-squared error for the BKS estimator. On this extreme case, the BKS estimator (17.1) is

essentially worthless as it expends $\Omega(m)$ matvecs to achieve a nontrivial result. (Recall that any matrix attribute estimation problem can be solved at a trivial cost of $\min(m, n)$ matvecs.) The Johnson–Lindenstrauss estimator (17.2), by contrast, computes any row norm of any matrix $\boldsymbol{B}$ to root-mean-squared relative error ε in just $O(1/\varepsilon^2)$ matvecs, independent of the dimensions of $\boldsymbol{B}$.

Theorem 17.1 demonstrates that dedicated algorithms for the row-norm estimation problem can significantly outperform diagonal estimators applied to the outer product matrix $\boldsymbol{BB}^*$. This result justifies our study of the row-norm estimation problem as distinct from diagonal estimation, and motivates the development of purpose-build row-norm estimators.

Proof of Theorem 17.1. Both the BKS and Johnson–Lindenstrauss row-norm estimators (17.1) and (17.2) are averages of k iid copies of a single-vector unbiased estimator; ergo, the mean-squared error for each is $1/k$ times the variance of the single-vector estimator.

Begin with the Johnson–Lindenstrauss estimator

$$\widehat{\mathbf{srn}}_{\mathrm{JL}} = |\boldsymbol{B}\omega|^2,$$

and restrict attention to the ith coordinate $\widehat{\mathbf{srn}}_{\mathrm{MC}}(i) = |\boldsymbol{\beta}_i^*\omega|^2$. We may rewrite this coordinate in the form of a Girard–Hutchinson trace estimator $\widehat{\mathbf{srn}}_{\mathrm{MC}}(i) = \omega^*(\boldsymbol{\beta}_i\boldsymbol{\beta}_i^*)\omega$. The matrix has simple eigenvalue $\|\boldsymbol{\beta}_i\|^2$ and a multiple eigenvalue 0 with multiplicity $n - 1$. The average eigenvalue is $\overline{\lambda} = \|\boldsymbol{\beta}_i\|^2/n$. Therefore, by Fact 13.2, we obtain

$$\mathrm{Var}(\widehat{\mathbf{srn}}_{\mathrm{MC}}(i)) = K_{\mathbb{K}} \left[\left(\|\boldsymbol{\beta}_i\|^2 - \frac{\|\boldsymbol{\beta}_i\|^2}{n} \right)^2 + (n-1) \left(\frac{\|\boldsymbol{\beta}_i\|^2}{n} \right)^2 \right] = \frac{n-1}{n} K_{\mathbb{K}} \cdot \|\boldsymbol{\beta}_i\|^4.$$

Now, we treat the single-vector BKS estimator

$$\widehat{\mathbf{srn}}_{\mathrm{BKS}} = \boldsymbol{BB}^*\omega \odot \overline{\omega}$$

and restrict attention to the ith coordinate $\widehat{\mathbf{srn}}_{\mathrm{BKS}}(i) = \boldsymbol{\beta}_i^* \boldsymbol{B}^*\omega \cdot \overline{\omega}_i$. Observe that we can write $\overline{\omega}_i = \omega^* \mathbf{e}_1$ and $\boldsymbol{\beta}_i^* = \mathbf{e}_i^* \boldsymbol{B}$. Therefore,

$$\widehat{\mathbf{srn}}_{\mathrm{BKS}}(i) = \overline{\omega}_1 \boldsymbol{B}(i,:)\boldsymbol{B}^*\omega = \omega^*(\mathbf{e}_i\mathbf{e}_i^*\boldsymbol{BB}^*)\omega.$$

Invoking Fact 13.2, we obtain

$$\mathrm{Var}(\widehat{\mathbf{srn}}_{\mathrm{BKS}}(i)) \le K_{\mathbb{K}} \|\mathbf{e}_i\mathbf{e}_i^*\boldsymbol{BB}^*\|_{\mathrm{F}}^2 = K_{\mathbb{K}} \|\boldsymbol{\beta}_i^*\boldsymbol{B}^*\|^2 = K_{\mathbb{K}} \sum_{j=1}^{m} |\boldsymbol{\beta}_i^*\boldsymbol{\beta}_j|^2.$$

This completes the proof of the stated results. $\qquad\square$

Remark 17.2 (History). Standard probabilistic proofs of the Johnson–Lindenstrauss lemma [JL84] show that $\widehat{\mathbf{srn}}_{\mathrm{JL}}$ approximates each squared row norm of $\boldsymbol{B}$ up to a constant factor, provided the number of matvecs is $k = O(\log n)$. As such, it is natural to attribute the estimator $\widehat{\mathbf{srn}}_{\mathrm{JL}}$ to Johnson and Lindenstrauss. However, despite being an immediate corollary of the Johnson–Lindenstrauss lemma (1984), I was unable to find a reference where $\widehat{\mathbf{srn}}_{\mathrm{JL}}$ was used to estimate the row norms of an implicit matrix earlier than Spielman and Srivastava's work in 2008 [SS08]. The papers [DMMW12; LMP13] also use this technique. The value of purpose-built row-norm estimators over diagonal estimators is emphasized by Mathur, Moka, and Botev [MMB21] and in recent works of Michael Lindsey [Lin23; FL24]. $\qquad\diamond$

Numerical demonstration and application: Leverage-score estimation

As an application and a demonstration of Theorem 17.1, we compare the Johnson–Lindenstrauss row-norm estimator and the BKS and XNʏsDɪᴀɢ diagonal estimators for the problem of *leverage-score estimation*. Recall that the leverage scores of a matrix $\boldsymbol{B}$ are the squared row norms of $\boldsymbol{Q} \coloneqq \mathbf{Orth}(\boldsymbol{B})$. Estimating leverage scores is major application for row-norm estimators. Following [CP19; MMMW+23], we focus on leverage-score estimation for polynomial regression.

Let $x_1, \ldots, x_m$ denote equally spaced points on $[-1, 1]$, and define the polynomial regression matrix $\boldsymbol{B} \in \mathbb{R}^{m \times n}$ with entries $b_{ij} = T_{j-1}(x_i)$; here, T_k denotes the kth Chebyshev polynomial (of the first kind). We set $m \coloneqq 10^5$ and $n \coloneqq 10^3$. Figure 17.1 shows the results of applying the Johnson–Lindenstrauss row-norm estimator to $\boldsymbol{Q}$ and the BKS and XNʏsDɪᴀɢ diagonal estimators to $\boldsymbol{Q}\boldsymbol{Q}^*$, where $\boldsymbol{Q} = \mathbf{Orth}(\boldsymbol{B})$. For each method, we report the approximation ratio between the true leverage scores $\boldsymbol{\ell} \coloneqq \mathbf{srn}(\boldsymbol{Q})$ and the approximate leverage scores $\widehat{\boldsymbol{\ell}}$, calculated as

$$\text{approximation ratio} \coloneqq \max \left\{ \max \left\{ \frac{\widehat{\ell}_i}{\ell_i}, \frac{\ell_i}{\widehat{\ell}_i} \right\} : i = 1, \ldots, n \right\}$$

As seen in the left panel, the approximation ratio for the Johnson–Lindenstrauss row-norm estimator is dramatically lower than for the diagonal estimators. The diagonal estimators also produce many negative leverage-score estimates, which are clearly vacuous. The right panel shows the (positive) leverage-score estimates as a function of the position x. The Johnson–Lindenstrauss estimates closely track the true leverage scores, whereas the other estimates carry essentially no information.
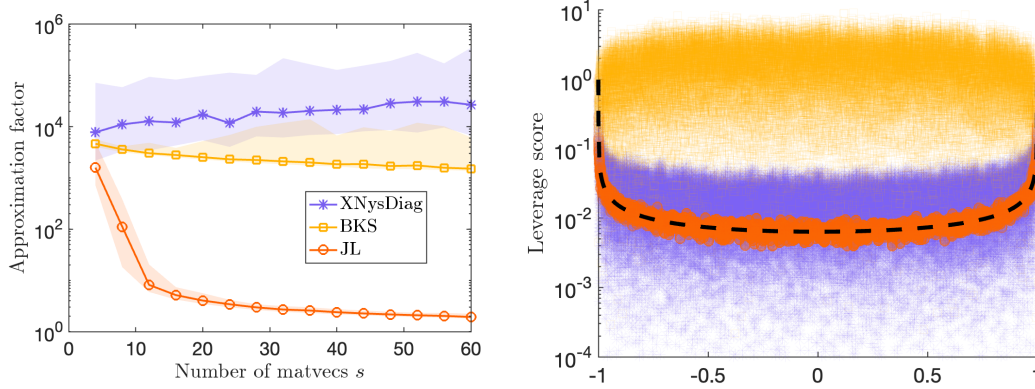
Figure 17.1: *Left:* Approximation factor for Johnson–Lindenstrauss row-norm estimator (orange circles) and BKS (yellow squares) and XNʏsDɪᴀɢ (purple asterisks) diagonal estimators for estimating the leverage scores of the polynomial regression matrix. Lines trace the median of 100 trials, and shaded regions show 10% and 90% quantiles. *Right:* Scatter plot of *positive* leverage-score estimates by these methods (single trial with $s = 60$). The black dashed line indicates the true leverage scores.

Remark 17.3 (Estimating leverage scores: Better algorithms). Given a matrix $B \in \mathbb{K}^{m \times n}$, calculating $Q = \textbf{Orth}(B)$ and applying a row-norm estimate is a poor algorithm. Forming Q expends an expensive $\mathcal{O}(mn^2)$ operations and, if one expends the effort to compute Q, the leverage scores are cheap to compute exactly. To obtain leverage-score estimates quickly, one should instead apply stochastic row-norm estimates to a preconditioned version of the matrix B. This fast approach requires roughly $\mathcal{O}(mn + n^3)$ operations; see [DMMW12] for details. Additionally, the case of polynomial regression, continuous analogs of the leverage scores are known in closed form. For theser easons, Fig. 17.1 serves as comparison point for different algorithms, not as a serious application for row-norm estimation. ⋄

17.2 The square-root trick: Diagonal estimation via row-norm estimation

The last section provided strong evidence that one should not, as a matter of practice, use diagonal estimators to solve row-norm estimation problems. But what about the other way around? Can it ever be beneficial to use row-norm estimators to solve diagonal estimation problems?

The following model describes one scenario where the answer to this question is yes.

> **Square-root matvec model.** The matrix A is psd, and we have *efficient* access to matrix–vector products $z \mapsto A^{1/2}z$ with the square root $A^{1/2}$.

I know of two scenarios where the square-root matvec model applies. First, if the matrix A is *sufficiently well-conditioned*, then matvecs with $A^{1/2}$ can be computed efficiently using the Lanczos algorithm [Che24, Ch. 6]. Second, if $A = f(M)$ is a *nonnegative* function $f : \mathbb{R} \to \mathbb{R}_+$, then matvecs with A are often computed with the Lanczos algorithm. In this setting, matvecs with $A^{1/2} = f^{1/2}(M)$ can often be obtained at the *same* computational cost by applying the Lanczos method with $f^{1/2}$ rather than f. The first setting was observed by Mathur, Moka, and Botev [MMB21], and the second is used in [Lin23].

In the square-root matvec model, we can estimate $\mathbf{diag}(A) = \mathbf{srn}(A^{1/2})$ by estimating the squared row norms of $A^{1/2}$. This approach to diagonal estimation is called the *square-root trick*. It was developed by Mathur, Moka, and Botev [MMB21] and has been used to great effect in the works of Michael Lindsey [Lin23; FL24]. Section 17.7 will illustrate the power of this technique, using the subgraph centrality task from Section 16.5 as an example.

17.3 Variance-reduced row-norm estimators

Row-norm estimators using a HUTCH++-style variance reduction technique were developed by Sobczyk and Luisier [SL22; Sob24]. The original version of their estimator is constructed as follows. Let s be a number of matvecs, evenly divisible by four. Generate a random matrix $\mathbf{\Omega} \in \mathbb{K}^{n \times (s/4)}$ and iid isotropic columns $\boldsymbol{\gamma}_1, \dots, \boldsymbol{\gamma}_{s/4}$. Begin by computing the product

$$Y \coloneqq B^*(B\mathbf{\Omega});$$

Then, orthogonalize $Q \coloneqq \mathbf{Orth}(Y)$, obtaining a low-rank approximation $\widehat{B} \coloneqq BQQ^* \approx B$. We may decompose the squared row norms by incoking the identity

$$\mathbf{srn}(B) = \mathbf{srn}(BQ) + \mathbf{srn}(B(I - QQ^*)),$$

which is immediate from the relation $\mathbf{srn}(B) = \mathbf{diag}(BB^*)$. The first term can be computed exactly by forming BQ, and the second term can be estimated with the Johnson–Lindenstrauss row-norm estimator (17.2). This process yields the variance-reduced estimator

$$\mathbf{srn}_{\mathrm{SL4}} \coloneqq \mathbf{srn}(BQ) + \frac{1}{s/4} \sum_{i=1}^{s/4} |B(I - QQ^*)\boldsymbol{\gamma}_i|^2.$$

We shall call this the Sobczyk–Luisier 4 (SL4) row-norm estimator, as it apportions its matvecs in four batches of size $s/4$. The SL4 method produces an unbiased

Program 17.1 `sl4.m`. Sobczyk–Luisier 4 algorithm for row-norm estimation. The `sqrownorms` subroutine is defined in Program F.3.

```
function srn = sl4(B,Bt,m,n,s)
% Input:  Functions B() and Bt() computing matrix products B(X) =
%         B*X and Bt(X) = B'*X, dimensions m and n, and number of
%         matvecs s
% Output: Estimate srn of the squared row norms of B

k = floor(s/4);                         % Approximation rank s/4
Om = random_signs(m,k);                 % Matrix of random signs
Y = Bt(B(Om));                          % Matvecs with B and B'
[Q,~] = qr(Y,"econ");                   % Orthonormalize
BQ = B(Q);                              % Matvecs with B
k = s - 3*k;                            % Remaining matvecs
Ga = random_signs(n,k);                 % Matrix of random signs
Ga = Ga - Q * (Q'*Ga);                  % Orthogonalize against Q
srn = sqrownorms(BQ) + sqrownorms(B(Ga))/k; % SL4 estimator

end
```

estimate of the squared row norms, and it requires $3s/4$ matvecs with $\boldsymbol{B}$, $s/4$ matvecs with $\boldsymbol{B}^*$, and $O(ns^2)$ additional arithmetic operations. Code is provided in Program 17.1.

Before moving on to leave-one-out estimates of the row norms, let us mention a refinement to the Sobczyk–Luisier estimator. If we generate a random matrix $\boldsymbol{\Omega} \in \mathbb{K}^{m \times (s/3)}$ with $s/3$ columns, we can compute a low-rank approximation $\widehat{\boldsymbol{B}} = \boldsymbol{BQQ}^*$ to $\boldsymbol{B}$ using $2s/3$ matvecs:

$$\boldsymbol{Y} \coloneqq \boldsymbol{B}^*\boldsymbol{\Omega} \quad \text{and} \quad \boldsymbol{Q} \coloneqq \mathbf{Orth}(\boldsymbol{Y}),$$

leaving $s/3$ matvecs to estimate the row norms of the residual. This procedure results in a modified version of the Sobczyk–Luisier estimator, which we call the Sobczyk–Luisier 3 (SL3) estimator:

$$\mathbf{srn}_{\mathrm{SL3}} \coloneqq \mathbf{srn}(\boldsymbol{BQ}) + \frac{1}{s/3} \sum_{i=1}^{s/3} |\boldsymbol{B}(\mathbf{I} - \boldsymbol{QQ}^*)\boldsymbol{\gamma}_i|^2.$$

The computational cost of the SL3 estimator is $2s/3$ matvecs with $\boldsymbol{B}$, $s/3$ matvecs with $\boldsymbol{B}^*$, and $O(ns^2)$ additional arithmetic operations. Code is provided in Program 17.2.

How do the SL3 and SL4 estimates compare? The SL3 estimator uses a cruder low-rank approximation than the SL4 estimator, requiring one less step of subspace

Program 17.2 `sl3.m`. Sobczyk–Luisier 3 algorithm for row-norm estimation. The `sqrownorms` subroutine is defined in Program F.3.

```
function srn = sl3(B,Bt,m,n,s)
% Input:  Functions B() and Bt() computing matrix products B(X) =
%         B*X and Bt(X) = B'*X, dimensions m and n, and number of
%         matvecs s
% Output: Estimate srn of the squared row norms of B

k = floor(s/3);                          % Approximation rank s/3
Om = random_signs(m,k);                  % Matrix of random signs
Y = Bt(Om);                              % Matvecs with B'
[Q,~] = qr(Y,"econ");                    % Orthonormalize
BQ = B(Q);                               % Matvecs with B
k = s - 2*k;                             % Remaining matvecs
Ga = random_signs(n,k);                  % Matrix of random signs
Ga = Ga - Q * (Q'*Ga);                   % Orthogonalize against Q
srn = sqrownorms(BQ) + sqrownorms(B(Ga))/k; % SL3 estimator

end
```

iteration. For a fixed budget of s matvecs, however, the rank of the SL3 estimator is higher than the SL4 estimator, $s/3$ versus $s/4$. Its higher rank often makes the SL3 estimator preferable to the SL4 estimator. For instance, when applied to a matrix $\boldsymbol{B}$ with exponentially decaying singular values $\sigma_i(\boldsymbol{B}) \le \alpha^i$, the SL4 estimator produces estimates of the squared row norms with root-mean-squared error of size $\approx \alpha^{s/2}$, whereas the SL3 estimator achieves a faster error decay $\approx \alpha^{2s/3}$. Thus, for problems with rapid singular value decay, the SL3 estimator is preferable.

17.4 XRowNorm: A leave-one-out row-norm estimator

The SL4 estimator is based on a rank-$(s/4)$ randomized SVD approximation to $\boldsymbol{B}^*$ with one step of subspace iteration, and the SL3 estimator is based on a rank-$(s/3)$ approximation with no subspace iteration. By using a leave-one-out design, we can enjoy the best of both worlds, employing a rank-$(s/3)$ approximation with one step of subspace iteration. We call the resulting estimator XRowNorm. As we will see in Section 17.6, XRowNorm and SL3 perform similarly, with XRowNorm offering a small but noticeable benefit for some instances; see Fig. 17.2.

Begin by drawing a matrix $\boldsymbol{\Omega} \in \mathbb{K}^{n \times (s/3)}$ with iid isotropic columns, and define a rank-$(s/3)$ approximation to $\boldsymbol{B}^*$:

$$Y := B^*(B\Omega), \quad Q := \mathbf{Orth}(Y), \quad \widehat{B} := BQQ^*.$$

Now, employ the leave-one-out method. We obtain a family of downdated low-rank approximations

$$\boldsymbol{Q}_{(i)} \coloneqq \mathbf{Orth}(\boldsymbol{Y}_{-i}), \quad \widehat{\boldsymbol{B}}_{(i)} \coloneqq \boldsymbol{B}\boldsymbol{Q}_{(i)}\boldsymbol{Q}_{(i)}^*.$$

The squared row norms can be decomposed as

$$\mathbf{srn}(\boldsymbol{B}) = \mathbf{srn}\left(\boldsymbol{B}\boldsymbol{Q}_{(i)}\right) + \mathbf{srn}\left(\boldsymbol{B}(\mathbf{I} - \boldsymbol{Q}_{(i)}\boldsymbol{Q}_{(i)}^*)\right).$$

We estimate the second term by the single-vector Johnson–Lindenstrauss estimator using the left-out vector $\boldsymbol{\omega}_i$,

$$\widehat{\mathbf{srn}}_i \coloneqq \mathbf{srn}\left(\boldsymbol{B}\boldsymbol{Q}_{(i)}\right) + \left|\boldsymbol{B}(\mathbf{I} - \boldsymbol{Q}_{(i)}\boldsymbol{Q}_{(i)}^*)\boldsymbol{\omega}_i\right|^2, \tag{17.5}$$

and average to form the XRowNorm estimator

$$\widehat{\mathbf{srn}}_{\mathrm{X}} \coloneqq \frac{1}{s/3}\sum_{i=1}^{s/3}\widehat{\mathbf{srn}}_i = \frac{1}{s/3}\sum_{i=1}^{s/3}\left[\mathbf{srn}\left(\boldsymbol{B}\boldsymbol{Q}_{(i)}\right) + \left|\boldsymbol{B}(\mathbf{I} - \boldsymbol{Q}_{(i)}\boldsymbol{Q}_{(i)}^*)\boldsymbol{\omega}_i\right|^2\right]. \tag{17.6}$$

Efficient formula and implementation. To evaluate the XRowNorm estimator efficiently, we compute the products $\boldsymbol{G} \coloneqq \boldsymbol{B}\boldsymbol{\Omega}$ and $\boldsymbol{Y} \coloneqq \boldsymbol{B}^*\boldsymbol{G}$, then form the QR decomposition $\boldsymbol{Y} = \boldsymbol{Q}\boldsymbol{R}$. By Theorem 14.1, the randomized SVD downdating is described implicitly by the matrix $\boldsymbol{S}$, obtained by normalizing the columns of $\boldsymbol{R}^{-*}$. We substitute the downdating formula (14.5) into (17.5) and (17.6), and we simplify algebraically. We omit the details. The resulting formulas are

$$\widehat{\mathbf{srn}}_i = \mathbf{srn}(\boldsymbol{Z}) - \mathbf{srn}(\boldsymbol{Z}\boldsymbol{s}_i) + |\boldsymbol{g}_i - \boldsymbol{Z}\boldsymbol{x}_i|^2$$

and

$$\widehat{\mathbf{srn}}_{\mathrm{X}} = \mathbf{srn}(\boldsymbol{Z}) + \frac{1}{s/3}\left[-\mathbf{srn}(\boldsymbol{Z}\boldsymbol{S}) + \mathbf{srn}(\boldsymbol{G} - \boldsymbol{Z}\boldsymbol{X})\right].$$

where $\boldsymbol{Z} \coloneqq \boldsymbol{B}\boldsymbol{Q}$ and

$$\boldsymbol{X} \coloneqq \boldsymbol{W} - \boldsymbol{S} \cdot \mathbf{Diag}(\mathbf{diagprod}(\boldsymbol{S}, \boldsymbol{W})) \quad \text{with } \boldsymbol{W} \coloneqq \boldsymbol{Q}^*\boldsymbol{\Omega}. \tag{17.7}$$

Observe that the formula for $\boldsymbol{X}$ is the same as the formula (14.17) for XTrace. The computational cost of XRowNorm is $2s/3$ matvecs with $\boldsymbol{B}$, $s/3$ matvecs with $\boldsymbol{B}^*$, and $O((m + n)s^2)$ additional arithmetic operations. An implementation of XRowNorm is provided in Program 17.3.

Program 17.3 `xrownorm.m`. Efficient implementation of the XRowNorm estimator. Subroutines `diagprod`, `random_signs`, and `cnormc` appear in Programs 14.1, F.1 and F.2.

```
function srn = xrownorm(B,Bt,n,s)
% Input:  Functions B() and Bt() computing matrix products B(X) =
%         B*X and Bt(X) = B'*X, number of rows n, and number of
%         matvecs s
% Output: Estimate srn of the squared row norms of B

% Randomized SVD and downdating
k = floor(s/3);         % Approximation rank is s/3
Om = random_signs(n,k); % Matrix of random signs
G = B(Om);              % Matvecs with B
Y = Bt(G);              % Matvecs with B'
[Q,R] = qr(Y,"econ");   % Orthogonalize
S = cnormc(inv(R'));    % cnormc normalizes columns

% Compute other necessary matrices
Z = B(Q);               % Matvecs with B
W = Q'*Om;
X = W - S .* diagprod(S,W).';

% Form estimate
srn = sqrownorms(Z) + (-sqrownorms(Z*S) + sqrownorms(G - Z*X))/k;

end
```

17.5 XSymRowNorm: Improved Hermitian row-norm estimation

For Hermitian matrices, we can develop an improved version of XRowNorm that uses an approximation with larger rank $s/2$. We call the resulting estimator XSymRowNorm.

Let $A \in \mathbb{K}^{n \times n}$ be a Hermitian matrix. Begin by drawing a matrix $\mathbf{\Omega} \in \mathbb{K}^{n \times (s/2)}$ and calculating

$$Y := A\mathbf{\Omega}, \quad Q := \mathbf{Orth}(Y).$$

Since A is Hermitian, the left-sided randomized SVD approximation $\widehat{A}_{\text{left}} = QQ^*A$ has the same quality as the right-sided approximation $\widehat{A} = AQQ^*$. Using the right-sided approximation, we may decompose the squared row norms

$$\mathbf{srn}(A) = \mathbf{srn}(AQ) + \mathbf{srn}(A(\mathbf{I} - QQ^*)).$$

Now, invoke the leave-one-out method and introduce downdated Q matrices

$$Q_{(i)} := \mathbf{Orth}(Y_{-i}).$$

Program 17.4 `xsymrownorm.m`. Efficient implementation of the XSᴍRᴏᴡNᴏʀᴍ estimator. Subroutines `diagprod`, `random_signs`, and `cnormc` appear in Programs 14.1, F.1 and F.2.

```matlab
function srn = xsymrownorm(A,n,s)
% Input:  Function A() computing matrix products A(X) = A*X, number
%         of rows n, and number of matvecs s
% Output: Estimate srn of the squared row norms of A

% Randomized SVD and downdating
k = floor(s/2);          % Approximation rank is s/2
Om = random_signs(n,k);  % Matrix of random signs
Y = A(Om);               % Matvecs
[Q,R] = qr(Y,"econ");    % Orthogonalize
S = cnormc(inv(R'));     % cnormc normalizes columns

% Compute other necessary matrices
Z = A(Q);                % Matvecs
W = Q'*Om;
X = W - S .* diagprod(S,W).';

% Form estimate
srn = sqrownorms(Z) + (-sqrownorms(Z*S) + sqrownorms(Y - Z*X))/k;

end
```

Using an orthogonal decomposition of the squared row norms and the a single-vector Johnson–Lindenstrauss estimator for the residual squared norms yields the basic XSᴍRᴏᴡNᴏʀᴍ estimators

$$\widehat{\mathbf{srn}}_i := \mathbf{srn}\left(A\boldsymbol{Q}_{(i)}\right) + \left|A\left(\mathbf{I} - \boldsymbol{Q}_{(i)}\boldsymbol{Q}_{(i)}^{*}\right)\omega_i\right|^2.$$

Finally, average to produce the XSᴍRᴏᴡNᴏʀᴍ estimator

$$\widehat{\mathbf{srn}}_{\mathrm{XS}} := \frac{1}{s/2}\sum_{i=1}^{s/2}\widehat{\mathbf{srn}}_i = \frac{1}{s/2}\sum_{i=1}^{s/2}\left[\mathbf{srn}\left(A\boldsymbol{Q}_{(i)}\right) + \left|A\left(\mathbf{I} - \boldsymbol{Q}_{(i)}\boldsymbol{Q}_{(i)}^{*}\right)\omega_i\right|^2\right].$$

Efficient formula and implementation. The efficient formula for XSᴍRᴏᴡNᴏʀᴍ is similar to XRᴏᴡNᴏʀᴍ. Compute the matrix product $\boldsymbol{Y} := A\boldsymbol{\Omega}$, form the QR decomposition $\boldsymbol{Y} = \boldsymbol{Q}\boldsymbol{R}$, and produce the downdating matrix $\boldsymbol{S}$ by normalizing the columns of $\boldsymbol{R}^{-*}$. Set $\boldsymbol{Z} := A\boldsymbol{Q}$ and produce the matrix $\boldsymbol{X}$ from (17.7). The basic XSᴍRᴏᴡNᴏʀᴍ estimates are

$$\widehat{\mathbf{srn}}_i = \mathbf{srn}(\boldsymbol{Z}) - \mathbf{srn}(\boldsymbol{Z}\boldsymbol{s}_i) + |\boldsymbol{y}_i - \boldsymbol{Z}\boldsymbol{x}_i|^2$$
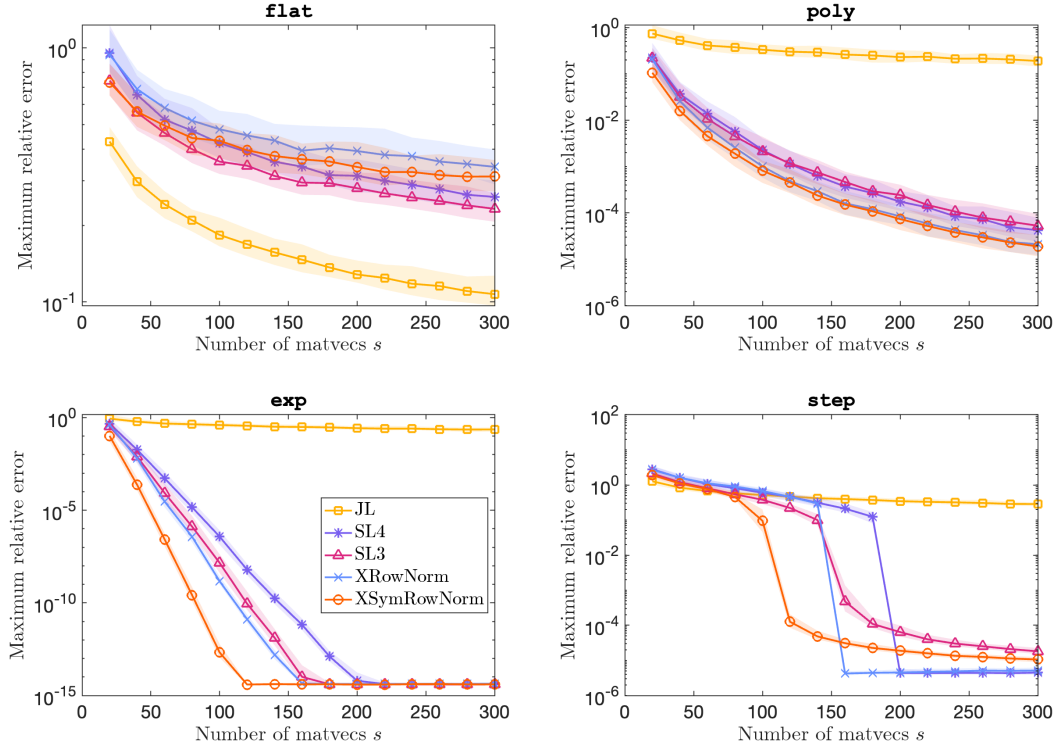
Figure 17.2: Relative error of squared row-norm estimates by the JL estimator (yellow squares), SL4 estimator (purple asterisks), SL3 (pink triangles), XRowNorm (blue crosses), and XSymRowNorm (orange circles) for the four test matrices (14.27) for different numbers of matvecs s. The error metric is maximum relative error, defined in (17.8). Lines show median of 100 trials, and error bars show 10% and 90% quantiles.

and the XSymRowNorm estimator is

$$\widehat{\mathbf{srn}}_{\mathrm{XS}} = \mathbf{srn}(\mathbf{Z}) + \frac{1}{s/2} \left[-\mathbf{srn}(\mathbf{ZS}) + \mathbf{srn}(\mathbf{Y} - \mathbf{ZX}) \right].$$

The computational cost of XSymRowNorm is s matvecs with $\mathbf{A}$ and $O(ns^2)$ additional arithmetic operations.

Remark 17.4 (XSymRowNorm for non-Hermitian matrices). While we have developed the XSymRowNorm estimator for the purpose of estimating the row norms of a Hermitian matrix, the estimator also works for normal matrices or square matrices that are nearly Hermitian. The XSymRowNorm estimator is always unbiased, is effective when low-rank approximations $\mathbf{QQ^*B}$ and $\mathbf{BQQ^*}$ obtained by applying a projector to the left or right side are similarly good low-rank approximations. ◇

17.6 Synthetic Experiments

Figure 17.2 shows a comparison of row-norm estimators for the four test matrices from (14.27). For each method, we plot the maximum relative error in the *squared* row norms:

$$\text{maxrelerr} = \max_{1 \le i \le n} \frac{|\widehat{\text{srn}}_i - \text{srn}_i|}{|\text{srn}_i|}. \tag{17.8}$$

The results are broadly similar to the experimental results for trace and diagonal estimation in Figs. 14.1 and 16.1. For problems with rapid spectral decay, most notably the exp matrix, the errors of the methods are sorted XSYMROWNORM < XROWNORM < SL3 < SL4 ≪ JL. The XROWNORM and SL3 estimators perform similarly on this example, but the XROWNORM estimator edges it out by incorporating an additional step of subspace iteration. The results for the step matrix also demonstrate the benefit of this extra subspace iteration step, with SL4 and XROWNORM outperforming SL3 and XSYMROWNORM for sufficiently large values of s.

17.7 Application: Subgraph centralities, again

Now equipped with the square-root trick and varianced-reduced diagonal estimators, we return to the task of estimating the subgraph centralities from Section 16.5. Recall that the subgraph centralities of a graph with adjacency matrix M are

$$\mathbf{sc} := \mathbf{diag}(\exp(M)).$$

In Section 16.5, we performed matvecs with $A = \exp(M)$ using the Lanczos method [Che24, Ch. 6] associated with the function $f(t) = \exp(t)$. To apply matvecs with $A^{1/2}$, we can simply change the function to $f(t) = \exp(t/2)$. Thus, for this problem, matvecs with $A^{1/2}$ are just as cheap as matvecs with A, making this an ideal setting to apply the square-root trick.

Figure 17.3 measures the accuracy of the subgraph centrality estimated by the BKS and XNYsDIAG diagonal estimators and by the JL and XSYMROWNORM row-norm estimators. When evaluated using the maximum relative error (*left*), we see that the JL and XSYMROWNORM estimators dramatically outperform both diagonal estimators, achieving nearly 10× smaller error using $s = 20$ matvecs. These results demonstrate how the square-root trick can yield diagonal estimates that are much more accurate when the diagonal entries span many orders of magnitude. When using the ℓ_2 error (*right*), the variance-reduced XNYsDIAG and XSYMROWNORM algorithms significantly outperform the Monte Carlo-based JL and BKS estimates. In particular, the XSYMROWNORM estimator achieves the best of both worlds, attaining both small maximum relative error and small ℓ_2 error.
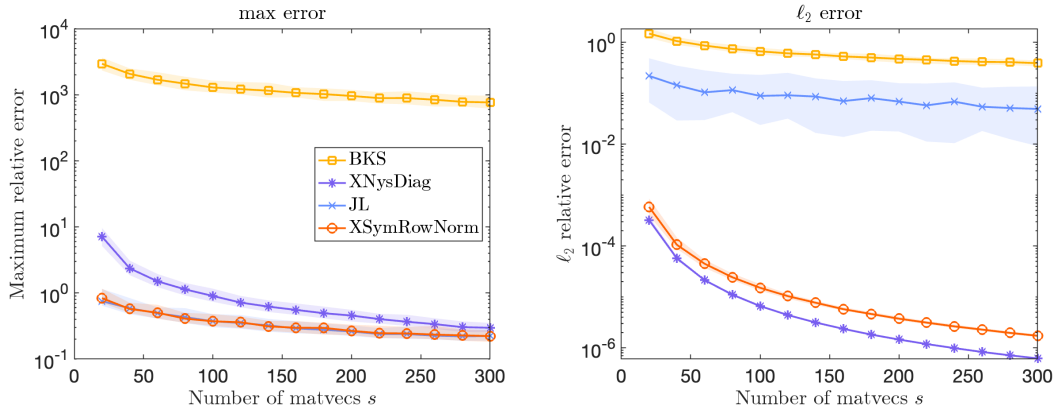
Figure 17.3: Maximum relative error of subgraph centrality estimates by the BKS diagonal estimator (yellow squares), XNʏsDɪᴀɢ (purple asterisks), JL row-norm estimator (blue crosses), and XSʏᴍRᴏᴡNᴏʀᴍ (orange circles) as a function of the number of matvecs s. We show both the the maximum relative error (*left*, (16.6)) and the relative ℓ_2 error (*right*). Lines show the median of 100 trials, and shaded regions show 10% and 90% quantiles.

C h a p t e r   18

POSTERIOR ERROR ESTIMATES FOR LOW-RANK APPROXIMATION

> *. . . we advocate using* a posteriori *error estimators to assess the quality of the output of a randomized linear algebra computation. These error estimators are often quite cheap, yet they can give (statistical) evidence that the computation was performed correctly. We also recommend adaptive algorithms that can detect when the accuracy is insufficient and make refinements. With this approach, it is not pressing to produce theory that justifies all of the internal choices (e.g., the specific type of random embedding) in the NLA algorithm.*
>
> Per-Gunnar Martinsson and Joel A. Tropp, *Randomized numerical linear algebra: Foundations and algorithms* [MT20]

In previous chapters, we employed the leave-one-out approach to estimate attributes of a matrix $\boldsymbol{B}$. In this chapter and the next one, we develop a conceptually distinct application of the leave-one-out technique. Rather than using leave-one-out to *estimate matrix attributes* as previously, we will use the leave-one-out method to *diagnose the quality* of randomized matrix approximations. This chapter will deploy the leave-one-out approach to compute error estimates for low-rank approximation, and next chapter will apply this machinery to estimate the variance of a general class of randomized matrix algorithms. The downdating formulas (Theorems 14.1 and 14.2) will be essential tools in devising efficient algorithms.

Sources. This chapter is based on the paper

Ethan N. Epperly and Joel A. Tropp. "Efficient Error and Variance Estimation for Randomized Matrix Computations". In: *SIAM Journal on Scientific Computing* 46.1 (Feb. 2024), A508–A528. DOI: 10.1137/23M1558537.

Outline. Section 18.1 describes the leave-one-out approach to error estimation in generality, and Section 18.2 applies this technique to the randomized SVD. Section 18.3 contains numerical experiments.

18.1 Leave-one-out error estimation: General approach

We begin by developing the leave-one-out error estimation technique in a general setting; the subsequent section (18.3) will apply this error estimation technique to the randomized SVD.

Setting. Let $\boldsymbol{B} \in \mathbb{K}^{m \times n}$ be an input matrix, and let $\boldsymbol{\Omega} \in \mathbb{K}^{n \times k}$ be a random test matrix with iid isotropic columns. Consider a randomized approximation to $\boldsymbol{B}$ that depends on $\boldsymbol{\Omega}$:

$$\widehat{\boldsymbol{B}} = \widehat{\boldsymbol{B}}_k = \widehat{\boldsymbol{B}}(\boldsymbol{\Omega}). \tag{18.1}$$

We assume that forming the approximation $\widehat{\boldsymbol{B}}$ requires evaluating $\boldsymbol{B}\boldsymbol{\Omega}$, and we assume that $\widehat{\boldsymbol{B}}(\boldsymbol{\Omega})$ is defined for input matrices $\boldsymbol{\Omega}$ with any number of columns. Examples include the randomized SVD and the randomized Nyström approximations, introduced in Section 13.2. We will abuse notation and write $\widehat{\boldsymbol{B}}_k$, $\widehat{\boldsymbol{B}}(\boldsymbol{\Omega})$, and $\widehat{\boldsymbol{B}}$ interchangeably to refer to the approximation (18.1). For simplicity, we shall assume that the approximation $\widehat{\boldsymbol{B}}$ only depends on the deterministic input matrix $\boldsymbol{B}$ and the random test matrix $\boldsymbol{\Omega}$, although the methods in this section apply equally well to approximations $\widehat{\boldsymbol{B}}(\boldsymbol{\Omega}, \boldsymbol{\Psi}, \ldots)$ that depend on additional random quantities $\boldsymbol{\Psi}, \ldots$ that are independent of $\boldsymbol{\Omega}$.

We are interested in efficiently estimating the mean-squared Frobenius error

$$\mathrm{MSE}_k \coloneqq \mathbb{E}\, \|\boldsymbol{B} - \widehat{\boldsymbol{B}}_k\|_{\mathrm{F}}^2.$$

This type of estimate has several applications. For example, it can be used to adaptively select the number of matvecs k that suffice to meet a target accuracy τ: Simply increment k until the estimate $\widehat{\mathrm{MSE}}$ satisfies $\widehat{\mathrm{MSE}} \le \tau$.

Existing approach: Girard–Hutchinson norm estimator. The Girard–Hutchinson norm estimator [MT20, §4.8] provides one way of estimating the error $\|\boldsymbol{B} - \widehat{\boldsymbol{B}}\|_{\mathrm{F}}$. Given a matrix $\boldsymbol{C} \in \mathbb{K}^{m \times n}$, the Girard–Hutchinson estimate of the squared Frobenius norm is

$$\widehat{\mathrm{SN}} \coloneqq \frac{1}{s}\|\boldsymbol{C}\boldsymbol{\gamma}_i\|^2, \tag{18.2}$$

where $\boldsymbol{\gamma}_1, \ldots, \boldsymbol{\gamma}_s$ are iid isotropic vectors. Observe that the Girard–Hutchinson squared norm estimate $\widehat{\mathrm{SN}}$ coincides with the Girard–Hutchinson estimator for $\mathrm{tr}(\boldsymbol{C}^*\boldsymbol{C}) = \|\boldsymbol{C}\|_{\mathrm{F}}^2$. Applying the Girard–Hutchinson norm estimator to $\boldsymbol{C} = \boldsymbol{B} - \widehat{\boldsymbol{B}}$ gives an unbiased estimate to the mean-squared error MSE_k. As disadvantages of this approach, computing the estimate $\widehat{\mathrm{SN}}$ requires s fresh matvecs with the matrix $\boldsymbol{B}$, and the only way to improve the estimate is to increase the number of matvecs

s. The leave-one-out approach will give us an alternative error estimation technique that require no additional matvecs with $\boldsymbol{B}$ and that automatically improves with the approximation quality parameter k.

Leave-one-out errorr estimation. By now, our approach to designing a leave-one-out error estimate should be routine. Begin by downating the approximation $\widehat{\boldsymbol{B}}$ by leaving out each column $\boldsymbol{\omega}_i$ out in turn, producing a family of approximations

$$\widehat{\boldsymbol{B}}_{(i)} \coloneqq \widehat{\boldsymbol{B}}(\boldsymbol{\Omega}_{-i}) \quad \text{for } i = 1, \ldots, k.$$

Then, use the left-out vector $\boldsymbol{\omega}_i$ to form a Girard–Hutchinson norm estimate of each downdated approximation:

$$\widehat{\mathrm{MSE}}_i \coloneqq \|(\boldsymbol{B} - \widehat{\boldsymbol{B}})\boldsymbol{\omega}_i\|^2 \approx \left\|\boldsymbol{B} - \widehat{\boldsymbol{B}}_{(i)}\right\|_{\mathrm{F}}^2.$$

Finally, average these basic estimates to obtain the leave-one-out error estimator

$$\widehat{\mathrm{MSE}} \coloneqq \frac{1}{k} \sum_{i=1}^{k} \widehat{\mathrm{MSE}}_i = \frac{1}{k} \sum_{i=1}^{k} \left\|(\boldsymbol{B} - \widehat{\boldsymbol{B}}_{(i)})\boldsymbol{\omega}_i\right\|^2.$$

This leave-one-out error estimation $\widehat{\mathrm{MSE}}$ has several attractive features. First, the error estimate requires with no additional matvecs with $\boldsymbol{B}$ beyond those necessary to form $\boldsymbol{B\Omega}$. Second, the quality of this error estimate automatically improves as the accuracy parameter k is increased. The leave-one-out error estimator does have one significant flaw: The estimator $\widehat{\mathrm{MSE}}$ *is an unbiased estimate of the mean-squared error of the mean-squared error of the approximation $\widehat{\boldsymbol{B}}_{k-1}$ with parameter $k-1$!*

Proposition 18.1 (Leave-one-out error estimator). *Instate the prevailing notation. The leave-one-out estimator $\widehat{\mathrm{MSE}}$ is an unbiased estimator for* MSE_{k-1}*. That is,*

$$\mathbb{E}[\widehat{\mathrm{MSE}}] = \mathrm{MSE}_{k-1}.$$

The mean-squared error for many types of approximation $\widehat{\boldsymbol{B}}_k$, such as the randomized SVD and the randomized Nyström approximations, are monotone $\mathrm{MSE}_k \leq \mathrm{MSE}_{k-1}$. For such algorithms, the leave-one-out estimate $\widehat{\mathrm{MSE}}$ *overestimates* of the error, on average.

Since it is often more convenient to work with the error rather than the squared error, we define the leave-one-out estimate of the *error* to be

$$\widehat{\mathrm{Err}} \coloneqq \widehat{\mathrm{MSE}}^{1/2}.$$

18.2 Randomized SVD error estimation

Efficient implementations of the leave-one-out error estimates for the randomized SVD can be derived using the randomized SVD downdating formula (14.5). The implementation takes different forms with, and without, additional subspace iteration. We will present only the version without subspace iteration; see [ET24] for the version with subspace iteration.

Generate a random matrix $\mathbf{\Omega}$ with iid isotropic columns, and form the randomized SVD approximation $\widehat{\boldsymbol{B}} := \boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{B}$ with a matrix product $\boldsymbol{Y} := \boldsymbol{B}\boldsymbol{\Omega}$ and QR decomposition $\boldsymbol{Y} = \boldsymbol{Q}\boldsymbol{R}$. For the derivation, introduce the downdating matrix

$$\boldsymbol{S} := \boldsymbol{R}^{-*} \cdot \mathbf{Diag}\left(\|\boldsymbol{R}^{-*}(:,i)\|^{-1} : 1 \le i \le k\right), \tag{18.3}$$

which contains the normalized columns of $\boldsymbol{R}^{-*}$. The leave-one-out error estimate is

$$\mathrm{MSE} = \frac{1}{k} \sum_{i=1}^{k} \left\|(\mathbf{I} - \boldsymbol{Q}(\mathbf{I} - \boldsymbol{s}_i\boldsymbol{s}_i^*)\boldsymbol{Q}^*\boldsymbol{B}\boldsymbol{\omega}_i\right\|^2.$$

By construction, $\mathbf{I} - \boldsymbol{Q}\boldsymbol{Q}^*$ is an orthoprojector annihilating $\mathrm{range}(\boldsymbol{Y}) = \mathrm{range}(\boldsymbol{B}\boldsymbol{\Omega})$, from which $(\mathbf{I} - \boldsymbol{Q}\boldsymbol{Q}^*)\boldsymbol{B}\boldsymbol{\omega}_i$. In addition, $\boldsymbol{Q}^*\boldsymbol{B}\boldsymbol{\Omega} = \boldsymbol{Q}^*\boldsymbol{Y} = \boldsymbol{R}$, so $\boldsymbol{Q}^*\boldsymbol{B}\boldsymbol{\omega}_i = \boldsymbol{r}_i$. Thus,

$$\mathrm{MSE} = \frac{1}{k} \sum_{i=1}^{k} \left\|\boldsymbol{Q}\boldsymbol{s}_i\boldsymbol{s}_i^*\boldsymbol{r}_i\right\|^2 = \frac{1}{k} \sum_{i=1}^{k} |\boldsymbol{s}_i^*\boldsymbol{r}_i|^2.$$

In the second equality, we use the fact that $\boldsymbol{Q}\boldsymbol{s}_i$ is unit vector. Finally, (18.3) implies that

$$\boldsymbol{s}_i^*\boldsymbol{r}_i = (\boldsymbol{S}^*\boldsymbol{R})_{ii} = \mathbf{Diag}\left(\|\boldsymbol{R}^{-*}(:,i)\|^{-1} : 1 \le i \le k\right)_{ii} = \|\boldsymbol{R}^{-*}(:,i)\|^{-1} = \|\boldsymbol{R}^{-1}(i,:)\|^{-1}.$$

Ergo, the leave-one-out estimate of the mean-squared error is

$$\mathrm{MSE} = \frac{1}{k} \sum_{i=1}^{k} \left\|\boldsymbol{R}^{-1}(i,:)\right\|^{-2},$$

and the leave-one-out error estimate is

$$\mathrm{Err} := \left(\frac{1}{k} \sum_{i=1}^{k} \left\|\boldsymbol{R}^{-1}(i,:)\right\|^{-2}\right)^{1/2}. \tag{18.4}$$

The $\boldsymbol{R}$ matrix, often an unused byproduct of the standard randomized SVD implementation, contains enough information by itself to provide an estimate of the error. See Program 18.1 for code.

Program 18.1 `rsvd_errest.m`. Randomized SVD for matrix low-rank approximation together with leave-one-out error estimate. Subroutine `random_signs` is provided in Program F.2.

```
function [U,S,V,est] = rsvd_errest(B,Bt,n,k)
% Input:  Functions B() and Bt() computing matrix products B(X) =
%         B*X and Bt(X) = B'*X, number of columns n, and rank k
% Output: Low-rank approximation Bhat to B, presented as an
%         economy-size SVD Bhat = U*S*V', error estimate

Om = random_signs(n,k);                    % Matrix of random signs
Y = B(Om);                                 % Matvecs with B
[Q,R] = qr(Y,"econ");                      % Orthogonalize
C = Bt(Q);                                 % Matvecs with B'
[UU,S,V] = svd(C',"econ");                 % SVD of factor matrix
U = Q*UU;                                  % Get left singular vectors
est = sqrt(mean(1./sqrownorms(inv(R))));   % Error estimate

end
```

The computational cost of the leave-one-out error estimator is just $O(k^3)$ operations to invert the matrix $\boldsymbol{R}$. In particular, the cost of the leave-one-out error estimator is independent of *both* the dimensions m and n of the input matrix $\boldsymbol{B}$, and it is always faster than the $O((m+n)k^2)$ *post-processing cost* of a standard randomized SVD implementation.

Remark 18.2 (Even faster leave-one-out error estimation). One can accelerate the leave-one-out error estimator even further by estimating the row norms of $\boldsymbol{R}^{-1}$ using a stochastic estimator from Chapter 17. Using the Johnson–Lindenstrauss row-norm estimator ((13.6)), $O(\log k)$ matvecs suffice to estimate all row norms of $\boldsymbol{R}^{-1}$ up to a constant relative error [Woo14, Thm. 4]. Further, each matvec $\boldsymbol{R}^{-1}\boldsymbol{\gamma}$ can be computed in $O(k^2)$ operations. Therefore, the accelerated leave-one-out error estimator requires just $O(k^2 \log k)$ operations, which is much less than the randomized SVD. ◇

18.3   Experiments

Figure 18.1 presents a comparison between the leave-one-out error estimator (18.4) and the Girard–Hutchinson norm estimator (18.2) with $s = 10$ matvecs applied to the randomized SVD on the four test matrices in (14.27). We measure the quality of the error estimates using the approximation factor

$$\alpha := \max\left\{\frac{\text{est}}{\text{err}}, \frac{\text{err}}{\text{est}}\right\} - 1. \tag{18.5}$$
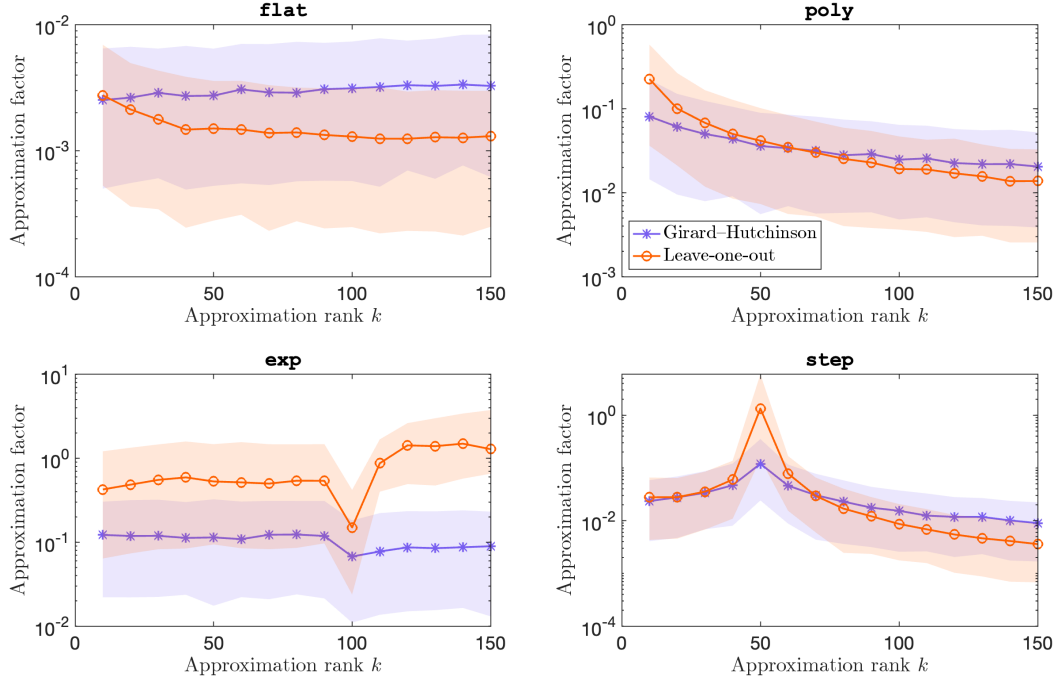
Figure 18.1: Approximation factor for the leave-one-out (orange circles) and Girard–Hutchinson (purple asterisks, $s = 10$ matvecs) estimates for the error of the randomized SVD on the test matrices from (14.27) The error metric is the approximation factor, defined in (18.5). Lines trace the median of 1000 trials, and error bars show the 10% and 90% quantiles.

This is a sensible measure of the approximation quality, because an error estimate with an approximation factor of α is guaranteed to satisfy

$$\frac{1}{1 + \alpha} \cdot \mathrm{est} \leq \mathrm{err} \leq (1 + \alpha)\mathrm{est}.$$

Here are my conclusions:

Leave-one-out error-estimator: pretty good for being free. Before commenting on the finer distinctions between the leave-one-out error estimator and the Girard–Hutchinson norm estimator, it is worth emphasizing from the outset that the leave-one-out error estimator has a basic advantage over the Girard–Hutchinson norm estimator: The leave-one-out error estimator requires no additional matrix–vector products with the matrix $\boldsymbol{B}$, and it requires a small number of arithmetic operations to calculate. The Girard–Hutchinson estimator requires fresh matrix–vector products with the matrix $\boldsymbol{B}$. Thus, even on examples where the Girard–Hutchinson norm estimator is better, the leave-one-out error estimator may still be preferable in many applications, since it is basically free to compute.

Self-improving. Another advantage of the leave-one-out error estimator is that it

is *self-improving*. When one increases the approximation rank k, the estimator automatically improves. This differs from the Girard–Hutchinson estimator, which requires additional matvecs to meaningfully improve quality of the estimator.

Overestimating the error when the singular values drop sharply. The major weakness of the leave-one-out error estimator is that it provides an unbiased estimate of the *rank-*$(k-1)$ randomized SVD error, which can overestimate the rank-k randomized SVD error when the singular values decrease rapidly. This phenomenon is visible at values $k \leq 100$ in the `exp` example and at $k = 50$ in the `step` example.

When the approximation becomes accurate to machine precision. Finally, for $k > 100$ on the `exp` example, the randomized SVD approximation becomes accurate to machine precision $\|B - \widehat{B}\|_F \approx 10^{-16}$. When this happens, the leave-one-out error estimator overestimates the true error by a factor of about 3 due to numerical issues. Fortunately, for most applications, small factors do not matter when an approximation is accurate to machine-precision.

C h a p t e r   19

MATRIX JACKKNIFE VARIANCE ESTIMATION

> Good *simple ideas, of which the jackknife is a prime example, are our*
> *most precious intellectual commodity, so there is no need to apologize*
> *for the easy mathematical level.*
>
> Bradley Efron, *The Jackknife, the Bootstrap and Other Resampling*
> *Plans* [Efr82, p. 1]

In the previous chapter, we discussed the leave-one-out error estimator as a way to assess the error of an approximation $\widehat{B} \approx B$ computed by a randomized algorithm. In this chapter, we shall turn our attention to assessing the quality of randomized approximations to quantities $Q(B)$ obtained from *transforming* a matrix $B \in \mathbb{K}^{m \times n}$ by a *nonlinear* function $Q : \mathbb{K}^{m \times n} \to \mathbb{K}^{m' \times n'}$. There are many examples of such transformations:

1. ***Matrix functions.*** $Q(B) = f(B)$, where $f : \mathbb{R} \to \mathbb{R}$ is a function.

2. ***Best rank-r approximation.*** $Q(B) = [\![B]\!]_r$.

3. ***Top-r singular values.*** $Q(B) = (\sigma_1(B), \ldots, \sigma_r(B))$ for given parameter r.

4. ***Dominant singular subspace projectors.*** $Q(B) = U(:, 1 : r)U(:, 1 : r)^*$ or $Q(B) = V(:, 1 : r)V(:, 1 : r)^*$, where $B = U\Sigma V^*$ is an SVD and r is a given parameter.

Each of these quantities can be estimated by applying a randomized algorithm to produce an initial matrix approximation $\widehat{B} \approx B$ and then using $Q(\widehat{B})$ as a proxy for $Q(B)$. This chapter addresses the following question:

> How can we assess the quality of the approximation $Q(\widehat{B}) \approx Q(B)$ at runtime?

This question is nontrivial. In contrast to the leave-one-out error estimation setting, we typically do not have access to the true quantity of interest $Q(B)$, even via matvecs. Subject to this limitation, it may be infeasible or computationally

intractable to directly estimate the *error* $\|Q(B) - Q(\widehat{B})\|_F$, and we must settle for other diagnostics of the quality of the computed solution. One useful diagnostic is the *variance* of the estimator $Q(\widehat{B})$. The significance of the variance can be justified by appealing to the following intuition [ET24, p. A511]:

> In order to trust the answer provided by a randomized algorithm, the output should be insensitive to the randomness used by the algorithm

The variance provides a quantitative measure of the random fluctuations in the output of a randomized algorithm, allowing us to put this principle into action.

This chapter develops a general approach for estimating the variance of a randomized matrix algorithm based on *jackknife resampling*, a standard technique in statistics. Jackknife resampling is a leave-one-out approach, allowing us to use our matrix downdating results to obtain highly efficient algorithms.

Sources. This chapter is based on the paper

Ethan N. Epperly and Joel A. Tropp. "Efficient Error and Variance Estimation for Randomized Matrix Computations". In: *SIAM Journal on Scientific Computing* 46.1 (Feb. 2024), A508–A528. DOI: 10.1137/23M1558537.

Outline. Section 19.1 begins by describing the bias–variance decomposition of a matrix estimator. We then introduce the matrix jackknife variance technique in generality in Section 19.2, and we discuss its use in Section 19.3. Section 19.4 describes how to efficiently compute matrix jackknife estimates for a broad class of "spectral transformations" of a Nyström approxiamtion to a psd matrix.

19.1 Bias–variance decomposition

Throughout this chapter, we shall be interested in estimating the *variance* of a random matrix $X \in \mathbb{K}^{m \times n}$, defined as

$$\operatorname{Var}(X) \coloneqq \mathbb{E}\,\|X - \mathbb{E}\,X\|_F^2.$$

The matrix variance is the sum of the scalar variances of each entry of X:

$$\operatorname{Var}(X) = \sum_{i=1}^m \sum_{j=1}^n \operatorname{Var}(x_{ij}) = \sum_{i=1}^m \sum_{j=1}^n \mathbb{E}\,|x_{ij} - \mathbb{E}\,x_{ij}|^2. \qquad (19.1)$$

The variance measures the variability of X with respect to the *Frobenius norm*; for discussion of other Schatten norms, see [ET24, §6]. The standard deviation is defined as $\operatorname{SD}(X) \coloneqq \operatorname{Var}(X)^{1/2}$.

Suppose $\widehat{\boldsymbol{B}}$ is a random estimator for the matrix $\boldsymbol{B}$. One convenient measure for the approximation quality $\widehat{\boldsymbol{B}} \approx \boldsymbol{B}$ is the *mean-squared error*

$$\mathrm{MSE} := \mathbb{E}\|\boldsymbol{B} - \widehat{\boldsymbol{B}}\|_{\mathrm{F}}^2.$$

The mean-squared error admits a *bias–variance decomposition*:

$$\mathrm{MSE} = \mathrm{Bias}^2 + \mathrm{Var}(\widehat{\boldsymbol{B}}) \quad \text{where Bias} := \|\boldsymbol{B} - \mathbb{E}\,\widehat{\boldsymbol{B}}\|_{\mathrm{F}}. \qquad (19.2)$$

Consequently, the variance is a *lower bound* on the mean-squared error.

19.2 Matrix jackknife variance estimation

Jackknife resampling is an established approach for approximating the variance of a statistical estimator. Typically, the method is applied to scalar statistics computed from observational data. Here, we will hijack the basic machinery for the matrix-valued outputs of randomized algorithms.

We begin with the general formalism. Let $\omega_1, \ldots, \omega_k$ be iid random variables taking values in a (measurable) space S, and use them to form a matrix-valued estimator $\boldsymbol{X}_k = \boldsymbol{X}_k(\omega_1, \ldots, \omega_k) \in \mathbb{K}^{m \times n}$. We will abuse notation and use $\boldsymbol{X}_k$ to denote both the *deterministic* function

$$\boldsymbol{X}_k : (\omega_1, \ldots, \omega_k) \in \mathsf{S}^k \longmapsto \boldsymbol{X}_k(\omega_1, \ldots, \omega_k) \in \mathbb{K}^{m \times n}.$$

and the output of that function for the random inputs $\omega_1, \ldots, \omega_k$. We assume that the estimator $\boldsymbol{X}_k$ is a *permutation-invariant function* of its inputs $\omega_1, \ldots, \omega_k$.

In statistics and randomized matrix computations, we are often interested in a *family* of estimators $(\boldsymbol{X}_k : k = 1, 2, \ldots)$ that can be instantiated for any natural number k. Examples in statistics include the sample mean and variance:

$$m_k(\omega_1, \ldots, \omega_k) = \frac{1}{k} \sum_{i=1}^{k} \omega_i;$$

$$v_k(\omega_1, \ldots, \omega_k) = \frac{1}{k-1} \sum_{i=1}^{k} |\omega_i - m_k(\omega_1, \ldots, \omega_k)|^2.$$

Examples in randomized matrix computations include the randomized SVD and the randomized Nyström approximation

$$\widehat{\boldsymbol{B}}(\omega_1, \ldots, \omega_k) = \boldsymbol{\Pi}_{\boldsymbol{B}[\omega_1 \cdots \omega_k]} \boldsymbol{B} \quad \text{and} \quad \widehat{\boldsymbol{A}}(\omega_1, \ldots, \omega_k) = \boldsymbol{A} \langle [\omega_1 \cdots \omega_k] \rangle.$$

We often assume the variance is monotone decreasing $\mathrm{Var}(X_k) \leq \mathrm{Var}(X_{k-1})$ in the sample size k.

The jackknife variance estimate was introduced by Tukey in 1958 [Tuk58]. Here is the idea. Define a family of *jackknife replicates* by recomputing the estimator with each one of the samples $\omega_1, \ldots, \omega_k$ left out in turn:

$$X^{(j)} := X_{k-1}(\omega_1, \ldots, \omega_{j-1}, \omega_{j+1}, \ldots, \omega_k).$$

The jackknife variance estimate is defined as

$$\mathrm{Jack}^2(X_{k-1}) := \sum_{j=1}^{k} \left\| X^{(j)} - X^{(\cdot)} \right\|_{\mathrm{F}}^2 \quad \text{where } X^{(\cdot)} := \frac{1}{k} \sum_{j=1}^{k} X^{(j)}. \tag{19.3}$$

The jackknife estimate serves as an approximation of the variance of the $(k-1)$-*sample estimator* X_{k-1}, and it typically provides a modest overestimate of the variance of the k-sample estimator X_k as well.

Observe that the formula (19.3) for the jackknife variance estimator resembles the traditional sample covariance estimate

$$\mathrm{Var}(X_{k-1}) \approx \frac{1}{k-1} \sum_{j=1}^{k} \left\| Z^{(j)} - Z^{(\cdot)} \right\|_{\mathrm{F}}^2 \quad \text{where } Z^{(\cdot)} := \frac{1}{k} \sum_{j=1}^{k} Z^{(j)} \tag{19.4}$$

for iid copies $Z^{(1)}, \ldots, Z^{(k)} \sim X_{k-1}$ of the estimator. An important difference between the jackknife estimate (19.3) and the sample covariance (19.4) is that the jackknife estimate (19.3) is *not* divided by $k-1$. This difference may be justified intuitively. The replicates $X^{(j)}$ are *not independent*, each one differs in a single input coordinate. Since only one out of $k-1$ inputs differs between each pair of replicates, the replicates $X^{(1)}, \ldots, X^{(k)}$ should have a "$(k-1)^{-1}$ fraction" of the variance of true iid copies $Z^{(1)}, \ldots, Z^{(k)} \sim X_{k-1}$.

The quality of jackknife variance estimates can be analyzed using the Efron–Stein–Steele inequality [ES81; Ste86]:

Fact 19.1 (Efron–Stein–Steele inequality). *Let $\omega_1, \ldots, \omega_k$ be independent random variables in a measurable space S, let $f : \mathsf{S}^k \to \mathbb{K}$ be a measurable function, and introduce iid copies $\omega'_1, \ldots, \omega'_k$. Then*

$$\mathrm{Var}(Z) \leq \frac{1}{2} \sum_{j=1}^{k} \mathbb{E} \left| Z - Z^{(j)} \right|^2$$

where $Z := f(\omega_1, \ldots, \omega_k)$ and $Z^{(j)} := f(\omega_1, \ldots, \omega_{j-1}, \omega'_j, \omega_{j+1}, \ldots, \omega_k)$.

The Efron–Stein–Steele inequality is a powerful result that can be used to great effect in high-dimensional probability. See [van14, §2.1] and [Tro21, §2.7] for accessible introductions and [BLM13] for many uses of this inequality in probability theory. It yields the following consequence for the jackknife estimator (19.3).

Theorem 19.2 (Matrix jackknife overestimates variance). *With the prevailing notation and assumptions,*

$$\operatorname{Var}(X_{k-1}) \leq \mathbb{E}\operatorname{Jack}^2(X_{k-1}).$$

This result states the matrix jackknife variance estimator (19.3) overestimates the variance of the $(k-1)$-sample estimator *on average*. The fact that the jackknife estimate overestimates the true variance is a bit disappointing, but it is remarkable that the jackknife variance estimator possesses guarantees at this level of generality. In practice, on the examples we consider in this thesis, the jackknife variance estimate tends to be accurate to within an order of magnitude, which is good enough to provide actionable information in applications.

Proof of Theorem 19.2. First consider a scalar statistic $x_{k-1} = x_{k-1}(\omega_1, \ldots, \omega_{k-1})$ and introduce replicates $x^{(j)} = x_{k-1}(\omega_1, \ldots, \omega_{j-1}, \omega_{j+1}, \ldots, \omega_k)$ with mean $x^{(\cdot)}$. Observe that the replicates $x^{(j)}$ share the same distribution. Additionally,

$$x_{k-1} = x_{k-1}(\omega_1, \ldots, \omega_{k-1}) = x^{(k)}.$$

By the Efron–Stein–Steele inequality,

$$\operatorname{Var}(x_{k-1}) \leq \frac{1}{2} \sum_{j=1}^{k-1} \mathbb{E}\left| x_{k-1} - x_{k-1}(\omega_1, \ldots, \omega_{j-1}, \omega'_j, \omega_{j+1}, \ldots, \omega_{k-1}) \right|^2,$$

where ω'_j denotes an independent copy of ω_j. The ω_j are iid random variables, so wecan replace ω'_j by ω_k. Thus, we obtain

$$\operatorname{Var}(x_{k-1}) \leq \frac{1}{2} \sum_{j=1}^{k-1} \mathbb{E}\left| x^{(k)} - x^{(j)} \right|^2.$$

Since each element of $\{x^{(\ell)} - x^{(j)} : \ell \neq j\}$ has the same distribution, we can replace $x^{(k)}$ by $x^{(\ell)}$ and average over the index ℓ, obtaining

$$\operatorname{Var}(x_{k-1}) \leq \frac{1}{2k} \sum_{j,\ell=1}^{k} \mathbb{E}\left| x^{(\ell)} - x^{(j)} \right|^2.$$

Next, we compute

$$
\begin{aligned}
\mathbb{E} \sum_{j=1}^{k} \left| x^{(j)} - x^{(\cdot)} \right|^2 &= \mathbb{E} \sum_{j=1}^{k} \left| x^{(j)} - \frac{1}{k} \sum_{\ell=1}^{k} x^{(\ell)} \right|^2 \\
&= \mathbb{E} \sum_{j=1}^{k} \left[\left| x^{(j)} \right|^2 - \frac{2}{k} \sum_{\ell=1}^{k} \mathrm{Re} \left(\overline{x^{(j)}} x^{(\ell)} \right) - \frac{1}{k^2} \sum_{\ell,p=1}^{k} \mathrm{Re} \left(\overline{x^{(\ell)}} x^{(k)} \right) \right] \\
&= \mathbb{E} \left[\frac{1}{2} \sum_{j=1}^{k} \left| x^{(j)} \right|^2 + \frac{1}{2} \sum_{\ell=1}^{k} \left| x^{(j)} \right|^2 - \frac{1}{k} \sum_{j,\ell=1}^{k} \mathrm{Re} \left(\overline{x^{(j)}} x^{(\ell)} \right) \right] \\
&= \frac{1}{2k} \sum_{j,\ell=1}^{k} \mathbb{E} \left| x^{(\ell)} - x^{(j)} \right|^2 = \mathrm{Jack}^2(x_{k-1}).
\end{aligned}
$$

In the second line, we expand the square. In the third line we consolidate the second and third terms, break the sum $\sum_{j=1}^{k} |x^{(j)}|^2$ into two equal pieces, and reindex by replacing j with ℓ. In the last line, we recombine. Joining the two previous displays leads to the the desired result

$$
\mathrm{Var}(x_{k-1}) \le \mathrm{Jack}^2(x_{k-1}). \tag{19.5}
$$

The matrix case follows by decomposing the matrix variance as a sum of the variance of its entries (19.1) and invoking (19.5) entry-by-entry. $\qquad\square$

Remark 19.3 (What about the bias?). Jackknife estimates of the bias were developed by Quenouille in 1949 [Que49], predating jackknife variance estimates by nearly a decade. Quenouille's bias estimate can be formed for a randomized matrix approximation, but its usefulness in this setting is unclear. Quenouille estimate is typically analyzed under the assumption that the bias can be expanded in reciprocal powers of k. Randomized low-rank approximation algorithms seemingly do not satisfy this property. Thus, it is unclear what, if any, insight Quenouille's bias estimate provides for randomized matrix algorithms. $\qquad\diamond$

19.3 Using matrix jackknife variance estimation

In principle, the use cases for matrix jackknife variance estimation could be very broad. In this thesis, we will apply jackknife variance estimation to quantities $X_k = Q(\widehat{A})$ computed from a randomized Nyström approximation $\widehat{A} \approx A$; see [ET24] for further examples. To instantiate the jackknife methodology, we treat

$$
X_k = X_k(\omega_1, \ldots, \omega_k)
$$

as a function of the iid columns ω_i of the random test matrix $\mathbf{\Omega}$. We will focus on this setting for the remainder of the chapter.

A limitation of the matrix jackknife variance estimator is that it captures just one term in the bias–variance decomposition (19.2). To approximate the mean-squared error, a separate estimate of the bias is necessary. However, the information provided by the jackknife variance estimate is still actionable.

> If the variance is high, the mean-squared error is high, and the output of the algorithm should not be trusted. If the variance is low, then the jackknife variance estimate is equivocal, as the mean-squared error could be low or high.

For a user, what does it mean if the variance is high? There are at least two possibilities:

1. **Too few samples.** The approximation $\widehat{A}$ is under-resolved. More samples (i.e., more columns in the test matrix $\mathbf{\Omega}$) are needed to produce an approximation $\widehat{A}$ of sufficiently high quality to estimate the quantity of interest.

2. **Bad conditioning.** The quantity of interest $\boldsymbol{Q}(\boldsymbol{A})$ could be *poorly conditioned* in the sense that small changes to the input $\boldsymbol{A}$ result in large changes to $\boldsymbol{Q}(\boldsymbol{A})$. In this case, even if the approximation $\boldsymbol{A} \approx \widehat{\boldsymbol{A}}$ is highly accurate, $\boldsymbol{Q}(\widehat{\boldsymbol{A}})$ can be far from $\boldsymbol{Q}(\boldsymbol{A})$ owing to the inherent sensitivity of the quantity of interest. When dealing with a badly conditioned problems, adding additional samples may not help improve the quality of the approximation $\boldsymbol{Q}(\widehat{\boldsymbol{A}}) \approx \boldsymbol{Q}(\boldsymbol{A})$.

In either case, high variance—diagnosed by a jackknife estimate—flashes a signal not to trust the computed output.

Jackknife variance estimation can be deployed at runtime to detect these bad behaviors. One possible use case is to run the jackknife variance estimate and provide a warning to the user if high variance is detected. As an alternative, jackknife variance estimation can be used to adaptively determine the approximation rank k until the variance drops below a specified threshold.

The main advantage of jackknife variance estimation is it provides a way of estimating the quality of a very general class of quantities produced by randomized matrix approximation algorithms. Like the leave-one-out error estimate, it requires *no*

additional matrix–vector products beyond those used to run the original algorithm. To achieve the maximum efficiency for the jackknife variance estimator, one can develop purpose-build fast implementations using the randomized SVD or the randomized Nyström downdating results (Theorems 14.1 and 14.2). The remainder of this chapter develops an example of application of the matrix jackknife for assessing the quality of "spectral transformations" of Nyström approximations; see [ET24] for several more examples.

19.4 Example: Spectral transformations of Nyström approximations

Many of the ways we *use* Nyström approximations to psd matrices can be encompassed under the umbrella of applying a *spectral transformation* to the matrix. We make the following definition:

Definition 19.4 (Spectral transformation). Let f denote a vector-valued function $f : \mathbb{R}_+^n \to \mathbb{C}^n$, and let $A \in \mathbb{K}^{n \times n}$ denote a psd matrix with eigendecomposition $A = U \operatorname{Diag}(\lambda(A))U^*$. The *spectral transformation* of A by f is

$$f[A] := U \operatorname{Diag}(f(\lambda(A)))U^*.$$

We remind the reader that the eigenvalues $\lambda(A)$ are sorted in *nonincreasing order*. The spectral transformation may fail to be uniquely defined if A has repeated eigenvalues.

The spectral transformation f is *rank-k* if

$$d(k + 1 : n) = 0 \implies [f(d)](k + 1 : n) \quad \text{for every } d \in \mathbb{R}_+^n.$$

For a rank-k spectral transformation, we can extend its domain of definition to matrices of size k by

$$f(d_1, \ldots, d_k) := [f(d_1, \ldots, d_k, 0 \ldots, 0)](1 : k). \tag{19.6}$$

Many of the application of Nyström approximations can be reinterpreted as spectral transformations. Here are several examples:

1. **Spectral projectors:** One of the main uses for Nyström approximation is to approximate eigenvectors. However, even when the eigenvalues are distinct, the eigenvectors are only defined up to a sign (or phase, if $\mathbb{K} = \mathbb{C}$); when there are repeated eigenvalues, the non-uniqueness of eigenvectors become more

severe. To resolve many of these uniqueness issues, we can use *spectral projectors*. Given a subset $\mathsf{S} \subseteq \{1, \dots, n\}$ and a matrix $A = U \operatorname{Diag}(\lambda(A))U^*$, the spectral projector associated with the index set S is

$$\Pi_{\mathsf{S}}(A) = U(:, \mathsf{S})U(:, \mathsf{S})^*.$$

Natural examples include $\mathsf{S} = \{1, \dots, k\}$ (top-k subspace) or $\mathsf{S} = \{j\}$ (single eigenvector). The spectral projector is the orthprojector on the *invariant subspace* spanned by the eigenvectors $\operatorname{span}\{u_s : s \in \mathsf{S}\}$ indexed by S. Spectral projectors are spectral transformation associated with the constant function $f(d) = \sum_{s \in \mathsf{S}} e_s$. As a spectral transformation, the spectral projector function f is rank-k if and only if $\mathsf{S} \subseteq \{1, \dots, k\}$.

2. **Rank-r truncation.** We can truncate rank-k Nyström approximations to smaller rank $r < k$ to reduce storage. The rank-r approximation operation $A \mapsto [\![A]\!]_r$ is also a spectral transformation, associated with the function

$$f(d_1, \dots, d_n) = (d_1, \dots, d_r, 0, \dots, 0) \quad \text{for every } d \in \mathbb{R}^n_+.$$

This spectral transformation is rank-k for every $1 \le k \le n$.

3. **Matrix functions.** Another use of Nyström approximations, developed by David Persson and collaborators [PK23; PMM25], is to approximate a (standard) matrix functions $f(A)$. The idea is straightforward: Given a Nyström approximation $\widehat{A} \approx A$, call $f(\widehat{A})$ as the *funNyström* approximation to $f(A)$. The funNyström approximation is the spectral transformation associated with the function

$$f(d_1, \dots, d_n) = (f(d_1), \dots, f(d_n)) \quad \text{for all } d \in \mathbb{R}^n_+.$$

For every $1 \le k \le n$, this spectral transformation is rank-k if and only if $f(0) = 0$. (The requirement that $f(0) = 0$ can always be satisfied by shifting $\widetilde{f} := f - f(0)$.)

Given these various settings, it is natural to seek ways of estimating the variance of spectral transformations of a Nyström approximation.

For the rest of this section, we consider the task of estimating a general spectral transformation $f[\widehat{A}]$ using a single-pass Nyström approximation $\widehat{A} = A\langle \Omega \rangle$ to a psd matrix A. We assume the test matrix $\Omega \in \mathbb{K}^{n \times k}$ is composed of *independent* columns, and we assume that f has rank k.

Variance estimation for spectral transformations: $O(k^4)$ algorithm

The simplest approach to variance estimation for spectral transformations passes through a variant of the downdating formula (14.25) for Nyström approximations.

Begin by computing the Nyström approximation as in Program 2.4: We form $Y := A\Omega$, compute $H := \Omega^* Y$, factorize $H = R^* R$, define $F := YR^{-1}$, and compute an SVD $F = U\Sigma V^*$. The Nyström approximation is now given as

$$\widehat{A} = UDU^* \quad \text{where } D = \Sigma^2.$$

(For numerically stable implementation in practice, one should add shift correction to this procedure; see (2.11).) Following (14.23), the family of downdated Nyström approximations $\widehat{A}_{(i)} = A\langle \Omega_{-i} \rangle$ is

$$\widehat{A}_{(i)} = UDU^* - z_i z_i^* \quad \text{where } Z := FR^{-*} \cdot \mathbf{Diag}\left(\mathbf{srn}(R^{-1})\right)^{-1/2}.$$

Using the SVD $F = U\Sigma V^*$, we can rewrite the downdating rule as

$$\widehat{A}_{(i)} = U(D - w_i w_i^*)U^* \quad \text{where } W = \Sigma V^* R^{-*} \cdot \mathbf{Diag}\left(\mathbf{srn}(R^{-1})\right)^{-1/2}.$$

Consequently, the entire family of Nyström approximations is given by the W matrix, which can be computed in $O(k^3)$ operations.

Using the downdating matrix W in a straightforward way, we can compute the jackknife variance estimate in $O(k^4)$ operations. The spectral transformation of each $\widehat{A}_{(i)}$ is given as

$$f[\widehat{A}_{(i)}] = Uf[D - w_i w_i^*]U^*,$$

where we have used the extension (19.6) of spectral functions to $k \times k$ matrices. Thus, we have

$$\text{Jack}^2(f[\widehat{A}]) = \sum_{j=1}^{k} \left\| f[\widehat{A}_{(j)}] - \frac{1}{k} \sum_{i=1}^{k} f[\widehat{A}_{(i)}] \right\|_F^2 = \sum_{j=1}^{k} \left\| X^{(j)} - X^{(\cdot)} \right\|_F^2,$$

where

$$X^{(j)} := f[D - w_j w_j^*] \quad \text{and} \quad X^{(\cdot)} := \frac{1}{k} \sum_{j=1}^{k} X^{(j)}.$$

Computing each $X^{(j)}$ directly requires a spectral decomposition of $X^{(j)}$ at $O(k^3)$ cost. Since there are k replicates $X^{(j)}$, the total cost is $O(k^4)$ operations. Once the replicates have been formed, the jackknife variance estimate requires an additional $O(k^3)$ arithmetic operations. Code is provided in Program 19.1.

Program 19.1 `nystrom_jack.m`. Single-pass Nyström approximation with jack-knife variance estimation for spectral transformation. Subroutine `nystrom` is provided in Program 2.3.

```
function [U,D,jack] = nystrom_jack(A,n,k,f)
% Input:  Function A() computing matrix products A(X) = A*X, number
%         of rows n, rank k, spectral transformation f()
% Output: Shift-corrected Nystrom approximation Ahat = U*D*U',
%         represented by factors U and D, and jackknife standard
%         deviation estimate jack

[F,mu,~,R] = nystrom(A,n,k); % Compute Nystrom approximation
[U,S,V] = svd(F,"econ");     % Economy-size SVD
D = max(S.^2 - mu, 0);       % Apply shift correction

W = ((S*V')/R')*diag(sqrownorms(inv(R)).^(-0.5)); % Downdating matrix
Xs = zeros(k,k,k);                                % Jackknife replicates
for j = 1:k
    [Q,d] = eig(D - W(:,j)*W(:,j)',"vector");
    d = d(end:-1:1); Q = Q(:,end:-1:1); % Evals in decreasing order
    Xs(:,:,j) = Q * diag(f(d)) * Q';     % Spectral transform
end
jack = norm(Xs - mean(Xs,3),"fro");      % Jackknife estimate

end
```

Acceleration to $O(k^3)$ operations

In some cases, variance estimation for spectral transformations can be accelerated to $O(k^3)$ operations. Specifically, consider a spectral transformation for which $f(d)$ has only r nonzero entries for every $d \in \mathbb{R}^n_+$. To compute $f[D - w_j w_j^*]$ efficiently, observe that $D - w_j w_j^*$ is a rank-one modification to a diagonal matrix D. The spectral decomposition of each $D - w_j w_j^* = Q_{(j)} \mathbf{Diag}(d_{(j)}) Q_{(j)}^*$ can be computed in $O(k^2)$ operations; see [Dem97, §5.3.3]. Then, each $X^{(j)} = f[D - w_j w_j^*]$ can be computed in $O(k^2 r)$ operations. Thus, the total runtime using this approach is $O(k^3 r)$ operations. If $r = O(1)$, the total cost is $O(k^3)$ operations, as promised. See the code of [ET24] for an implementation of this faster algorithm.

Experiment

To demonstrate the effectiveness of the jackknife variance estimation technique, we use it to assess the variance of spectral transformations of Nyström approximations to a psd matrix A. We set A to be the `exp` matrix (14.27c), and we consider two spectral transformations: the dominant rank-10 projector $f[A] := \mathbf{\Pi}_{1:10}(A)$ and the
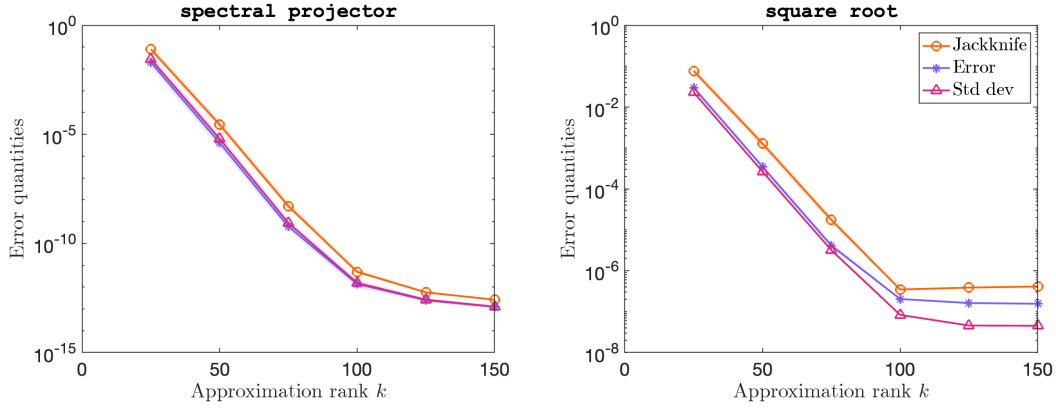
Figure 19.1: Jackknife standard deviation estimate (orange circles), error (purple asterisks), and standard deviation (pink triangles) for spectral transformations of Nyström approximations of ranks $25 \leq k \leq 150$. Spectral transformations are the dominant rank-10 projector $f[A] \coloneqq \Pi_{1:10}(A)$ (*left*) and the matrix square root $f[A] \coloneqq A^{1/2}$ (*right*).

matrix square root $f[A] \coloneqq A^{1/2}$. We test approximation ranks $25 \leq k \leq 150$.

Results appear in Fig. 19.1. For each spectral transformation and each Nyström approximation, we show the error $\|f[A] - f[\widehat{A}]\|_{\mathrm{F}}$, the standard deviation $\mathrm{Var}(f[\widehat{A}])^{1/2}$, and the jackknife standard deviation estimate $\mathrm{Jack}(f([\widehat{A}])$. For both examples, we see that the jackknife variance estimate serves as a modest overestimate of both the error and standard deviation, providing a useful diagnostic of the quality of the computed approximation. Many more examples of the jackknife methodology are provided in [ET24, §§1.2 & 5].

Chapter 20

LEAVE-ONE-OUT RANDOMIZED MATRIX ALGORITHMS: OPEN PROBLEMS

This chapter discusses open problems related to leave-one-out randomized matrix algorithms including error analysis (Section 20.1), leave-one-out algorithms for Hermitian indefinite matrices (Section 20.2), and numerically stable downdating formulas for subspace iteration (Section 20.3).

20.1 Open problem: Error analysis

This part of the thesis has discussed matrix attribute estimation algorithms. The basic versions of XTRACE and XNYSTRACE have *a priori* bounds on the mean-squared error, and most of the remaining algorithms have no error analysis at all. In my view, theoretical analysis of these algorithms is not a major limitation; they are based on sound principles—unbiased Monte Carlo approximation, variance reduction by low-rank approximation, and exchangeable leave-one-out design—and they achieve *spectral accuracy* in practice. Still, it is natural to desire for a more comprehensive mathematical analysis of these algorithms.

There are several natural topics for future research:

- Develop sharp error bounds for XTRACE and XNYSTRACE that hold with high-probability.

- Obtain *a priori* error bounds, either high-probability or on the mean-squared error, for the randomized diagonal estimators and row-norm estimators from Chapters 16 and 17.

- Obtain bounds on the variance of the leave-one-error estimate from Chapter 14.

20.2 Open problem: Hermitian indefinite matrices

This thesis has developed leave-one-out matrix attribute estimation algorithms for general matrices using the randomized SVD (e.g., XTRACE) and algorithms for psd matrices using single-pass Nyström approximation (e.g., XNYSTRACE). Can optimized leave-one-out algorithms be devised for matrices that have additional

Program 20.1 `xsymtrace.m`. Implementation of the XNᴙꜱTʀᴀᴄᴇ algorithm designed to work with Hermitian indefinite matrices. Subroutines `diagprod` and `random_signs` appear in Program 14.1 and Program F.2.

```
function [tr,est] = xsymtrace(A,n,s)
% Input:  Function A() computing matrix products A(X) = A*X, number
%         of rows n, and number of matvecs s
% Output: Estimate tr of trace(A), estimate est of the error
%         abs(tr - trace(A))

% Nystrom approximation
Om = random_signs(n,s);          % Test matrix of random signs
Y = A(Om);                       % Matrix product Y = A*Om
H = Om'*Y;
[L,D] = ldl((H+H')/2);           % LDLt factorization
F = Y/L';                        % Triangular substitution

% Downdating
Z = (F/D)/L;
d = diag(inv(H)); % Downdated approx is F*D*F'-Z(:,i)*Z(:,i)'/d(i)

% Compute vector of estimates
tr_vec = trace(D\(F'*F)) * ones(s,1) - sqcolnorms(Z) ./ d ...
    + abs(diagprod(Om,Z)) .^ 2 ./ d;
tr = mean(tr_vec);          % Trace estimate
est = std(tr_vec) / sqrt(s); % Error estimate

end
```

structure, but are not psd? In particular, is there a way to improve leave-one-out matrix algorithms for Hermitian *indefinite* matrices?

As we discussed in Remark 2.10, one *can* apply single-pass Nyström approximation to Hermitian indefinite matrices without modification, although it can be inaccurate on certain instances. Program 20.1 provides an implementation of XSʏᴍTʀᴀᴄᴇ, a version of XNᴙꜱTʀᴀᴄᴇ designed to work with Hermitian indefinite matrices. Results for this algorithm on our testbed of synthetic matrices with different spectral characteristics appear in Fig. 20.1. We use the same test matrices from previous sections, except that we randomize the sign of each eigenvalue so the matrices are indefinite. We see that the XSʏᴍTʀᴀᴄᴇ estimator is substantially less accurate than XTʀᴀᴄᴇ on three of four examples, demonstrating the failure of (single-pass) Nyström approximation for indefinite matrices. Thus, we do not regard XSʏᴍTʀᴀᴄᴇ as an effective general-purpose algorithm for trace estimation of Hermitian indefinite
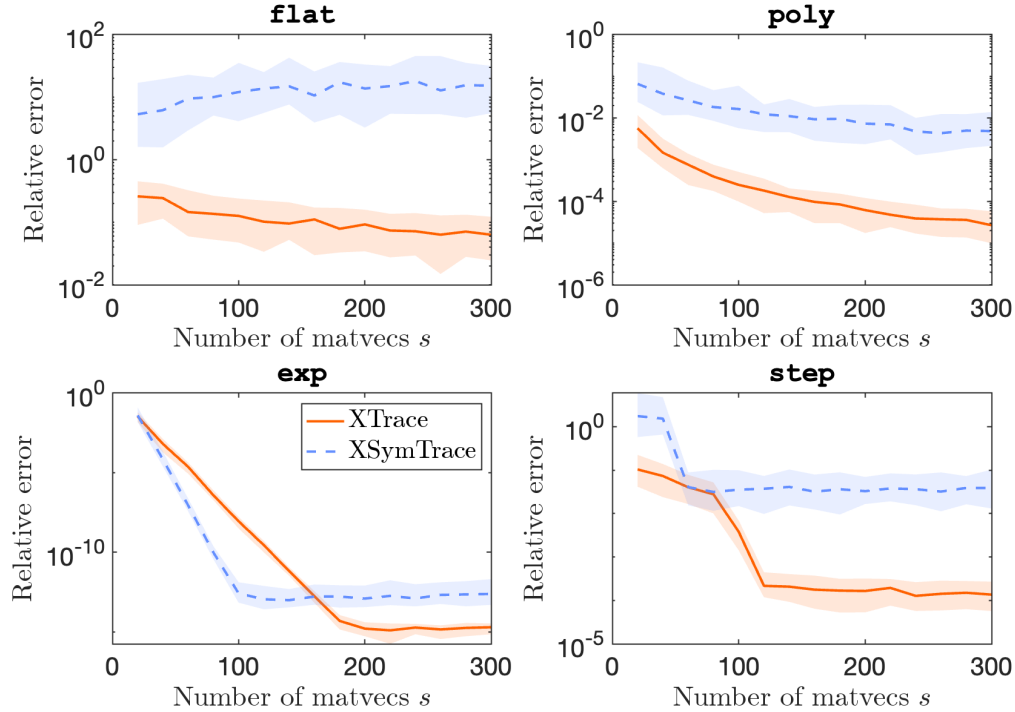
Figure 20.1: Comparison of XTRACE (orange solid) and XSYMTRACE (blue dashed) on four test matrices (14.27) with different spectral characteristics. Lines show median of 100 trials, and shaded regions show 10% and 90% quantiles.

matrices. (Nevertheless, its performance may be acceptable for matrices with only a *small* number of negative eigenvalues or with very rapid spectral decay.)

For this reason, determining the "right" approach to developing leave-one-out algorithms for Hermitian indefinite matrices remains an open problem. The oversampling technique discussed in Remark 2.10 may be a promising strategy, although downdating the regularized pseudoinverses $[\![\boldsymbol{\Omega}^* \boldsymbol{A} \boldsymbol{\Omega}]\!]_k^\dagger$ may be a nontrivial task. (The methods from Section 19.4 could prove helpful.) For the oversampled Nyström method to lead to substantial speedups over XTRACE, it is important to keep the oversampling factor as small as possible, potentially using smaller oversampling factors than considered by Park and Nakatsukasa [PN25]. I also believe there could be new approaches to low-rank approximation of Hermitian indefinite matrices that could be more natural and powerful than the truncated Nyström method.

20.3 Open problem: Numerically stable downdating for subspace iteration

In Section 18.2, we developed an efficient implementation of the leave-one-out error estimator for the randomized SVD (without subspace iteration). This method immediately yields error estimates for randomized subspace iteration (Section 2.4);

Program 20.2 `rsi_errest.m`. Randomized subspace iteration for producing a low-rank approximation to a general matrix with leave-one-out error estimation. Subroutines `cnormc` and `diagprod` are provided in Program F.1 and Program 14.1.

```matlab
function [U,D,V,est] = rsi_errest(B,Bt,n,k,q)
% Input:  Functions B() and Bt() computing matrix products B(X) =
%         B*X and Bt(X) = B'*X, number of columns n, rank k, and
%         number of subspace iteration steps q
% Output: Low-rank approximation Bhat to B, presented as an
%         economy-size SVD Bhat = U*S*V' and error estimate est

assert(mod(q,2) == 0);      % Only implemented for even steps
Om = randn(n,k);            % Gaussian random test matrix with n rows

Z = B(Om); Y = Z;           % First subspace iteration step
for i = 1:(q/2-1)
    Y = B(Bt(Y));           % Two subspace iteration steps
end

[Q,R] = qr(Y,"econ");       % Orthogonalize
C = Bt(Q);                  % Matvecs with B'
[UU,D,V] = svd(C',"econ");  % SVD of factor matrix
U = Q*UU;                   % Get left singular vectors

S = cnormc(inv(R'));        % Downdating matrix
QtZ = Q'*Z;
est = norm(Z-Q*QtZ+Q*S.*diagprod(S,QtZ).',"fro")/sqrt(k);

end
```

see Program 20.2 for an implementation.

Figure 20.2 charts performance of randomized subspace iteration and its error estimate as function of the subspace iteration count q. We see that the leave-one-out error estimator closely tracks the true value of the error across all values of q. However, for $q \geq 6$, the error (and error estimates) for randomized subspace iteration actually *increase* with the number of subspace iteration steps q. This behavior arise from numerical issues. As shown in Fig. 20.2, these numerical issues are easily cured by using intermediate orthogonalization, discussed in Section 2.4.

This brings us to the open problem. At present, it is not known whether there exists a *numerically stable* way of downdating the randomized approximation computed via orthogonalized subspace iteration. As such, the leave-one-out error estimation technique is currently limited to randomized subspace iteration without intermediate
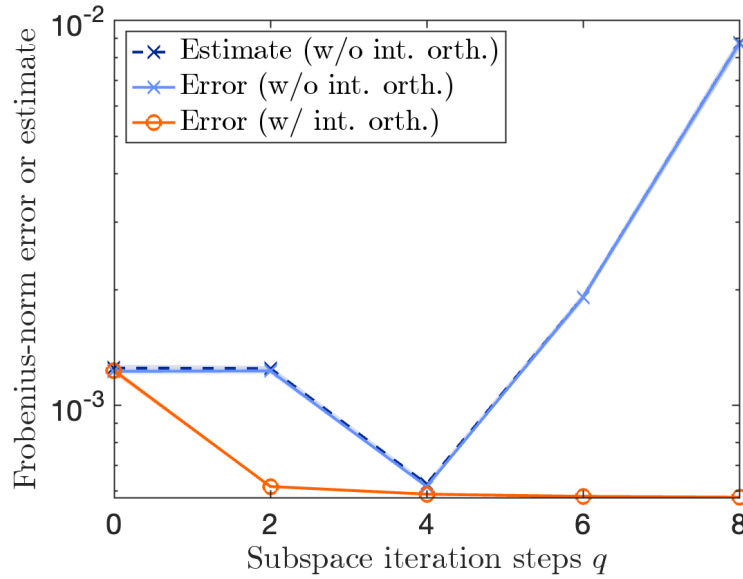
Figure 20.2: Error for randomized SVD with subspace iteration with (solid orange circles) and without (solid light blue crosses) intermediate reorthogonalization for rank-100 approximation of the `poly` matrix (14.27b) as a function of the subspace iteration count q. leave-one-out error estimate (without intermediate reorthogonalization) is shown as dashed dark blue crosses. Solid lines trace the median of 100 trials, and shaded regions show 10% and 90% quantiles.

reorthogonalization. The open problem is to remove this limitation by extending leave-one-out randomized matrix algorithms to be compatible with subspace iteration. There could be other benefits to stable downdating approaches for randomized subspace iteration, including numerically stable versions of XNysTrace incorporating subspace iteration.

Part III

Sketching, solvers, and stability

Dedicated to my grandparents Bob and Sarah Epperly, Kathie and Bill Quarles, and John and Christine Weidner.

Chapter 21

ALGORITHMS FOR LEAST SQUARES, A BRIEF HISTORY

> *But recent research led to new uses of randomization: random mixing and random sampling, which can be combined to form random projections. These ideas have been explored theoretically and have found use in some specialized applications (e.g., data mining), but they have had little influence so far on mainstream numerical linear algebra. Our paper answers a simple question, Can these new techniques beat state-of-the-art numerical linear algebra libraries in practice? Through careful engineering of a new least-squares solver, which we call Blendenpik, and through extensive analysis and experimentation, we have been able to answer this question, yes.*
>
> Haim Avron, Petar Maymounkov, and Sivan Toldeo, *Blendenpik: Supercharging LAPACK's least-squares solver* [AMT10, §1]

The third part of this thesis will investigate randomized algorithms for linear least-squares fitting and related problems. Our core algorithmic tool will be (linear) *randomized dimensionality reduction*, also known as *sketching*. This introductory chapter will set the stage, introducing the linear least-squares problem and algorithms for solving it from a historical perspective. Our historical journey will be brief emphasizing topics that will inform the technical development in subsequent chapters.

Sources. This is an introductory chapter, not based on any particular research article. The reference [BMR22] provides a definitive history of numerical linear algebra from antiquity to the first years of the twenty-first century. Our tale will hew closely to their account for the history of deterministic methods for least squares. In the past twenty-five years, randomized methods have greatly expanded the class of algorithms for least squares. My understanding of the history of these methods is informed by my reading of the original literature, surveys [Woo14; MT20; MDME+23], and discussions with the researchers involved.

Outline. We begin our tour in Section 21.1 with the origins of linear least squares and its solution by the normal equations. We encounter the numerical instabilities of

this approach in Section 21.2 and Golub's QR factorization as a stable alternative in Section 21.3. But how accurate is the QR factorization method? This question is answered in Section 21.4, which reviews perturbation theory, and Section 21.5, which describes forward and backward stability. For the largest problems, direct methods like QR factorization are prohibitively slow, and we must turn to iterative methods, discussed in Section 21.6. Randomization enters in Section 21.7, which describes the sketch-and-solve method as a quick way to obtain an approximate least-squares solution. To achieve higher accuracy, one can use randomization to precondition an iterative method, resulting in the sketch-and-precondition method (Section 21.9). This chapter concludes in Section 21.11 with questions about the numerical stability of sketch-and-precondition raised in the recent paper [MNTW24]. These concerns will be resolved in Chapter 22, which describes fast, numerically stable randomized least-squares solvers.

21.1 The overdetermined linear least-squares problem

The overdetermined linear least-squares problem

$$x = \operatorname*{argmin}_{z \in \mathbb{K}^n} \|c - Bz\| \tag{21.1}$$

is ubiquitous in modern science, engineering, mathematics, and computer science. Throughout this part of the thesis, $B \in \mathbb{K}^{m \times n}$ denotes a matrix of coefficients, $c \in \mathbb{K}^m$ is the right-hand side, and $x = B^\dagger c \in \mathbb{K}^n$ denotes *optimal* least-squares solution (i.e., the solution to (21.1)). Approximate minimizers will be called z or $\widehat{x}$. We will typically assume that B is full-rank, so that (21.1) has a unique solution.

Legendre published the first description of the method of least squares in 1805 [BMR22, p. 57]. Legendre was motivated by problems in geodesy, and he used the method of least squares to estimate the length of the Paris meridian between Dunkirk and Barcelona. In this early work, Legendre took a "bare hands" approach, writing down an expression for the total squared deviation and finding a linear system for the least-squares system by computing the partial derivatives and setting them to zero.

We can repeat Legendre's approach, aided by modern matrix notation (for now, focusing on the *real* case $\mathbb{K} = \mathbb{R}$.) The gradient of the least-squares objective function (21.1) is

$$\nabla_z \|c - Bz\|^2 = 2B^*(c - Bz).$$

Setting this derivative to zero, and rearranging, we obtain the *normal equations* for the least-squares problem:

$$(B^*B)x = B^*c. \tag{21.2}$$

Provided that B has full rank, the normal equations comprise a square, consistent system of linear equations, and the unique least-squares solution is $x = (B^*B)^{-1}(B^*c)$. A similar computation with Wirtinger derivatives establishes the normal equations in the complex case ($\mathbb{K} = \mathbb{C}$).

Legendre's claim to the discovery of the method of least squares was disputed by Gauss, who first published on least squares in 1811. (He claimed to have discovered the method in unpublished work in 1795 [BMR22, pp. 58–60]..) Gauss was motivated by problems in astronomy, and he successfully used the method of least squares to localize the orbit of Ceres. Gauss's work expanded on Legendre's by establishing a connection between least squares, statistics, and the "normal" distribution that now bears Gauss's name. Unfortunately, the disagreement between Legendre and Gauss was not cordial. To quote from [BMR22, p. 60],

> [Gauss] claimed that he already used his method [of least squares] (Unser Prinzip) in 1795 but that its publication was delayed by the Napoleonic wars. Although his name was mentioned by Gauss, Legendre was badly offended. In a letter to Gauss dated May 31, 1809, he rightfully stated that priority is only established by publication. Gauss did not answer him.

21.2   Numerical instabilities of the normal equations

As the nineteenth century became the twentieth, the generally accepted practice for solving a linear least-squares problems were to solve "Gauss' normal equations" [van74, p. 17] by the "Choleski square-root method" [Hou58, p. 339]. In the 1960s, issues of *numerical stability* associated with this approach became wildly known; Golub's paper [Gol65, §1] provides a crisp example (see also [Läu61]).

How accurately can we solve systems of linear equations? In order to understand the numerical instabilities associated with solving the normal equations, we must first understand how accurately we can solve linear systems of equations. We provide a brief review of numerical stability for linear systems, a subject pioneered by von Neumann and Goldstine [vG47] and Turing [Tur48].

To solve a system of linear equations

$$Mx = f \tag{21.3}$$

on a digital computer requires the real or complex entries of the matrix M and the vector f to be stored using finite-precision representations. Most modern computing platforms use *floating-point representations* of numbers; see [Hig02, §2] or [Ove01] for an introduction to floating-point numbers.

The accuracy of a system of floating-point numbers is characterized by the *unit roundoff u*, which captures the magnitude of errors in representing real numbers using floating-point representations and the scale of *rounding errors* incurred when arithmetic operations are performed. The unit roundoff is $u \approx 10^{-16}$ in double-precision arithmetic and $u \approx 10^{-8}$ in single precision. The precision of a floating-point number system is also sometimes characterized using the *machine epsilon*, which is twice the unit roundoff u.

Even *storing* the matrix M incurs errors on the order of the unit roundoff:

$$M_{\text{stored}} = M + E \quad \text{for } E \lessapprox \|M\|u.$$

Solving the linear system (21.3) with the stored matrix M_{stored} results in an approximate solution

$$\widehat{x} = M_{\text{stored}}^{-1} f \approx M^{-1} f - M^{-1} E M^{-1} f.$$

Here, we expanded the inverse $(M + E)^{-1} \approx M^{-1} - M^{-1} E M^{-1}$ to first order in the perturbation E. Therefore, the (relative) *forward error* is roughly of size

$$\frac{\|\widehat{x} - M^{-1}f\|}{\|M^{-1}f\|} \approx \frac{\|M^{-1}EM^{-1}f\|}{\|M^{-1}f\|} \leq \|M^{-1}\| \cdot \|E\| \approx \|M^{-1}\|\|M\|u =: \text{cond}(M)u.$$

We see that the *relative* forward error in solving the linear system of equations (21.3) using the stored floating-point representation of M is on the order of the *condition number* $\text{cond}(M) := \|M^{-1}\|\|M\|$ times the unit roundoff.

This conclusion is striking. Even the act of *storing the matrix M* has already introduced errors in the computed solution $\widehat{x}$ of size $\text{cond}(M)u$ [GV13, §2.7.11]. Of course, solving a linear system of equations requires many arithmetic operations, each of which introduces additional *rounding errors*. Fortunately, it can be shown that—even in the presence of such rounding errors—our heuristic calculation describes the correct magnitude of the numerical errors produced in solving $Mx = f$ with stable methods such as Cholesky or QR factorization:

> Numerically stable algorithms for solving a linear system $Mx = f$ produce a solution that is accurate up to a relative forward error of roughly $\text{cond}(M)u$.

See [Hig02, Chs. 7–10] for a rigorous treatment of perturbation theory and stability analysis for the solution of linear systems of equations.

Numerical instabilities arising from the normal equations. From our discussion of linear systems, the numerical issues with solving the normal equations (21.2) are now apparent. By the boxed maxim above, we would expect that solving the normal equations produces a solution with a relative error of size roughly $\text{cond}(B^*B)u$. Defining the condition number of a rectangular matrix as

$$\text{cond}(B) := \|B\|\|B^\dagger\| = \frac{\sigma_{\max}(B)}{\sigma_{\min}(B)},$$

we see that the condition number of B^*B is the *square* of the condition number of B:

$$\text{cond}(B^*B) = \text{cond}(B)^2.$$

Thus, we obtain the following rule-of-thumb for solution of least-squares problems by the normal equations:

> Solutions to the least-squares problem (21.1) that *explicitly* form the normal equations (21.2) yield relative forward errors of roughly $\text{cond}(B)^2 u$.

This problem of *squaring the condition number* is the main deficiency with solving least-squares problems by forming the normal equations. Golub's paper [Gol65, §1] identifies a related potential problem, called *numerical rank deficiency*, where the *numerically computed B^*B matrix* becomes rank-deficient.

21.3   Numerically stable algorithms by QR factorization

In addition to elucidating the numerical instabilities of the normal equations, Golub's paper [Gol65] also identifies a solution: QR factorization.

Recall that a QR factorization of a matrix B refers to one of two factorizations. The *full QR factorization* takes the form

$$B = Q_{\text{full}} \begin{bmatrix} R \\ 0 \end{bmatrix}$$

where $Q_{\text{full}} \in \mathbb{K}^{m \times m}$ is unitary and $R \in \mathbb{K}^{n \times n}$ is upper triangular. The *thin* or *economy-size* QR *factorization* is

$$B = QR,$$

where $Q \in \mathbb{K}^{m \times n}$ has orthonormal columns and $R \in \mathbb{K}^{n \times n}$ is upper triangular. We emphasize that each of these representations constitutes an *exact* decomposition of the matrix B, in contrast to the partial QR decompositions discussed in Chapter 9. A full QR decomposition can be converted to an economy-size QR decomposition by extracting the first n columns of Q_{full}; that is, $Q := Q_{\text{full}}(:, 1 : n)$. Unless otherwise stated, QR decompositions in this thesis are economy-sized.

The QR factorization for a *square* matrix and stable algorithms for it them was developed by Givens in the 1950s for use in computations at Oak Ridge National Laboratory. (The technical report [Giv54] describes the use of "Givens rotations" in eigenvalue computations.) In 1958, Householder developed a faster and equally stable approach using relection matrices now known as Householder reflectors [Hou58]. Golub's insight was to extend Householder's method to compute QR factorizations of a tall rectangular matrix B. From there, the unique solution x to a full-rank least-squares problem (21.1) is readily obtained as

$$x = R^{-1}(Q^*b).$$

Golub's procedure has dramatically better numerical stability properties than direct solution of the normal equations. With appropriate implementation, it runs in $O(mn^2)$ operations.

21.4   Perturbation theory for least-squares

Had QR factorization truly defeated the normal equations fundamental flaw, *squaring the condition number*? Just a year after Golub's paper [Gol65] was published, Golub and Wilkinson provided a somewhat disappointing answer [GW66] by proving the following perturbation theorem:

Fact 21.1 (First-order perturbation theory for least squares)**.** *Consider the least-squares problem* (21.1)*. Let $\varepsilon > 0$, and consider perturbations ΔB and Δc of magnitude*

$$\|\Delta B\| \leq \varepsilon \|B\|, \quad \|\Delta c\| \leq \varepsilon \|c\|.$$

Then, the solution of the perturbed least-squares problem

$$\widehat{x} = \underset{z \in \mathbb{K}^n}{\operatorname{argmin}} \|(c + \Delta c) - (B + \Delta B)z\|^2,$$

satisfies

$$\|\widehat{x} - x\| \le \text{cond}(B)\varepsilon \cdot \left(\frac{\|c\|}{\|B\|} + \|x\| \right) + \text{cond}(B)^2\varepsilon \cdot \frac{\|c - Bx\|}{\|B\|} + O(\varepsilon^2).$$

Moreover, this bound is approximately attainable.

Golub and Wilkinson summarize the message of their theorem, writing [GW66, p. 144]

> We conclude that although the use of the orthogonal transformation [i.e, QR factorization] avoids some of the ill effects inherent in the use of the normal equations the value of $\text{cond}(B)^2$ is still relevant to some extent.

(We have amended the quote to use the present notation.) In particular, Golub and Wilkinson's analysis implies that, for a good least-squares solver, the (forward) error should scale like $\text{cond}(B)u$ when the residual is small and like $\text{cond}(B)^2u$ when the residual is large.

Golub and Wilkinson's first-order perturbation theorem (Fact 21.1) describes the essential nature of the sensitivity of the least-squares problems to small perturbations in the inputs. Genuine perturbation *bounds* (rather than first-order *estimates*) were developed later by Wedin in his 1973 paper [Wed73]; see [Hig02, Thm. 20.1] for a clean modern statement and a proof. Here is a simplication of the (Golub–Wilkinson–)Wedin result that I have used in my work [Epp24b; EMN25]:

Fact 21.2 (Perturbation bounds for least squares)**.** *Consider a perturbed least-squares problem* (21.1)*:*

$$\widehat{x} = \underset{z \in \mathbb{K}^n}{\text{argmin}} \|(c + \Delta c) - (B + \Delta B)\widehat{x}\| \quad \text{with } \|\Delta B\| \le \varepsilon\|B\|, \ \|\Delta c\| \le \varepsilon\|c\|.$$

Then, provided that $\text{cond}(B)\varepsilon \le 0.1$*, the following bounds hold.*

$$\|x - \widehat{x}\| \le 2.23 \ \text{cond}(B) \left(\|x\| + \text{cond}(B)\frac{\|c - Bx\|}{\|B\|} \right) \varepsilon, \qquad (21.4)$$

$$\|B(x - \widehat{x})\| \le 2.23 \left(\|B\|\|x\| + \text{cond}(B)\|c - Bx\| \right) \varepsilon. \qquad (21.5)$$

A useful feature of this result is the bound on the error $\|B(x - \widehat{x})\|$. I call this quantity the *residual error*. It is the norm distance between the computed and the true residual

$$\|B(x - \widehat{x})\| = \|(c - Bx) - (c - B\widehat{x})\|.$$

In addition, the residual norm of the solution $\widehat{x}$ admits the *Pythagorean decomposition*

$$\|c - B\widehat{x}\|^2 = \|c - Bx\|^2 + \|B(x - \widehat{x})\|^2. \tag{21.6}$$

Observe that the bound (21.4) on the *forward error* $\|x - \widehat{x}\|$ follows immediately from the bound (21.5) on the residual error, together with the identity

$$\|y\| \leq \frac{\|By\|}{\sigma_{\min}(B)} = \operatorname{cond}(B) \cdot \frac{\|By\|}{\|B\|} \quad \text{for every } y \in \mathbb{K}^n. \tag{21.7}$$

21.5 Forward and backward stability

Let us take a break from our tour through history and stop to ask: What does it mean to solve a least-squares problem accurately? These questions have been lurking in the background of our discussion. Now, we address them head-on.

In Section 21.2, we saw that the mere act of *storing* a matrix already introduces perturbations on the order of the unit roundoff u. Since *backward perturbations* (i.e., perturbations to the inputs B and c) are inevitable, it is natural to suggest that a least-squares problem has been solved accurately if the computed solution $\widehat{x}$ is the *exact* solution to a corrupted version of the least-squares problem

$$\widehat{x} = \operatorname*{argmin}_{z \in \mathbb{K}^n} \|(c + \Delta c) - (B + \Delta B)z\|^2. \tag{21.8}$$

Here, the backward perturbations Δc and ΔB should be of norm "roughly" $O(u)$. Specifically, we make the following definition:

Definition 21.3 (Backward stable). A vector $\widehat{x} \in \mathbb{K}^n$ is a *backward stable* solution to the least-squares problem (21.1) if it is the *exact* solution to a modified least-squares problem (21.8), where the perturbations admit bounds $\|\Delta c\| \lesssim \|c\| u$ and $\|\Delta B\| \lesssim \|B\| u$. A least-squares algorithm is *backward stable* if it produces backward stable solutions on every input matrix B that is *numerically full-rank*; that is, $\operatorname{cond}(B)u \ll 1$.

Informally, we use the notation $\alpha \lesssim \beta$ to indicate that α is at most $\gamma\beta$ for some "modest" prefactor γ. Since we are considering numerical algorithms on $m \times n$ matrices, we will allow the prefactor to depend *polynomially* on the parameters m and n. Many authors also stipulate that the prefactor γ be "modest" in the sense that it is a low-degree polynomial in m and n with small constants. We will not put such a requirement our formal definition of stability, though a method with a prefactor $\gamma = 10^{100}$ is certainly not stable in any practically useful sense. We defer a formal description of the notations $\lesssim$ and $\ll$ to Section 22.1.

Backward stability is generally considered to be the gold standard stability property for a numerical algorithm (with one small caveat, discussed in Remark 21.5). The (Householder) QR factorization method of Golub is backward stable [Hig02, Thm. 20.3].

The distance $\|\widehat{x} - x\|$ from the computed solution $\widehat{x}$ to the true solution x is called the *forward error*, and the relative forward error is $\|\widehat{x} - x\|/\|x\|$. If $\widehat{x}$ is computed by a backward stable method, the size of the forward error (and the residual error $\|B(x - \widehat{x})\|$) is controlled by the Golub–Wilkinson–Wedin theorem (Fact 21.2):

$$\|x - \widehat{x}\| \lesssim \text{cond}(B) \left(\|x\| + \text{cond}(B)\frac{\|c - Bx\|}{\|B\|} \right) u; \qquad (21.9)$$

$$\|B(x - \widehat{x})\| \lesssim (\|B\|\|x\| + \text{cond}(B)\|c - Bx\|) \, u. \qquad (21.10)$$

As we will see later in this chapter, there are many interesting least-squares algorithms that satisfy weaker notions of numerical stability than backward stability. In particular, we have the concept of *forward stability*:

Definition 21.4 ((Strong) forward stability). An approximate least-squares solution is $\widehat{x} \in \mathbb{K}^n$ is *forward stable* if it satisfies (21.9) and *strongly forward stable* if it satisfies (21.10). Similarly, a least-squares algorithm is (strongly) forward stable if it always produces a (strongly) forward stable solution whenever B is numerically full-rank; that is, $\text{cond}(B)u \ll 1$.

Backward stability implies strong forward stability implies forward stability, but the reverse implications do not hold. We will encounter methods that are strongly forward stable but not backward stable in Sections 22.2 and 22.4.

Remark 21.5 (Columnwise stability). The notion of backward stability (Definition 21.3) is more precisely called *normwise backward stability*. It states that the computed solution to a least-squares problem is the exact solution to a perturbed least-squares problem (21.8) that is close to the original least-squares problem (21.1) *in norm*. The Householder QR algorithm also satisfies a stronger stability property called *columnwise backward stable* [Hig02, Thm. 20.3], where the perturbation ΔB satisfies

$$\|\Delta b_i\| \lesssim \|b_i\|u \quad \text{for each } i = 1, \dots, n.$$

The distinction between normwise and columnwise backward stability can be important for least-squares problems in which the column norms b_i span multiple orders of magnitude.

Fortunately, any normwise backward stable least-squares solver can be converted into a columnwise backward stable solver. First, preprocess the matrix B by scaling its columns to have unit norm. Then solve the least-squares problem using a normwise stable solver. Finally, post-processing the computed least-squares solution x by counter-scaling its entries. The columnwise backward stable of this approach follows from the theorems of van der Sluis [van69]. The conclusion is that, by using column scaling, the distinction between normwise and columnwise stability can be elided. For the rest of this thesis, we will only refer to normwise concepts of stability. ⋄

21.6 Krylov iterative methods: CGNE, CGLS, and LSQR

In the decades after Golub introduced the Householder QR factorization method for solving least-squares problems, the fundamental paradigm for the solution of least-squares problem remained largely unchanged. If one requested the solution a least-squares problem in a programming environment such as MATLAB, it would be computed by Householder way of QR factorization. Certainly, there were improvements—researchers developed fast blocked algorithms for Householder QR [Lan98], applied pivoting to solve sparse [GN86] and (nearly) rank-deficient [BG65] and packaged high-quality QR implementations were into optimized software packages like LAPACK [ABBB+99]—but the fundamental structure "QR factorize then solve" remained in place.

While the period 1966–2000 saw relatively little development in direct, high-accuracy least-squares algorithms, researchers began exploring iterative methods for least-squares problems, most notably *Krylov subspace methods*. These methods produce approximate solutions to the least-squares problem (21.1) via a sequence of matrix–vector products $z \mapsto Bz$ and $v \mapsto B^*v$. The first methods, variously known as "conjugate gradient least squares" (CGLS) or "conjugate gradient normal equations" (CGNE), worked by applying the conjugate gradient method [HS52] to the normal equations (21.2). Paige and Saunder provided improvement with their LSQR method [PS82]. LSQR is based on the Golub–Kahan process for bidiago-nalizing a matrix [GV13, §10.4]; it produces the same output as CGLS and CGNE in exact arithmetic, but it has improved stability properties in finite precision. Code for LSQR is provided in Program 21.1. Research on Krylov solvers for least-squares problems continued into the 2010s, producing the LSMR [FS11] and LSMB [HG18] algorithms.

Program 21.1 `mylsqr.m`. Implementation of LSQR for solving least-squares problems, iteratively.

```
function X = mylsqr(B,Bt,c,x0,iter)
% Input:  Functions B() and Bt() computing matrix products B(z) =
%           B*z and Bt(z) = B'*z, right-hand side c, initialization x0,
%           number of iterations iter
% Output: Approximate least-squares solutions stacked columnwise as X

% First step of Golub-Kahan bidiagonalization
r = c - B(x0); beta = norm(r); u = r / beta;
v = Bt(u); alpha = norm(v); v = v / alpha;
w = v; phibar = beta; rhobar = alpha;

X = zeros(size(v,1),iter+1); X(:,1) = x0; % Initialize history
for i = 1:iter
    % Golub-Kahan bidiagonalization
    u = B(v) - alpha*u; beta = norm(u); u = u / beta;
    v = Bt(u) - beta*v; alpha = norm(v); v = v / alpha;
    rho = sqrt(rhobar^2 + beta^2);
    c = rhobar / rho;
    s = beta / rho;
    theta = s * alpha;
    rhobar = - c * alpha;
    phi = c * phibar;
    phibar = s * phibar;

    % Update solution, search direction
    X(:,i+1) = X(:,i) + (phi/rho) * w;
    w = v - (theta/rho) * w;
end

end
```

We have already seen that the *accuracy* of solutions to least-squares problems depends on the condition number $\mathrm{cond}(\boldsymbol{B})$ of the matrix $\boldsymbol{B}$. The conditioning also controls the rate of convergence for iterative methods. In particular, we have the following convergence result for CGNE and LSQR, derived from standard results for conjugate gradient [Saa03, eq. (6.128)]:

Fact 21.6 (CGNE and LSQR convergence). *Let $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$ denote the iterates produced by the CGNE or LSQR algorithm (in* exact arithmetic*). Then*

$$\|\boldsymbol{B}(\boldsymbol{x} - \boldsymbol{x}_k)\| \leq 2 \left(\frac{\mathrm{cond}(\boldsymbol{B}) - 1}{\mathrm{cond}(\boldsymbol{B}) + 1} \right)^k \|\boldsymbol{B}(\boldsymbol{x} - \boldsymbol{x}_0)\| \leq 2 \mathrm{e}^{-k/\mathrm{cond}(\boldsymbol{B})} \|\boldsymbol{B}(\boldsymbol{x} - \boldsymbol{x}_0)\|.$$

This bound shows that CGNE and LSQR take at most $O(\text{cond}(\boldsymbol{B})\log(1/\varepsilon))$ iterations to solve a least-sqaures problem to "ε accuracy" in exact arithmetic. While the bound Fact 21.6 is not always quantitatively sharp, it has some predictive power: Except in rare cases, iterative methods converge slowly for ill-conditioned problems.

Iterative methods for least-squares problems are important tools. In particular, they provide a path to solve large sparse least-squares problems that would be impossible to solve using direct methods. However, by the turn of the century, direct methods based on QR factorization—proposed by Golub to solve least-squares problems in 1965—still remained the most effective algorithms for solving most least-squares problems.

21.7 Randomization enters: The sketch-and-solve method

The late 1990s and early 2000s saw the origins of the modern field of randomized numerical linear [FKV98; PTRV98]. The first researchers in this effort were computer scientists, who saw randomization as a way of speeding up computations for large-scale data analysis.

Drineas, Mahoney, and Muthukrishnan developed the first randomized methods for least-squares in 2006 [DMM06]. They proposed computing an *approximate* least-squares solution by subsampling a collection of rows of $\boldsymbol{B}$ (and entries of $\boldsymbol{c}$), reweighting, then solving the subsampled system. Later that year, Sarlós [Sar06] proposed algorithms that multiply the matrix $\boldsymbol{B}$ and $\boldsymbol{c}$ by a *randomized dimensionality reduction matrix* (also called a *sketching matrix*) and solve the "sketched" least-squares problem. These randomized algorithms offered new strategies for quickly obtaining an approximate solution to a least-squares problem.

In modern language, the algorithms of Drineas et al. and Sarlós are examples of the *sketch-and-solve paradigm*. Sketch-and-solve algorithms for the least-squares problem (21.1) proceed as follows:

1. ***Generate*** a random *sketching matrix* $\boldsymbol{S} \in \mathbb{K}^{m \times d}$ with $n \le d \ll m$ columns.

2. ***Sketch*** the data $\boldsymbol{B}$ and $\boldsymbol{c}$ by computing the matrix products $\boldsymbol{S}^*\boldsymbol{B}$ and $\boldsymbol{S}^*\boldsymbol{c}$.

3. ***Solve*** the reduced least-squares problem

$$\widehat{\boldsymbol{x}} = \operatorname*{argmin}_{\boldsymbol{z} \in \mathbb{K}^n} \|(\boldsymbol{S}^*\boldsymbol{B})\boldsymbol{z} - (\boldsymbol{S}^*\boldsymbol{c})\|^2. \qquad (21.11)$$

An implementation appears in Program 21.2.

Program 21.2 `sketch_solve.m`. Sketch-and-solve method for solving overdetermined linear least-squares problems.

```
function x = sketch_solve(B,c,S)
% Input:  Matrix B, right-hand side c, and sketching matrix S
% Output: Approximate least-squares solution x

x = (S'*B) \ (S'*c);

end
```

21.8 Which sketch should I use? The subspace embedding property

Our discussion of sketch-and-solve raises two questions: "Which random matrix S should we pick as the sketching matrix?" and "How accurate is the accurate is the sketch-and-solve solution?" To answer both of these questions, we appeal to the concept of a *subspace embedding*, which is essentially due to Sarlós [Sar06].

Definition 21.7 (Subspace embedding). A matrix $S \in \mathbb{K}^{m \times d}$ is called a *subspace embedding* for a matrix $F \in \mathbb{K}^{m \times n}$ with *distortion* $\eta \in (0, 1)$ when

$$(1 - \eta)\|Fz\| \leq \|(S^*F)z\| \leq (1 + \eta)\|Fz\| \quad \text{for every } z \in \mathbb{K}^n. \quad (21.12)$$

A sketching matrix S is said to be an *oblivious subspace embedding* with *dimension* n, distortion η, and *failure probability* $\delta \in [0, 1)$ if, for every matrix $F \in \mathbb{K}^{m \times n}$, the condition (21.12) holds with probability at least $1 - \delta$.

A subspace embedding maps a high-dimensional vector $v \in \mathbb{K}^m$ to a low-dimensional vector $S^*v \in \mathbb{K}^d$ while *simultaneously* preserves the norm of each vector $v \in \text{range}(F)$. The range of the F is the titular subspace, for which S is an embedding. The following theorem characterizes the accuracy of the sketch-and-solve method when implemented with a subspace embedding:

Theorem 21.8 (Sketch-and-solve). *Let* $S \in \mathbb{K}^{d \times m}$ *be a subspace embedding* for $[B \quad c]$ *of distortion* η, *and let* $\widehat{x}$ *be the sketch-and-solve solution* (21.11). *Then*

$$\|c - B\widehat{x}\| \leq \min\left\{\left(1 + \frac{9\eta^2}{(1-\eta)^4}\right)^{1/2}, \frac{1+\eta}{1-\eta}\right\} \cdot \|c - Bx\|$$

$$= (1 + O(\eta^2))\|c - Bx\|.$$

The theory for sketch-and-solve is surprisingly nuanced; see Appendix C for an introduction this theory, a proof of this result, and a discussion of history.

$$S^* = $$

0	0	0	+½	-½	0	0	0	+½	0	-½	0	+½	0	+½
+½	-½	+½	0	-½	0	-½	0	0	0	0	+½	0	-½	0
0	0	-½	-½	0	-½	0	0	0	+½	0	0	-½	0	0
0	0	+½	0	0	0	-½	0	-½	0	+½	0	-½	-½	0
+½	-½	0	0	+½	0	+½	+½	0	0	0	-½	0	-½	0
0	0	0	-½	0	+½	0	0	0	+½	0	+½	0	0	+½
0	0	0	0	0	+½	0	-½	0	+½	+½	0	0	0	-½
-½	-½	0	-½	0	+½	-½	0	-½	0	+½	0	0	0	+½
0	-½	+½	0	0	0	0	+½	-½	0	0	-½	0	-½	0
+½	0	0	0	+½	0	0	+½	0	+½	0	0	-½	0	0

Figure 21.1: Example of a sparse sign embedding, illustrated with $\zeta = 4$, $m = 15$, and $d = 10$. Observe that there are precisely ζ nonzero entries per *row* of S or *column* of S^*.

Theorem 21.8 provides guarantees for the that the sketch-and-solve method when implemented with any subspace embedding. The literature contains many constructions of subspace embeddings (see [MT20, §§8–9] and [MDME+23, Chs. 2, 6, & 7]), and debates about which construction to use continue to this day. Appendix B weighs in on this debate by providing detailed discussion of the most popular subspace embedding constructions and presenting numerical comparisons. Based on the results documented in this appendix, I recommend *sparse sign embeddings* as the subspace embedding of choice for most applications:

Definition 21.9 (Sparse sign embedding). A *sparse sign embedding* $S \in \mathbb{R}^{m \times d}$ with *sparsity* ζ is a random matrix constructed as follows. Each row is independent and possesses exactly ζ nonzero entries. The nonzero entries are placed in uniformly random positions (selected *without replacement*) and have uniform $\pm \zeta^{-1/2}$ values.

As we document in Appendix B, sparse-sign embeddings have a nearly ideal distortion parameter η and support a fast product operation $B \mapsto S^* B$, owing to sparsity. See Fig. 21.1 for a cartoon illustration of the sparse sign embedding construction.

The best-known analysis fo *sparse* oblivious subspace embeddings due to Cohen [Coh16, Thm. 4.2]:

Fact 21.10 (Sparse sign embeddings). *Fix parameters $n \geq 1$, $\eta \in (0, 1)$, and $\delta \in (0, 1)$. With embedding dimension $d = O(n \log(n/\delta)/\eta^2)$ and sparsity $\zeta = O(\log(n/\delta)/\eta)$, a sparse sign embedding $S \in \mathbb{R}^{m \times d}$ is an oblivious subspace embedding with dimension n, distortion η, and failure probability δ. Since S*

is sparse, matrix–vector products S^*v can be computed in $O(n \log(n/\delta)/\eta)$ operations.

Evidence provided in Appendix B suggests that sparse sign embeddings are reliable with more aggressive parameter choices (e.g., $d = 2n$ and $\zeta = 8$). Providing a theoretical explanation for the reliable performance of sparse sketching matrices with smaller values of d and ζ has been a major research effort in recent years; see [CDDR24; CDD25a; Tro25].

As a corollary of Fact 21.10, we get the following runtime guarantees for sketch-and-solve with a sparse sign embedding:

Corollary 21.11 (Sketch-and-solve). *Using a sparse sign embedding with Cohen's parameter settings (Fact 21.10), one can achieve a $(1+\varepsilon)$-approximate least-squares solution*

$$\|c - B\widehat{x}\| \le (1 + \varepsilon)\|c - Bx\|$$

with 99% probability with a sketching matrix of embedding dimension

$$d = O\left(\frac{n \log n}{\varepsilon}\right).$$

The total runtime is

$$O\left(\frac{mn \log n}{\varepsilon^{1/2}} + \frac{n^3 \log n}{\varepsilon}\right). \tag{21.13}$$

Corollary 21.11 shows that, using a sketching matrix like a sparse sign embedding with a fast multiply operation, sketch-and-solve runs in roughly $O(mn + n^3)$ operations. For large, highly overdetermined problems ($m \gg n \gg 1$), this is a large speedup over Householder QR factorization. However, the runtime sketch-and-solve depends inversely on the least-squares suboptimality ε. As such, improving the accuracy of sketch-and-solve is costly, and sketch-and-solve is mainly useful as a way of obtaining a cheap, approximate solution to a least-squares problem for a relatively large value of $\varepsilon \in [0.25, 1]$. Indeed, we have already seen sketch-and-solve type methods used to construct approximate interpolative decompositions in Section 9.4 and CUR decompositions in Section 10.4. (Note that weighted row sampling is a type of sketching.)

21.9 Making randomized least-squares accurate: Sketch-and-precondition

The papers [DMM06; Sar06] demonstrate that randomized algorithms offer a fast way of obtaining an approximate least-squares solutions. Can randomized meth-

ods also solve least-squares problems to high accuracy? Rokhlin and Tygert answered this question by developing the sketch-and-precondition method [RT08]. This method was popularized by Avron, Maymounkov, and Toledo [AMT10], who developed a variant of the algorithm they called *Blendenpik*. The papers [MSM14; CFS21] propose further variants of sketch-and-precondition.

Rokhlin and Tygert's core observation is that the sketched matrix S^*B provides an excellent preconditioner for the least-squares problem (21.1). This preconditioner can be combined with a Krylov iterative method (Section 21.6) such as CGNE or LSQR to solve the least-squares problem rapidly to high precision.

Here is the main technical result for sketch-and-precondition, in modern language.

Proposition 21.12 (Randomized preconditioning). *Let $S \in \mathbb{K}^{d \times m}$ be a subspace embedding for B with distortion $\eta < 1$. Construct the sketched matrix S^*B, and let*

$$S^*B = UM$$

be any factorization S^*B into a matrix $U \in \mathbb{K}^{d \times n}$ with orthonormal columns and a nonsingular matrix M. Then

$$\sigma_{\max}(BM^{-1}) \leq \frac{1}{1-\eta}, \quad \sigma_{\min}(BM^{-1}) \geq \frac{1}{1+\eta}.$$

Consequently,

$$\mathrm{cond}(BM^{-1}) \leq \frac{1+\eta}{1-\eta} = 1 + O(\eta).$$

Proof. We establish the upper bound on $\sigma_{\max}(BM^{-1})$; the lower bound on $\sigma_{\min}(BM^{-1})$ follows on similar lines. By the variational characterization of the largest singular value, we have

$$\sigma_{\max}(BM^{-1}) = \max_{\|v\|=1} \|BM^{-1}v\|.$$

Now, apply the subspace embedding property to bound

$$\sigma_{\max}(BM^{-1}) = \max_{\|v\|=1} \|BM^{-1}v\| \leq \frac{1}{1-\eta} \max_{\|v\|=1} \|S^*BM^{-1}v\|.$$

But $S^*BM^{-1} = U$, so

$$\sigma_{\max}(BM^{-1}) = \max_{\|v\|=1} \|BM^{-1}v\| \leq \frac{1}{1-\eta} \cdot \max_{\|v\|=1} \|S^*BM^{-1}v\|$$

$$= \frac{1}{1-\eta} \cdot \max_{\|v\|=1} \|Uv\| = \frac{1}{1-\eta} \cdot \max_{\|v\|=1} \|v\| = \frac{1}{1-\eta}.$$

In the penultimate equality, we used the fact that U has orthonormal columns, and thus $\|Uv\| = \|v\|$. $\qquad \square$

Provided that S is a subspace embedding with small distortion η, randomized preconditioning brings the condition number of B below an *absolute constant*. For example, $\text{cond}(BM^{-1}) \leq 3$ when $\eta \leq 1/2$. This results shows that randomization produces a preconditioner of *extraordinary* quality. Using this preconditioner with a Krylov method like LSQR or CGNE produces a solution that converges very rapidly (see Corollary 21.13 below). This observation motivates the following (prototype) sketch-and-precondition algorithm:

1. ***Generate*** a sketching matrix $S \in \mathbb{R}^{n \times d}$.

2. ***Sketch*** the matrix B, computing S^*B.

3. ***Form*** an orthonormal decomposition $S^*B = UM$.

4. ***Solve*** the least-squares problem (21.1) using LSQR with preconditioner M.

Implementations details will follow in next section and code is provided in Program 21.3.

As we will see below in Corollary 21.13, with an appropriate implementation, the sketch-and-precondition algorithm runs in at most operations $O(mn \log(m/\varepsilon) + n^3 \log n)$ to produce a $(1+\varepsilon)$-approximate least-squares solution. Since the accuracy parameter ε appears inside the logarithm, we see that the sketch-and-precondition method can achieve high accuracy (machine-precision accuracy, even) in a small number of LSQR steps. Contrast this with the sketch-and-solve method, whose runtime scales with the inverse-accuracy parameter $1/\varepsilon$ and thus cannot achieve high accuracy without an astronomical slowdown. For large, highly over-determined problems ($1 \ll n \ll m$) sketch-and-precondition is asymptotically faster than the QR factorization method, which requires $O(mn^2)$ operations.

I consider the sketch-and-precondition method to be one of the most striking examples of a randomized algorithm for matrix computations. Here, randomness can be used to solve a least-squares problems to any desired accuracy level (in exact arithmetic at least) at a cost that is (asymptotically) faster than the QR factorization method. Forty-five years after its ascendance to the throne, the QR method faces its first serious challenger as the method of choice for solving general dense least-squares problems, at least ones satisfying $1 \ll n \ll m$.

21.10 Implementation and analysis of sketch-and-precondition

The sketch-and-precondition algorithm will be important in the rest of this thesis. In this section, we will discuss implementation choices and present analysis.

Choice of orthonormal decomposition. To put the randomized preconditioning idea into practice, the two natural orthonormal decompositions are the QR decomposition

$$S^* B = QR$$

and the (economy-size) SVD

$$S^* B = UM \quad \text{for } M = \Sigma V^*.$$

In the former case, the inverse-preconditioning operation $R^{-1} z$ can efficiently performed by triangular substitution. In the latter case, $M^{-1} = V \Sigma^{-1}$ has a simple closed form. For reasons that will emerge in Section 22.7, we will use the SVD-based implementation for this thesis.

Initialization. In their original paper, Rokhlin and Tygert suggested initializing via the sketch-and-solve and solve solution

$$x_0 := (S^* B)(S^* c) = M^{-1}(U^*(S^* c)).$$

to initialize the sketc Their motivation for using this initialization was to reduce the number of iteration steps required to converge. As we will see, the choice of initialization also has *numerical stability* implications for the sketch-and-precondition algorithm (Section 22.4).

Interestingly, the matter of initialization is not mentioned in the influential paper [AMT10]. By default, their code uses the trivial, zero initialization $x_0 := 0$, although they do provide the sketch-and-solve initialization as an optional flag.

Choice of iterative method. The sketch-and-precondition algorithm can be implemented using any preconditioned Krylov iterative method. In this thesis, we will use preconditioned LSQR in all our implementations.

In this thesis, we will precondition LSQR "explicitly" by performing the change of variables

$$x = M^{-1} y \tag{21.14}$$

and applying standard LSQR to the preconditioned least-squares problem

$$y = \operatorname*{argmin}_{z \in \mathbb{K}^n} \left\| c - B(M^{-1} z) \right\|. \tag{21.15}$$

Program 21.3 `sketch_precondition.m`. Sketch-and-precondition method for solving overdetermined linear least-squares problems. Subroutine `mylsqr` is provided in Program 21.1.

```
function X = sketch_precondition(B,c,S,iter)
% Input:  Matrix B, right-hand side c, sketching matrix S, and
%         iteration count iter
% Output: Approximate least-squares solution x

    [U,D,V] = svd(S'*B,"econ");       % Sketch and compute SVD
    prec = V / D;                     % Form preconditioner
    y0 = U'*(S'*c);                   % Sketch-and-solve initialize
    Y = mylsqr(@(z) B*(prec*z), ...   % Preconditioned LSQR
               @(z) prec'*(B'*z), ...
               c, y0, iter);
    X = prec * Y;                     % Change of variables

end
```

We emphasize that *the product BM^{-1} should not be formed explicitly*. Instead, its action is achieved by applying the matrices in sequence $z \mapsto B(M^{-1}z)$. Likewise, the adjoint is applied as $v \mapsto M^{-*}(B^*v)$. Once (21.15) has been solved, the solution x to the least-squares problem (21.1) is obtained from the change-of-variables formula (21.14). See also [Mei24, Alg. 8] for an alternate approach to preconditioning LSQR.

Implementation. An implementation of sketch-and-precondition using the above recommendation appears in Program 21.3.

Analysis. By combining Fact 21.6, Theorem 21.8, and Proposition 21.12, we can analyze the sketch-and-precondition method.

Corollary 21.13 (Sketch-and-precondition: Exact arithmetic)**.** *Let $S \in \mathbb{K}^{d \times m}$ be a subspace embedding* for $[B \quad c]$ *with distortion $\eta < 1$. The sketch-and-precondition algorithm (Program 21.3) in* exact arithmetic *satisfies the bound*

$$\|c - Bx_k\|^2 \le \left(1 + \frac{36}{(1-\eta)^4} \cdot \eta^{2+k}\right) \|c - Bx\|^2. \tag{21.16}$$

In particular, using a sparse sign embedding with $\eta = 1/2$ (see Fact 21.10), sketch-and-precondition produces a $(1 + \varepsilon)$-approximate least-squares solution

$$\|c - Bx_k\| \le (1 + \varepsilon)\|x - Bx\|.$$

after $k = O(\log(1/\varepsilon))$ iterations and $O(mn \log(n/\varepsilon) + n^3 \log n)$ operations.
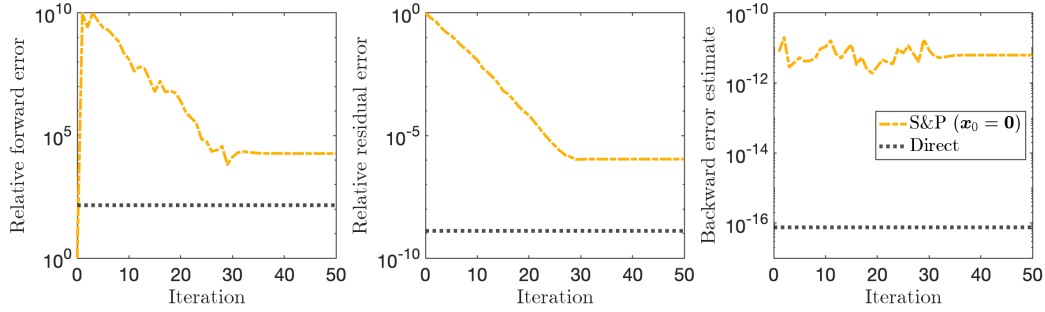
Figure 21.2: Forward (*left*), residual (*middle*), and backward (*right*) error of sketch-and-precondition *with the zero initialization* (single execution). Errors for direct method (MATLAB's `mldivide`) shown for reference.

Proof. By Theorem 21.8,

$$\operatorname{cond}(\boldsymbol{B}\boldsymbol{M}^{-1}) \le \frac{1+\eta}{1-\eta}.$$

Therefore,

$$\frac{\operatorname{cond}(\boldsymbol{B}\boldsymbol{M}^{-1}) - 1}{\operatorname{cond}(\boldsymbol{B}\boldsymbol{M}^{-1}) + 1} \le \eta.$$

Therefore, by the Pythagorean identity (21.6) and the LSQR guarantee (Fact 21.6), we obtain the bound

$$\|\boldsymbol{c} - \boldsymbol{B}\boldsymbol{x}_k\|^2 \le 4\eta^{2k} \|\boldsymbol{B}(\boldsymbol{x} - \boldsymbol{x}_0)\|^2 + \|\boldsymbol{c} - \boldsymbol{B}\boldsymbol{x}\|^2.$$

Finally, invoking the Pythagorean identity (21.6) and the sketch-and-solve guarantee (Theorem 21.8),

$$\|\boldsymbol{B}(\boldsymbol{x} - \boldsymbol{x}_0)\|^2 = \|\boldsymbol{c} - \boldsymbol{B}\boldsymbol{x}_0\|^2 - \|\boldsymbol{c} - \boldsymbol{B}\boldsymbol{x}\|^2 \le \frac{9\eta^2}{(1-\eta)^4} \|\boldsymbol{c} - \boldsymbol{B}\boldsymbol{x}\|^2.$$

Combining the two previous displays, we obtain the sketch-and-precondition guarantee (21.16). The runtime guarantee for producing a $(1+\varepsilon)$-approximate least-squares solution follows from Fact 21.10. □

21.11 Is sketch-and-precondition numerically unstable?

In 2023, Meier, Nakasukasa, Townsend, and Webb released a preprint demonstrating that the version of sketch-and-precondition from the popular paper [AMT10]—that is, the version *with the zero initialization* $\boldsymbol{x}_0 := \boldsymbol{0}$—is numerically unstable [MNTW24].

A variant of their experiment appears in Fig. 21.2. Here, I generated a random least-squares problem of dimensions 4000×50 using Program 21.4 with condition

Program 21.4 `random_ls_problem.m`. Generate a random least-squares problem with prescribed dimensions, condition number, and residual norm. Subroutine `haarorth` is provided in Program F.7.

```
function [B,c,x,r,s,V] = random_ls_problem(m,n,condB,rnorm)
% Input:  Dimensions m and n, condition number condB, and residual
%         norm rnorm
% Output: Matrix B, right-hand side c, solution x = B\c, residual r
%         = c - B*x, singular values s, and right singular vectors V

    U = haarorth(m,n+1);               % Haar-random left sing vecs
    V = haarorth(n,n);                 % Haar-random right sing vecs
    s = logspace(-log10(condB),0,n)';  % Log-spaced sing vals
    B = U(:,1:n)*diag(s)*V';
    x = orth(randn(n,1));              % Unit-vector solution
    r = U(:,end) * rnorm;              % Residual orthog to range(B)
    c = B*x + r;

end
```

number $\texttt{condB} = \mathrm{cond}(B) = 10^{12}$ and residual norm $\texttt{rnorm} = \|c - Bx\| = 10^{-4}$. The sketching matrix S is a sparse sign embedding (Definition 21.9) with dimensions 4000×100 and sparsity level $\zeta = 4$.

The left panel of Fig. 21.2 plots the relative forward error $\|x - \widehat{x}\|/\|x\|$; the middle panel plots the relative residual error $\|B(x - \widehat{x})\|/\|Bx\|$; and the right panel plots the *backward error* $\|\Delta B\|_{\mathrm{F}}/\|B\|_{\mathrm{F}}$, the magnitude of the *minimal* backward perturbation ΔB in (21.8). In fact, we report an *estimate* of this quantity, computed using Program 22.4 described below. As we see, all of these error quantities are well above the error levels for MATLAB's QR-based `mldivide`, shown as a black dotted line. Therefore, we recognize that sketch-and-precondition *with the zero initialization $x_0 := 0$* is numerically unstable, in both the backward and forward senses (see Section 21.5).

The work of Meier et al. raised serious questions about the viability of sketch-and-precondition as a replacement for least-squares solvers based on QR factorization, at least in applications that demand high numerical accuracy. Perhaps QR factorization's place on the throne as the default least-squares solver for general-purpose software remained safe after all.

Fortunately, the instabilities of sketch-and-precondition are curable. In next chapter, we will develop fast, numerically stable randomized least-squares solvers, including

a simple modification of the sketch-and-precondition that is backwards. Using these stable algorithms, we can solve least-squares problems to machine accuracy up to an order of magnitude faster than traditional least-squares solvers based on QR factorization.

Chapter 22

FAST, STABLE RANDOMIZED LEAST-SQUARES SOLVERS

> *For me,* `eig(A)` *epitomizes the successful contribution of numerical analysis to our technological world. Physicists, chemists, engineers, and mathematicians know that computing eigenvalues of matrices is a solved problem. Simply invoke* `eig(A)`*. . . and you tap into the work of generations of numerical analysts. The algorithm involved, the* QR *algorithm, is completely reliable, utterly nonobvious, and amazingly fast.*
>
> Lloyd N. Trefethen, *An Applied Mathematician's Apology* [Tre22, p. 52]

Last chapter, we saw the results of Meier, Nakatsuksa, Townsend, and Webb [MNTW24], who showed that a common implementation of the sketch-and-precondition algorithm is numerically unstable, achieving errors in floating-point arithmetic that are are orders of magnitude higher than standard direct methods. Meier et al.'s work left open the question of whether *any* randomized least-squares solver is numerically stable, even in the forward sense.

This chapter answers this question in the affirmative. We discuss three algorithms: the *sketch-and-descend algorithm* and sketch-and-precondition algorithm *with sketch-and-solve initialization*, which are (strongly) forward stable but not backward stable, and *sketch-and-precondition with iterative refinement* (SPIR), which is stable in both the (strong) forward and backward senses.

Sources. This chapter is based on the papers

Ethan N. Epperly. "Fast and Forward Stable Randomized Algorithms for Linear Least-Squares Problems". In: *SIAM Journal on Matrix Analysis and Applications* (Dec. 2024), pp. 1782–1804. DOI: 10.1137/23M1616790

and

Ethan N. Epperly, Maike Meier, and Yuji Nakatsukasa. "Fast Randomized Least-Squares Solvers Can Be Just as Accurate and Stable as Classical Direct Solvers". In: *Communications on Pure and Applied Mathematics, accepted* (2025). Preprint available at https://arxiv.org/abs/2406.03468v2.

Outline. After some notation in Section 22.1, we discuss the sketch-and-descend, sketch-and-precondition, and SPIR algorithms in Sections 22.2, 22.4 and 22.5. For our stability proofs, we will need to analyze modified versions of sketch-and-precondition using the Lanczos algorithm in place of LSQR. For this reason, our treatment will be briefly interrupted in Section 22.3 with an introduction to the Lanczos algorithm and its relationship to other Krylov methods. Section 22.6 contains experiments, and Section 22.7 investiates estimation of the backward error.

22.1 Notation

Before we go further, let us establish some notation. We are interested in understanding the behavior of iterative least-squares algorithms that operate on $m \times n$ matrices and may require $O(\log(1/u))$ iterations to converge to the maximum numerically achievable accuracy. As such, the total number of arithmetic operations for algorithms in this chapter will be a polynomial in these three parameters: the dimensions m and n and the iteration count $\log(1/u)$. Since each arithmetic operation incurs some rounding error, it is natural to expect that iterative least-squares algorithms should be only accurate up to error $\text{poly}(m, n, \log(1/u))u$. (Here, poly denotes an unspecified polynomial function.)

With this in mind, we write $\alpha \lesssim \beta$ or $\beta \gtrsim \alpha$ if $\alpha \leq \gamma\beta$ for a prefactor $\gamma = \text{poly}(m, n, \log(1/u))$. We write $\alpha \asymp \beta$ if $\alpha \lesssim \beta \lesssim \alpha$. Similarly, we write $\alpha \ll \beta$ if $\alpha \leq \beta/\gamma$ for a *sufficiently large* polynomial factor $\gamma = \text{poly}(m, n, \log(1/u))$. (That is, the factor γ is as large a polynomial function as needed to complete the analysis.) For an algorithm to be stable in a practical sense, we hope the polynomial factors γ are small. As we will see in Chapter 23, the proofs are already complicated when we make no effort to track the prefactors.

We work in the standard model of floating-point arithmetic [Hig02, §2.2]. We write $\text{fl}(z)$ for the value of a quantity z evaluated in floating-point, and we introduce $\text{err}(z) := \text{fl}(z) - z$. When using the fl and err notations, we will always assume that the quantity is computed most natural way, i.e., $AR^{-1}z$ is evaluated as `A*(R\z)`, not `A*(inv(R)*z)` and certainly not `(A*inv(R))*z`.

Throughout, we focus on the least-squares problem (21.1) with matrix B and right-hand side c. The condition number of B will be denoted $\kappa := \text{cond}(B)$.

22.2 Sketch-and-descend: A strongly forward stable least-squares solver

After Meier, Nakatsukasa, Townsend, and Webb demonstrated instabilities in sketch-and-precondition [MNTW24], it was natural to ask whether other least-squares algorithms were numerically stable. (The stabilizing effects of the sketch-and-solve initialization, discussed below in Section 22.4, were not known at the time.) Joel Tropp suggested to me the possibility that *iterative sketching* methods would be numerically stable. As it happens, these methods are strongly forward stable (but not backward stable) when implemented appropriately.

Iterative sketching methods and the sketch-and-descend method

We now describe the family of iterative sketching methods, culminating in a method we call the *sketch-and-descend* method.

Iterative Hessian sketch. In 2016, Pilanci and Wainwright developed an alternative to the sketch-and-precondition paradigm for rapidly solving least-squares problems to high-accuracy. They called their approach *iterative Hessian sketch* [PW16]. Their basic idea is to reformulate the least-squares problem (21.1) as a sum of a quadratic term and a linear term:

$$x = \underset{z \in \mathbb{K}^n}{\operatorname{argmin}} \left[\|Bz\|^2 - 2\mathrm{Re}((B^*c)^*z) \right]. \tag{22.1}$$

Then, reformulate this equation by introducing an offset x_i:

$$x = \underset{z \in \mathbb{K}^n}{\operatorname{argmin}} \left[\|B(z - x_i)\|^2 - 2\mathrm{Re}((B^*[c - Bx_i])^*(z - x_i)) \right]. \tag{22.2}$$

Both (22.1) and (22.2) are exact reformulations of the least-squares problem (21.1), with all three optimization problems sharing the same solution x (which is unique when B has full rank).

Now, we can introduce sketching. Beginning from an initialization $x_0 \in \mathbb{K}^n$, we iterate for $i = 0, 1, \ldots$. In their original algorithm, Pilanci and Wainwright draw a *new sketch* $S_i \in \mathbb{K}^{m \times d}$ at each iteration, and they sketch only the quadratic term, resulting in the iteration

$$x_{i+1} = \underset{z \in \mathbb{K}^n}{\operatorname{argmin}} \left[\|S_i^* B(z - x_i)\|^2 - 2\mathrm{Re}((B^*[c - Bx_i])^*(z - x_i)) \right].$$

Solving this optimization problem, we see that the resulting iteration is

$$x_{i+1} := x_i + [(S_i^* B)^*(S_i^* B)]^{-1} B^*(c - Bx_i).$$

This iteration constitutes Pilanci and Wainwright's iterative Hessian sketch algorithm for the least-squares problem (21.1).

An advantage of the iterative Hessian sketch approach is that it generalizes nicely to more general optimization problems, including *constrained* least-squares problems; see [PW16] for details. For this thesis, we will focus only on the unconstrained least-squares problem (21.1).

From iterative Hessian sketch to sketch-and-descend. In the years following Pilanci and Wainwright's original paper, researchers developed optimizations to the iterative Hessian sketch method for the unconstrained linear least-squares problems (21.1). First, the iteration-dependent sketching matrix S_i was replaced by a single sketching matrix $S \in \mathbb{K}^{m \times d}$ used across all iterations. Second, the procedure was accelerated by incorporating dampling and momentum

$$x_{i+1} \coloneqq x_i + \alpha [(S^*B)^*(S^*B)]^{-1} B^*(c - Bx_i) + \beta(x_i - x_{i-1}). \tag{22.3}$$

When implemented with a subspace embedding of distortion η, the optimal values of the coefficients α and β are

$$\alpha = (1 - \eta^2)^2, \quad \beta = \eta^2. \tag{22.4}$$

These refinements were developed in concurrent papers of Ozaslan, Pilanci, and Arikan [OPA19] and Lacotte and Pilanci [LP23].

In my view, the name iterative Hessian sketch is somewhat of a misnomer for the modern version of the procedure (22.3) for least squares, as only a single sketch is computed. (When I have publicly presented on this work, the names *iterative Hessian sketch* and *iterative sketching* for this procedure caused confusion.) With the optimizations of [OPA19; LP23], the iteration (22.3) can be interpreted as preconditioned gradient descent with heavy-ball momentum for the least-squares objective (21.1). Thus, the procedure (22.3) can be interpreted as a version of sketch-and-precondition where a (momentum-accelerated) gradient method is used in place of a Krylov solver. To avoid potential confusion, we will refer to the iteration (22.3) as the *sketch-and-descend* method for this thesis.

Numerically stable implementation of sketch-and-descend

The original works [PW16; OPA19; LP23] do not offer precise guidance on how to implement the iterative Hessian sketch/sketch-and-descend method. Directly forming and manipulating the matrix $(S^*B)^*(S^*B)$ can lead to numerical issues,

331

analogous to those involved in solving the normal equations (Section 21.2). To address this issues, my paper [Epp24b] proposes implementing the iteration (22.3) by first computing a QR decomposition (or other orthonormal factorization) of the sketched matrix

$$S^*B = QR,$$

and using the update rule

$$x_{i+1} := x_i + \alpha R^{-1}(R^{-*}(B^*(c - Bx_i))) + \beta(x_i - x_{i-1}). \qquad (22.5)$$

The matrices R^{-1} and R^{-*} are applied by triangular substitution, not explicit inversion.

For numerical stability, it is important to evaluate (22.5) in the parenthesized order. The following implementation

$$x_{i+1} := x_i + \alpha R^{-1}(R^{-*}(B^*c - B^*(Bx_i)))) + \beta(x_i - x_{i-1}), \qquad \text{(Bad!)}$$

equivalent to (22.5) in exact arithmetic, can be numerically disastrous.

As a refinement, we can choose the initial iterate x_0 to be the sketch-and-solve solution (Section 21.7). Using this good initialization can reduce the iteration count needed to convergence. Code for the sketch-and-descend method using these improvements is provided in Program 22.1. Evidence that these modifications improve stability appears in [Epp24b, Fig. 3].

Remark 22.1 (Sketch-and-descend versus sketch-and-precondition). The sketch-and-descend method with optimal parameters (22.4) and the sketch-and-precondition method *with sketch-and-solve initialization* (Section 22.4) typically perform quite similarly in practice. As such, I usually recommend the sketch-and-precondition method in practice, as the algorithm has no free parameters. An exception where sketch-and-descend may be preferable is in parallel computing environments [MSM14], because sketch-and-descend requires fewer parallel synchronization steps. ◇

Analysis in exact arithmetic

With an appropriate embedding and suitable parameter choices α, β, the sketch-and-descend method converges geometrically, and the total runtime is $O(mn \log(n/\varepsilon) + n^3 \log n)$, the same as sketch-and-precondition. Indeed, with the optimal parameters α, β, the *rate of convergence* is even the same for sketch-and-descent as for sketch-and-precondition. We have the following result:

Program 22.1 `sketch_descend.m`.   Sketch-and-descend method for solving overdetermined linear least-squares problems.

```matlab
function X = sketch_descend(B,c,S,alpha,beta,iter)
% Input:  Matrix B, right-hand side c, sketching matrix S, damping
%         and momentum coefficients alpha and beta, and iteration
%         count iter
% Output: Approximate least-squares solution x

    X = zeros(size(B,2),iter+1);    % Initialize history
    [Q,R] = qr(S'*B,"econ");        % Sketch and compute QR
    X(:,1) = R\(Q'*(S'*c));         % Sketch-and-solve initialize
    X(:,2) = X(:,1) + alpha * (R\(R'\(B'*(c-B*X(:,1))))); % Step one
    for i = 2:iter
        X(:,i+1) = X(:,i)...
            + alpha * (R\(R'\(B'*(c-B*X(:,i)))))... % Damped gradient
            + beta * (X(:,i) - X(:,i-1));           % Momentum
    end

end
```

Theorem 22.2 (Sketch-and-descend: Exact arithmetic). *Let $S \in \mathbb{K}^{d \times m}$ be a subspace embedding* for $[\boldsymbol{B} \quad \boldsymbol{c}]$ *with distortion $\eta < 1$, and let $\boldsymbol{x}_i$ denote the iterates produced by the sketch-and-descend method (Program 22.1) in exact arithmetic. For the trivial parameter choices $\alpha = 1$ and $\beta = 0$, we have*

$$\|\boldsymbol{c} - \boldsymbol{B}\boldsymbol{x}_k\|^2 \leq \left(1 + 27\eta \cdot [(2 + \sqrt{2})\eta]^{2k}\right) \|\boldsymbol{c} - \boldsymbol{B}\boldsymbol{x}\|^2,$$

provided the convergence rate is less than one: $(2 + \sqrt{2})\eta < 1$. With the optimal parameter choices (22.4), we have the improved bound

$$\|\boldsymbol{c} - \boldsymbol{B}\boldsymbol{x}_k\|^2 \leq \left(1 + \frac{256}{(1 - \eta)^2} \cdot k\eta^{2k-1}\right) \|\boldsymbol{c} - \boldsymbol{B}\boldsymbol{x}\|^2.$$

In particular, when implemented with a sparse sign embedding with Cohen's parameter choices (Fact 21.10), sketch-and-descend produces a $(1 + \varepsilon)$-approximate least-squares solution

$$\|\boldsymbol{c} - \boldsymbol{B}\boldsymbol{x}_k\| \leq (1 + \varepsilon)\|\boldsymbol{x} - \boldsymbol{B}\boldsymbol{x}\|.$$

with 99% probability after $k = \mathcal{O}(\log(1/\varepsilon))$ iterations and $\mathcal{O}(mn \log(n/\varepsilon) + n^3 \log n)$ operations.

The essence of this result appears in the works of Pilanci and coauthors [PW16; OPA19; LP23]. The precise result stated here is an immediate consequence of [Epp24b, Thms. 3.1 & B.1].

Analysis in finite precision arithmetic

In my paper [Epp24b], I proved that sketch-and-descend *with trivial parameter choices* $\alpha = 1$ *and* $\beta = 0$ is strongly forward stable. We have the following result [Epp24b, Thm. 4.1]:

Theorem 22.3 (Sketch-and-descend is strongly forward stable). *Let $S \in \mathbb{K}^{m \times d}$ be a subspace embedding* for $[\boldsymbol{B} \quad \boldsymbol{c}]$ *of distortion* $\eta \leq 0.29$ *with* $d \leq m$, *suppose $\boldsymbol{B}$ is numerically full-rank $\kappa u \ll 1$, and assume the multiply operation is stable $\|\mathrm{err}(S^* \boldsymbol{v})\| \lesssim \|\boldsymbol{v}\| u$. The sketch-and-descend method with trivial parameter choices $\alpha = 1$ and $\beta = 0$ is geometrically convergent until reaching strong forward stability*

$$\|\boldsymbol{B}(\boldsymbol{x} - \mathrm{fl}(\boldsymbol{x}_k))\| \leq 20 \sqrt{\eta} \left[(2 + \sqrt{2})\eta + \gamma \kappa u \right]^k + \gamma \left(\|\boldsymbol{B}\| \|\boldsymbol{x}\| + \kappa \|\boldsymbol{c} - \boldsymbol{B}\boldsymbol{x}\| \right) u. \quad (22.6)$$

Here, the prefactor $\gamma = \mathrm{poly}(m, n)$. In particular, for real data and implemented with a sparse sign embedding with Cohen's parameter settings (Fact 21.10), sketch-and-descend produces a strongly forward stable solution in $O(mn \log(n/u) + n^3 \log n)$ operations.

The statement in [Epp24b] is for real arithmetic, but the proof transfers to complex arithmetic without issue. The bound (22.6) on the residual error leads to a bound on the forward error $\|\boldsymbol{x} - \mathrm{fl}(\boldsymbol{x}_k)\|$ by using the identity (21.7). We will omit proof of this result because our upcoming results for sketch-and-precondition are of greater practical relevance.

22.3 Intermezzo: Lanczos, conjugate gradient, and LSQR

Error analysis of Krylov iterative methods in finite precision arithmetic remains a notoriously difficult and incomplete area of study. Moreover, our theoretical understanding is uneven across different methods: The Lanczos algorithm has been extensively analyzed in finite precision, but virtually nothing is known about LSQR. Consequently, for analytical purposes, it is useful to study a version of sketch-and-precondition based on the Lanczos algorithm.

An introduction the Lanczos algorithm is beyond the scope of this thesis; see Tyler Chen's monograph [Che24]. The Lanczos algorithm is a Krylov subspace method that takes as input a *Hermitian* matrix $\boldsymbol{M} \in \mathbb{K}^{n \times n}$ (or, more precisely, a subroutine implementing matrix–vector products $\boldsymbol{v} \mapsto \boldsymbol{M}\boldsymbol{z}$) and a vector $\boldsymbol{f} \in \mathbb{K}^n$. It produces a *Hermitian* tridiagonal matrix $\boldsymbol{T} \in \mathbb{K}^{k \times k}$ and a matrix $\boldsymbol{Q} \in \mathbb{K}^{n \times k}$ with orthonormal columns such that

Program 22.2 `lanczos.m`. Lanczos algorithm for partially tridiagonalizing a matrix.

```
function [Q,T] = lanczos(M,f,k)
% Input:  Function M() computing matrix-vector products, vector f,
%         and number of steps k
% Output: Matrix Q with orthonormal columns and tridiagonal matrix T

Q = zeros(length(f),k+1); T = zeros(k+1);        % Initialize
Q(:,1) = f / norm(f);
for i = 1:k
    if i == 1
        Q(:,2) = M(Q(:,1));
    else
        Q(:,i+1) = M(Q(:,i)) - T(i,i-1)*Q(:,i-1);
    end
    T(i,i) = Q(:,i+1)'*Q(:,i);
    Q(:,i+1) = Q(:,i+1) - T(i,i)*Q(:,i);
    T(i,i+1) = norm(Q(:,i+1)); T(i+1,i) = T(i,i+1);
    Q(:,i+1) = Q(:,i+1) / T(i,i+1);
end
T = T(1:k,1:k); Q = Q(:,1:k);

end
```

- The first column of Q is $q_1 = f/\|f\|$.

- For each $1 \le t \le k$, the leading t columns of Q form a basis for the Krylov subspace $\operatorname{span}\{f, Mf, \ldots, M^{t-1}f\}$.

- The matrix $T = Q^*MQ$.

The computational cost of the Lanczos method is k matvecs with M plus $O(nk)$ additional storage and arithmetic. See Program 22.2.

The Lanczos method is frequently used to approximate the action $g(M)f$ of a function of M on the vector f. Specifically, the *Lanczos function approximation* is defined as

$$\widehat{g(M)f} := Q\, g(Q^*MQ)\, Q^*f = \|f\| \cdot Q g(T)\mathbf{e}_1.$$

In particular, the choice $g(x) = x^{-1}$ yields the *Lanczos linear solver*:

$$\widehat{M^{-1}f} = \|f\| \cdot QT^{-1}\mathbf{e}_1.$$

We have the following equivalence:

> *In exact arithmetic*, the LSQR algorithm applied to the least-squares problem
>
> $$x = \underset{z \in \mathbb{K}^n}{\mathrm{argmin}} \|c - Bx\|$$
>
> produces the same sequence of iterates $x_0, x_1, \ldots$ as either the conjugate gradient method or the Lanczos method applied to the normal equations
>
> $$(B^* B)x = B^* c$$

Given both the strong theoretical results for Lanczos and the equivalence between LSQR, CG, and Lanczos, it is natural to analyze versions of sketch-and-precondition that use Lanczos on the normal equations in place of LSQR.

Remark 22.4 (Reorthogonalization). When the Lanczos algorithm is executed in finite-precision arithmetic, rounding errors leads the matrix Q to deviate from orthonormality. This loss of orthogonality can lead to significant problems for some applications of the Lanczos method, which may be mitigated using different types of *reorthogonalization* strategies; see [Dem97, §7.4–7.6] for a discussion of several options. Remarkably, for Lanczos function approximation and Lanczos linear solves, the Lanczos algorithm produces satisfactory results in finite precision arithmetic even without reorthogonalization [DK91; DGK98; MMS18]; see [Che24, Ch. 4] for a discussion of this surprising phenomenon. All results in this thesis apply to the plain version of the Lanczos algorithm without any form of reorthogonalization. ⋄

22.4 Sketch-and-precondition with sketch-and-solve initialization

After the initial release of Meier et al.'s manuscript, a curious thing was observed: Using the sketch-and-solve initialization greatly improves the numerical stability of sketch-and-precondition. The published version of [MNTW24] provides evidence that the *forward error* of sketch-and-precondition with sketch-and-solve initialization is close to that of QR-based direct methods (i.e., evidence of *weak* forward stability). My paper [Epp24b] shows that the residual error is also comparable with QR-based methods, providing evidence of *strong* forward stability. For difficult problems, however, sketch-and-precondition with sketch-and-solve initialization is still not to backward stable; see Fig. 22.1 ahead.

In [MNTW24; Epp24b], the stabilizing effects of the sketch-and-solve initialization are presented as empirical findings, and no rigorously explanation was available. Maike Meier, Yuji Nakatsukasa, and myself provided such an explanation by proving

the following result [EMN25, Thm. 8.2]:

Theorem 22.5 (Sketch-and-precondition: Strong forward stability)**.** *Assume we work over the* real *field* $\mathbb{K} = \mathbb{R}$*. Let* $S \in \mathbb{R}^{m \times d}$ *be a subspace embedding* for *$[B \quad c]$ of distortion $\eta \leq 0.9$ and with $d \leq m$, suppose B is numerically full-rank $\kappa u \ll 1$, and assume the multiply operation is stable $\|\mathrm{err}(S^* v)\| \lesssim \|v\| u$. Then the* Lanczos-based *implementation of sketch-and-precondition with sketch-and-solve initialization produces a strongly forward stable solution to the least-squares problem* (21.1) *after* $O(\log(1/u))$ *iterations. To produce such a solution with a sparse sign embedding with Cohen's parameter settings (*Fact 21.10*), sketch-and-precondition requires at most* $O(mn \log(n/u) + n^3 \log n)$ *operations.*

A proof of a simplified version of this result is provided in Chapter 23.

22.5 Sketch-and-precondition with iterative refinement

In matrix computations, *iterative refinement* is a standard approach to improving the quality of a computed solution [Hig02, Ch. 12 & §20.5]. In its basic form, iterative refinement executes the following sequence of steps:

$$
\begin{aligned}
x_1 &:= \textsc{NotFullyStableSolver}(B, c); \\
x_2 &:= x_1 + \textsc{NotFullyStableSolver}(B, c - Bx_1).
\end{aligned}
\tag{22.7}
$$

Here, NotFullyStableSolver is any subroutine producing an approximate least-squares solution: $\textsc{NotFullyStableSolver}(B, c) \approx B^\dagger c$. Equation (22.7) describes one step of iterative refinement. In general, multiple steps of iterative refinement may be necessary to achieve full accuracy. (That is, we continue on as $x_3 := x_2 + \textsc{NotFullyStableSolver}(B, c - Bx_2)$, etc.).

The most conventional use of iteration refinement is *mixed-precision iterative refinement*. Here, the NotFullyStableSolver routine is furnished by a factorization of B computed using a lower numerical precision (i.e., single precision):

$$
(Q, R) \leftarrow \textsc{LowPrecisionQR}(B); \quad \textsc{NotFullyStableSolver}(B, c) = R^{-1}(Q^* c).
$$

In this thesis, we will perform iterative refinement in a *uniform numerical precision* u (e.g., double precision) and use sketch-and-precondition as the NotFullyStableSolver subroutine.

This motivates us to consider *sketch-and-precondition with iterative refinement* (SPIR, [EMN25]). We begin with the sketch-and-solve initialization

$$
x_0 := (S^* B)^\dagger (S^* c).
$$

Program 22.3 `spir.m`. Sketch-and-precondition with iterative refinement for solving overdetermined linear least-squares problems.

```matlab
function X = spir(B,c,St,iter1,iter2)
% Input:  Matrix B, right-hand side c, sketching matrix St, and
%         iteration count iter
% Output: Approximate least-squares solution x

    [~,D,V] = svd(St*B,"econ");       % Sketch and compute SVD
    prec = V / D;                     % Form preconditioner
    y0 = Ut*(St*c);                   % Sketch-and-solve initialize
    Y1 = mylsqr(@(z) B*(prec*z), ...  % Sketch-and-precondition
               @(z) prec'*(B'*z), ...
               c, y0, iter1);
    Y2 = mylsqr(@(z) B*(prec*z), ...  % Refinement step
               @(z) prec'*(B'*z), ...
               c, Y(:,end), iter2);
    X = prec * [Y1 Y2(:,2:end)];      % Change of variables

end
```

Then, we produce a first approximate solution by running sketch-and-precondition with initialization x_0:

$$x_1 \coloneqq \text{SKETCHANDPRECONDITION}(B, c, \texttt{initialization} = x_0),$$

Then, we apply a single step of iterative refinement

$$x_2 \coloneqq x_1 + \text{SKETCHANDPRECONDITION}(B, c - Bx_1, \texttt{initialization} = 0). \quad (22.8a)$$

Perhaps confusingly, it is imperative that we use the *zero initialization* for the iterative refinement step. Alternatively, rather than the traditional refinement step (22.8a), we can use the following *restarting refinement step*:

$$x_2 \coloneqq x_1 + \text{SKETCHANDPRECONDITION}(B, c, \texttt{initialization} = x_1). \quad (22.8b)$$

In our experience, the traditional (22.8a) and restarting (22.8b) refinement steps have similar stability properties. An implementation of SPIR appears in Program 22.3.

The backward stability of SPIR may be surprising in view of the classic work of Golub and Wilkinson [GW66], who showed that standard *mixed-precision* iterative refinement (22.7) does not greatly substantially improve the numerical accuracy for least-squares problems except for nearly consistent systems. The setting of SPIR is differs from Golub and Wilkinson's setting. SPIR performs iterative refinement

in *uniform precision*, while using preconditioned Krylov methods (appropriately initialized) as the solvers. This change in setting explains the difference in stability behavior.

Unfortunately, we do not have a *completely* unconditional proof of backward stability for SPIR. We need one small assumption:

Definition 22.6 (Non-pathological rounding error assumption). A vector $\widehat{x} \in \mathbb{K}^n$ satisfies the *non-pathological rounding error assumption* if

$$\|\widehat{x}\| \gtrsim \|x\| + \kappa u \left(\|x\| + \kappa \frac{\|c - Bx\|}{\|B\|} \right).$$

The Golub–Wilkinson–Wedin theorem (Fact 21.2) shows that the errors for a strongly forward or backward approximate least-squares solution have magnitude

$$\|\widehat{x} - x\| \lesssim \kappa u \left((\|x\| + \kappa \frac{\|c - Bx\|}{\|B\|} \right).$$

Moreover, this bound is approximately attained for "most" backward perturbations ΔB and Δc [GW66, §5]. So the non-pathological rounding error assumption just states that (1) the error $\widehat{x} - x$ is of the typical size expected for a backward stable method (that is, $\|\widehat{x} - x\|$ is not abnormally small) and (2) the errors $\widehat{x} - x$ do not catastrophically cancel with the true solution x in such a way that $\widehat{x}$ has an unusually small norm.

With this assumption, we state the following backward stability theorem:

Theorem 22.7 (SPIR: Backward stability). *Assume we work over the* real *field* $\mathbb{K} = \mathbb{R}$. *Let* $S \in \mathbb{R}^{m \times d}$ *be a subspace embedding* for $[B \quad c]$ *of distortion* $\eta \leq 0.9$ *and with* $d \leq m$, *suppose* B *is numerically full-rank* $\kappa u \ll 1$, *and assume the multiply operation is stable* $\|\mathrm{err}(S^* v)\| \lesssim \|v\| u$. *Finally, assume that the computed solution satisfies the non-pathological rounding error assumption. Then the* Lanczos-based *implementation of SPIR with traditional refinement* (22.8a) *produces a backward stable solution to the least-squares problem* (21.1) *after* $O(\log(1/u))$ *total iterations. In particular, implemented with a sparse sign embedding with Cohen's parameter settings (Fact 21.10), the runtime is* $O(mn \log(n/u) + n^3 \log n)$ *operations.*

Chapter 23 provides a proof of a simplified version of this result.

Remark 22.8 (FOSSILS). Our paper [EMN25] also introduces FOSSILS, a backward stable version of iterative Hessian sketch/sketch-and-descend. While there are a few scenarios when one might prefer FOSSILS over SPIR (cf. Remark 22.1), SPIR is generally the preferable method in practice. See [EMN25, §4.4] for discussion. ◇

Remark 22.9 ((Numerically) rank-deficient problems). For problems that are numerically rank-deficient ($\kappa u \gtrsim 1$), the sketch-and-precondition algorithm does not work properly. In this case, there are three options:

1. **"Smoothed analysis."** One way of addressing numerical rank-deficiency is to add a small independent random perturbation to each matrix entry (say, a centered Gaussian with standard deviation $10u\|\boldsymbol{B}\|_{\mathrm{F}}$). This operation has a regularizing effect on a matrix, causing it to become numerically full rank with high probability. This idea was proposed in the context of randomized least-squares solvers by Meier et al. [MNTW24]. The idea of perturbing the input to an algorithm to avoid bad instances is known as smoothed analysis [ST01], and it has established itself as a core tool in the design of randomized matrix algorithms [KT24, §6.3]. A disadvantage of this approach is that it destroys any benevolent structure that may be present in the matrix $\boldsymbol{B}$, such as sparsity.

2. **Ridge regularization.** Another way of regularizing a least-squares problem is to add a small ridge-regularization penalty to the least-squares objective:

$$\boldsymbol{x}_{\mathrm{reg}} = \underset{z \in \mathbb{K}^n}{\operatorname{argmin}} \|\boldsymbol{c} - \boldsymbol{B}z\|^2 + \mu^2\|z\|^2.$$

The ridge-regularized least-squares problem is equivalent to an ordinary least-squares problem involving an augmented matrix:

$$\boldsymbol{x}_{\mathrm{reg}} = \underset{z \in \mathbb{K}^n}{\operatorname{argmin}} \left\| \begin{bmatrix} \boldsymbol{c} \\ \boldsymbol{0} \end{bmatrix} - \begin{bmatrix} \boldsymbol{B} \\ \mu\mathbf{I} \end{bmatrix} z \right\|.$$

To solve rank-deficient problems, we can add a small amount of ridge regularization (say, $\mu = 10u\|\boldsymbol{B}\|_{\mathrm{F}}$). See *arXiv version 2* of [EMN25] for details.

3. **Truncation.** A third approach is to truncate the least-squares problem by computing an SVD $\boldsymbol{S}^*\boldsymbol{B} = \boldsymbol{U\Sigma V}^*$ of the sketched matrix. Then, instead of using the standard-inverse preconditioner $\boldsymbol{M}^{-1} = \boldsymbol{V\Sigma}^{-1}$, one executes preconditioned LSQR with a *rectangular* inverse-preconditioner $\boldsymbol{P} := \boldsymbol{V}(:, 1 : r)\boldsymbol{\Sigma}(1 : r, 1 : r)^{-1}$, where $1 \leq r \leq n$ is the number of singular values of $\boldsymbol{S}^*\boldsymbol{B}$
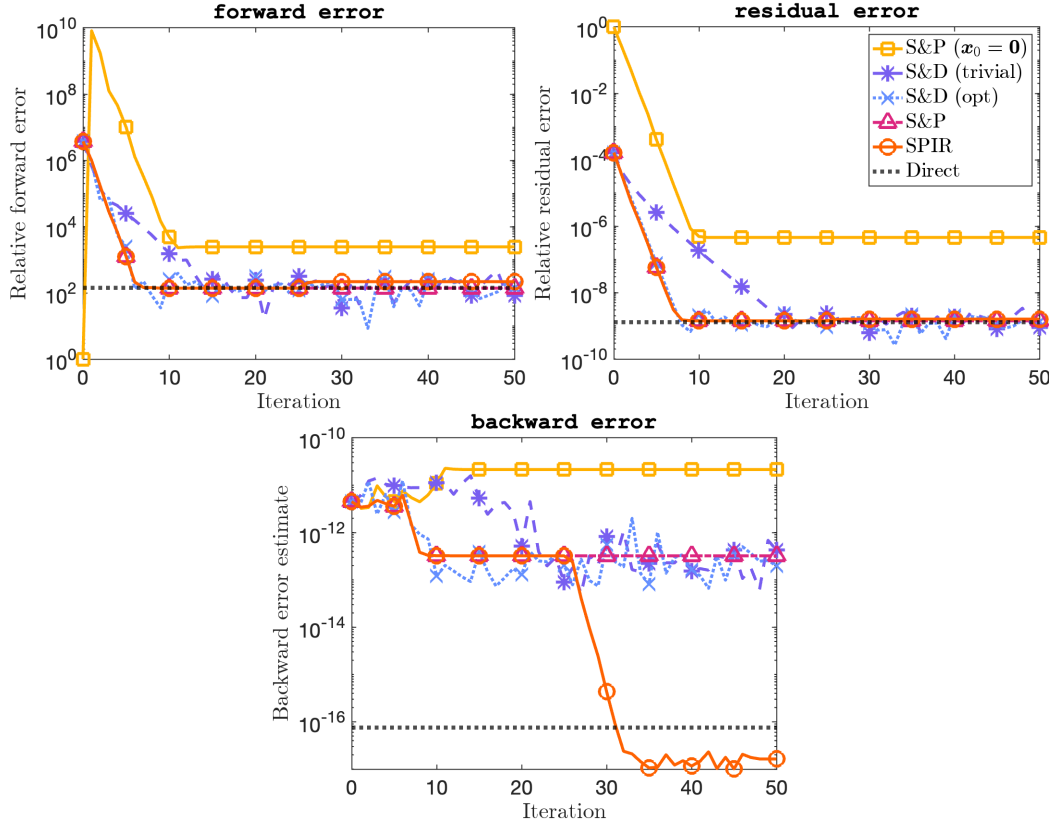
Figure 22.1: Forward (*top left*), residual (*top right*), and backward (*bottom*) errors for five randomized least-squares solvers: sketch-and-precondition with zero initialization (yellow solid squares), sketch-and-descend with trivial parameters $\alpha = 1$, $\beta = 0$ (purple dashed asterisks), sketched and descend with optimized parameters (22.4) (blue dotted crosses), sketch-and-precondition with sketch-and-solve initialization (pink dash-dotted triangles), and SPIR (restarted after 25 iterations, orange solid circles). We present a single execution. Errors for direct method (MATLAB's `mldivide`) shown for reference.

above a level, say, $10\sigma_1 u$. Truncating in this way has the effect of restricting the solution to lie in the range of $V(:, 1 : r)$, a subspace on which the linear transformation B is numerically full-rank. See [EMN25] for details.

◇

22.6 Experiments

Figure 22.1 compares the accuracy, stability, and convergence rates of different randomized least-squares problems. We use the same problem instance from Fig. 21.2, and we use a sparse sign embedding with embedding dimension $d := 20n$ and sparsity level $\zeta = 4$ for all methods. (A larger embedding dimension is necessary

to ensure convergence of sketch-and-descend with trivial parameter choices.) We see that all methods, except sketch-and-precondition with the zero initialization, are strongly forward stable. The convergence rate of sketch-and-descend with the trivial parameter settings is slower than the other methods; all other methods have the same geometric rate of convergence. We see that all methods except SPIR fail to be backward stable, with the backward error stagnating far above the level of the direct method. SPIR, with refinement performed starting at iteration 25, converges to full backward stability. In fact, SPIR achieves a slightly smaller backward error than even the direct method.

22.7 The backward error and its estimation

As stated in Definition 21.3, an approximate least-squares solution $\widehat{x} \in \mathbb{K}^n$ is backward stable if there are small perturbations Δc and ΔB, of unit roundoff size, such that $\widehat{x}$ is the exact solution of the perturbed system

$$\widehat{x} = \operatorname*{argmin}_{z \in \mathbb{K}^n} \|(c + \Delta c) - (B + \Delta B)z\|.$$

The minimal size of these perturbations is called the backward error.

Definition 22.10 (Backward error)**.** Introduce a parameter $\theta \in [0, +\infty]$. The *backward error* is

$$\mathrm{BE}_\theta(\widehat{x}) := \min \left\{ \| [\Delta B \quad \theta \cdot \Delta c] \|_\mathrm{F} : \widehat{x} = \operatorname*{argmin}_{z \in \mathbb{K}^n} \|(c + \Delta c) - (B + \Delta B)z\| \right\}.$$

The most natural values for θ are $\theta = +\infty$ (i.e., Δc is required to be set to zero) or, provided B and c are commensurately scaled, $\theta = 1$ (equal weight placed on both ΔB and Δc). Assuming the normalization $\|B\| = \|c\| = 1$, an approximate least-squares solution $\widehat{x}$ is backward stable if and only if $\mathrm{BE}_1(\widehat{x}) \lesssim u$.

To deploy randomized least-squares solvers, it is desirable to have *estimates* of the backward error that are computable at runtime. These error estimates can be used as *stopping criteria* for the algorithm.

The question of how to compute or estimate the backward error of an approximate least-squares solution has a surprisingly deep history. As Higham writes [Hig02, §20.7], "Although it has been known since the 1960s that a particular method for solving the LS problem, namely the Householder QR factorization method, yields a small normwise backward error... it was for a long time an open problem to obtain a formula for the backward error of an arbitrary approximate solution."

Program 22.4 `backerr_est.m`. Karlson–Waldén estimate of the backward error for an least-squares solution.

```
function be = backerr_est(B,c,x,s,V)
% Input:  Matrix B, column vector of singular values s,
%         right singular vectors V of B, right-hand side c, and
%         approximate LS solution x
% Output: Backward error estimate be for min ||c-B*x||

    r = c - B*x;                    % Residual
    om = norm(r) / norm(x);
    be = om / norm(r) * norm((V'*(B'*r)) ./ (s.^2 + om^2).^0.5);

end
```

This problem was resolved in 1995 by Waldén, Karlson, and Sun [WKS95], who discovered a formula for it. Higham developed a numerically stable version of Waldén, Karlson, and Sun's formula. Unfortunately, the Waldén–Karlson–Sun–Higham formula requires a steep $O(m^3)$ operations to evaluate, making it impractical for many use cases.

The Karlson–Waldén estimate. The great cost of the Waldén–Karlson–Sun–Higham formula inspired a flurry of research effort over two decades to develop and analyze computationally efficient *estimates* for the backward error. This line of work was initiated by Karlson and Waldén [KW97] and refined over a series of follow-up works [Gu98; Grc03; GSS07; GJT12]. These papers converged on estimates of the following form [GSS07, §2]:

$$\widehat{\mathrm{BE}}_\theta(\widehat{\boldsymbol{x}}) := \frac{\theta}{(1 + \theta^2 \|\widehat{\boldsymbol{x}}\|^2)^{1/2}} \left\| \left(\boldsymbol{B}^* \boldsymbol{B} + \frac{\theta^2 \|\boldsymbol{c} - \boldsymbol{B}\widehat{\boldsymbol{x}}\|^2}{1 + \theta^2 \|\widehat{\boldsymbol{x}}\|^2} \right)^{-1/2} \boldsymbol{B}^*(\boldsymbol{c} - \boldsymbol{B}\widehat{\boldsymbol{x}}) \right\|.$$

Following previous papers [GJT12], we shall this the *Karlson–Waldén estimate* for the backward error. This estimate can be evaluated in $O(mn^2)$ operations. Given a QR factorization or SVD of $\boldsymbol{B}$, then $O(mn)$ operations suffice [GSS07]. An implementation of the Karlson–Waldén estimate appears in Program 22.4.

The series of papers [Gu98; Grc03; GSS07] gave increasingly sharp characterizations of the quality of the Karlson–Waldén estimate. Gratton, Jiránek, and Titley-Peloquin conclusively resolved this line of research, providing a sharp analysis of the estimate [GJT12]. Here is a simplified version of their result:

Fact 22.11 (Karlson–Waldén estimate). *Over the* real *field* $\mathbb{K} = \mathbb{R}$, *the Karlson–Waldén estimate is sharp within a factor of* $\sqrt{2}$:

$$1 \cdot \widehat{\mathrm{BE}}_\theta(\widehat{x}) \le \mathrm{BE}_\theta(\widehat{x}) \le \sqrt{2} \cdot \widehat{\mathrm{BE}}_\theta(\widehat{x}) \tag{22.9}$$

for every $\theta \in [0, +\infty]$ *and* $\widehat{x} \in \mathbb{R}^n$. *The constants* 1 *and* $\sqrt{2}$ *in* (22.9) *cannot be improved.*

Improving the Karlson–Waldén estimate with sketching. The cost of $O(mn^2)$ operations for the Karlson–Waldén estimate is still prohibitive for deployment with randomized least-squares solvers, since our goal is to achieve runtimes of roughly $O(mn + n^3)$ operations. Fortunately, we can accelerate the Karlson–Waldén estimate with sketching. My coauthors and I in [EMN25] proposed the *sketched Karlson–Waldén estimate*:

$$\widehat{\mathrm{BE}}_{\mathrm{sk},\theta}(\widehat{x}) := \frac{\theta}{(1 + \theta^2 \|\widehat{x}\|^2)^{1/2}} \left\| \left((S^*B)^*(S^*B) + \frac{\theta^2 \|c - B\widehat{x}\|^2}{1 + \theta^2 \|\widehat{x}\|^2} \right)^{-1/2} B^*(c - B\widehat{x}) \right\|.$$

Here, S is a subspace embedding. We have the following analysis for the sketched Karlson–Waldén estimate, adapted from [EMN25, Prop. 4.2]:

Proposition 22.12 (Sketched Karlson–Waldén estimate). *Consider the* real *field* $\mathbb{K} = \mathbb{R}$, *let* $\theta \in [0, +\infty]$, $\widehat{x} \in \mathbb{R}^n$, *and let* $S \in \mathbb{R}^{m \times d}$ *be a subspace embedding with distortion* η. *The sketched Karlson–Waldén estimate is sharp up to the following bounds:*

$$(1 - \eta) \cdot \widehat{\mathrm{BE}}_\theta(\widehat{x}) \le \mathrm{BE}_\theta(\widehat{x}) \le \sqrt{2}(1 + \eta) \cdot \widehat{\mathrm{BE}}_\theta(\widehat{x}).$$

To compute the sketched Karlson–Waldén estimate, we can modify Program 22.4 to use the SVD of the sketched B matrix, $S^*B = U\Sigma V^*$, in place of the original B matrix. With a sparse sign embedding (Definition 21.9), the estimator requires a one-time set-up cost of roughly $O(mn + n^3)$ operations, after which the cost of computing the estimator is $O(mn)$ operations.

Adaptively stopping sketch-and-precondition and SPIR. We can use the sketched Karlson–Waldén estimate to adaptively stop SPIR when backward stability has been achieved. We can also use automatic methods to switch between the two stages of SPIR. See [EMN25, §4] for details.

Chapter 23

PROOFS OF STABILITY

Once in a lifetime a user of computer arithmetic should examine the details of a backward error analysis.

Beresford N. Parlett, *The Symmetric Eigenvalue Problem*, [Par98, §2.6.1]

The two main classes of rounding error analysis are not, as my audience might imagine, 'backwards' and 'forwards', but rather 'one's own' and 'other people's'. One's own is, of course, a model of lucidity; that of others serves only to obscure the essential simplicity of the matter in hand.

James Wilkinson, NAG 1984 Annual General Meeting [Wil85]

In this chapter, we will provide stability analysis for sketch-and-precondition and sketch-and-precondition with iterative refinement (SPIR). As Wilkinson's quote at the beginning of the section highlights, every numerical analyst has their own approach to rounding error analysis. My approach more qualitative, and I prefer to avoid carefully tracking the size of constants and prefactors. Even at this level of granularity, the numerical stabilility analysis of randomized iterative methods involves lengthy calculations and estimates.

For this thesis, I will provide the simplest argument for strong forward stability of sketch-and-precondition and backward stability of SPIR that I am aware of. To make the analysis as simple as possible, this thesis will develop first-order perturbation estimates. The original paper [EMN25] contains detailed analysis which accounts for the higher-order terms. Throughout this section, important intermediate results are boxed:

> I am an important result.

Sources. This chapter is adapted from the following paper:

Ethan N. Epperly, Maike Meier, and Yuji Nakatsukasa. "Fast Randomized Least-Squares Solvers Can Be Just as Accurate and Stable as Classical Direct Solvers". In: *Communications on Pure and Applied Mathematics, accepted* (2025). Preprint available at https://arxiv.org/abs/2406.03468v2.

Outline. Section 23.1 introduces standing assumptions and notation for this section. Sections 23.2 and 23.3 analyze the floating-point errors incurred during sketching. Next, in Sections 23.4 and 23.5, we analyze the stability of multiplications by the matrices B and R^{-1}, as well as their adjoints. We take a brief pause from analysis in Section 23.6, which describes existing stability results for Lanczos linear solves. Section 23.7 contains the core of the stability analysis, establishing a general formula the floating point errors introduced in a single iterative refinement step with sketch-and-precondition. As a consequence of this general formula, we derive strong forward stability for sketch-and-precondition (with sketch-and-solve initialization) in Section 23.8 and backward stability of SPIR in Section 23.9.

23.1   Standing assumptions and more notation

Throughout this section, we will assume the normalization

$$\|B\| = \|c\| = 1.$$

As in Chapter 22, we will denote $\kappa := \operatorname{cond}(B)$, and we will use the standing assumption that $\kappa u \ll 1$.

A vector $z \in \mathbb{K}^n$ is *exactly represented* if $\mathrm{fl}(z) = z$. Throughout, e, e', etc. denote arbitrary vectors of norm $\lesssim u$ (likewise, for matrices E, E', etc.). We will freely interchange the value of e from line-to-line, even in the same line (i.e., we could write $e + e' = e$). We will use HOT ("higher order terms") to denote any term proportional to u^2, suppressing all prefactors depending on m, n, the norms of various objects, and κ. So, for our purposes $\|z\|\kappa^{100}u^2 = \text{HOT}$. The "HOT" notation helps to streamline stability arguments to their essence, at the cost of losing precise information about the size of higher-order terms.

23.2   Sketching and QR factorizing

Our first order of business will be the assess the quality of the randomized preconditioner R in the presence of rounding errors.

First, we investigate the rounding errors incurred in sketching and computing the QR decomposition. We have assumed the sketching operation is numerically stable

in the sense that

$$\mathrm{fl}(S^*B) = S^*B + E.$$

QR factorization is backward stable [Hig02, Thm. 19.4], so there exists a matrix U for which

$$U\,\mathrm{fl}(R) = \mathrm{fl}(S^*B) + E = S^*B + E. \qquad (23.1)$$

Now, we use (23.1) to assess the quality of the *numerically computed* R matrix as a preconditioner for B. By the subspace embedding property, we have

$$\left\|B\,\mathrm{fl}(R)^{-1}\right\| \le \frac{1}{1-\eta} \cdot \left\|S^*B\,\mathrm{fl}(R)^{-1}\right\|$$

$$= \frac{1}{1-\eta} \cdot \left\|U + E\,\mathrm{fl}(R)^{-1}\right\| \le \frac{1}{1-\eta} + \cdot\frac{1}{1-\eta}\frac{\|E\|}{\sigma_{\min}(\mathrm{fl}(R))}.$$

To bound the minimum singular value of $\mathrm{fl}(R)$, we compute

$$\sigma_{\min}(\mathrm{fl}(R)) = \sigma_{\min}(S^*B + E) \ge \sigma_{\min}(S^*B) - \|E\|$$

$$\ge \frac{1}{1+\eta} \cdot \sigma_{\min}(B) - \|E\| \ge \frac{1}{(1+\eta)\kappa} - O(u) \ge \frac{\mathrm{const}}{\kappa}. \quad (23.2)$$

In the last inequality, we used the hypothesis $\kappa u \ll 1$ (so that $u \ll 1/\kappa$). Therefore,

$$\left\|B\,\mathrm{fl}(R)^{-1}\right\| \le \frac{1}{1-\eta} + O(\kappa u) \le \mathrm{const.} \qquad (23.3)$$

A similar argument shows

$$\|R\| \le \mathrm{const.} \qquad (23.4)$$

and

$$\sigma_{\min}(B\,\mathrm{fl}(R)^{-1}) \ge \frac{1}{1+\eta} - O(\kappa u) \ge \mathrm{const} > 0. \qquad (23.5)$$

Thus, the numerically computed matrix $\mathrm{fl}(R)$ is a good preconditioner for B, satisfying the following bound:

$$\mathrm{cond}(B\,\mathrm{fl}(R)^{-1}) = \mathrm{cond}(BR^{-1}) + O(\kappa u) \le \mathrm{const} < 1.$$

To make our lives easier, we make the following notational affordance: *Going forward, the symbol R will denote the numerically computed preconditioner R.*

23.3 The sketch-and-solve solution

Now, we analyze the numerically computed sketch-and-solve solution. Householder QR factorization is a backward stable for solving a least-squares problem [Hig02, Thm. 20.3]. Therefore, the numerically computed sketch-and-solve solution $x_0 = (S^*B)^\dagger(S^*c)$ is

$$\mathrm{fl}(x_0) = (S^*B + E)^\dagger(S^*c)$$

To first order, the pseudoinverse of a perturbation of a matrix F is

$$(F + E)^\dagger = F^\dagger + (F^*F)^{-1}E^*(I - FF^\dagger) - F^\dagger EF^\dagger + \mathrm{HOT}.$$

Instantiating this result with $F := S^*B$ and invoking the identity $x_0 = (S^*B)^\dagger(S^*c)$, we obtain

$$\|B(x - \mathrm{fl}(x_0))\| = \|B(x - x_0)\| + \left\|B(B^*SS^*B)^{-1}E^*S^*(c - Bx_0)\right\|$$
$$+ \left\|B(S^*B)^\dagger Ex_0\right\| + \mathrm{HOT}.$$

By the analysis of sketch-and-solve (Theorem 21.8), the residual error is bounded: $\|B(x - x_0)\| \lesssim \|c - Bx\|$. Now, invoking the subspace embedding property and the submultiplicative property of the spectral norm, we have

$$\|B(x - \mathrm{fl}(x_0))\| \lesssim \|c - Bx\| + \|S^*B(B^*SS^*B)^{-1}\|\|E\|\|c - Bx\|$$
$$+ \|S^*B(S^*B)^\dagger\|\|E\|\|x_0\| + \mathrm{HOT}.$$

We have $S^*B(B^*SS^*B)^{-1} = (SB)^{\dagger*}$, which has norm $\lesssim \kappa$. Thus,

$$\|B(x - \mathrm{fl}(x_0))\| \lesssim \|c - Bx\| + \|x_0\|u + \mathrm{HOT}.$$

Finally, we note that $\|x_0\|u \leq \|x\|u + \|x - x_0\|u = \|x\|u + \mathrm{HOT}$. We conclude

$$\|B(x - \mathrm{fl}(x_0))\| \lesssim \|c - Bx\| + \|x\|u + \mathrm{HOT}. \qquad (23.6)$$

The residual error of the numerically computed sketch-and-solve solution is bounded by a multiple of the residual $\|c - Bx\|$ plus $\|x\|u$.

23.4 Stability of multiplication and triangular solves

The basic primitive in all of these algorithms are taking linear combinations of vectors and performing matrix multiplications. The stability of these basic operations

is analyzed in [Hig02, §§2–3]. We will use the crudest versions of these results. For exactly represented vectors $z, w \in \mathbb{K}^n$ and $v \in \mathbb{K}^m$ and scalar $\xi \in \mathbb{K}$, we have

$$\|\text{err}(\xi \cdot z)\| \lesssim |\xi| \cdot \|z\| + \text{HOT}, \tag{23.7a}$$

$$\|\text{err}(z \pm w)\| \lesssim \|z \pm w\| u + \text{HOT}, \tag{23.7b}$$

$$\|\text{err}(\boldsymbol{B}z)\| \lesssim \|z\| u + \text{HOT}, \tag{23.7c}$$

$$\|\text{err}(\boldsymbol{B}^* v)\| \lesssim \|v\| u + \text{HOT}. \tag{23.7d}$$

Solves by triangular matrices will be another important primitive for us. The key fact about triangular solves is that they are *backward stable*. In particular, for an exactly represented vector z,

$$\text{fl}(\boldsymbol{R}^{-1}z) = (\boldsymbol{R} + \Delta\boldsymbol{R})^{-1}z \quad \text{for } \|\Delta\boldsymbol{R}\| \lesssim u.$$

Here, we used the bound $\|\boldsymbol{R}\| \lesssim 1$ (equation (23.4)). Using the first-order expansion $(\boldsymbol{R} + \Delta\boldsymbol{R})^{-1} = \boldsymbol{R}^{-1} - \boldsymbol{R}^{-1} \cdot \Delta\boldsymbol{R} \cdot \boldsymbol{R}^{-1} + \text{HOT}$, it follows that

$$\text{fl}(\boldsymbol{R}^{-1}z) = [\boldsymbol{R}^{-1} - \boldsymbol{R}^{-1} \cdot \Delta\boldsymbol{R} \cdot \boldsymbol{R}^{-1}]z + \text{HOT} = \boldsymbol{R}^{-1}z + \|\boldsymbol{R}^{-1}z\| \cdot \boldsymbol{R}^{-1}e + \text{HOT}.$$

We conclude that

$$\text{err}(\boldsymbol{R}^{-1}z) = \|\boldsymbol{R}^{-1}z\| \cdot \boldsymbol{R}^{-1}e + \text{HOT}. \tag{23.8a}$$

A similar argument shows that

$$\text{err}(\boldsymbol{R}^{-*}z) = \|\boldsymbol{R}^{-*}z\| \cdot \boldsymbol{R}^{-*}e + \text{HOT}. \tag{23.8b}$$

23.5 Stability of interleaved multiplications

The sketch-and-precondition algorithm consists sequences of multiplies with the matrix $\boldsymbol{B}$, the preconditioner $\boldsymbol{R}$, and their adjoints.

Let us first analyze the floating-point errors incurred in evaluating the product $\boldsymbol{R}^{-*}\boldsymbol{B}^*z$. Combining (23.7d) and (23.8b), we obtain

$$\text{err}(\boldsymbol{R}^{-*}\boldsymbol{B}^*z) = \underbrace{\|\boldsymbol{R}^{-*}\boldsymbol{B}^*z\| \cdot \boldsymbol{R}^{-*}e}_{\text{errors from applying } \boldsymbol{R}^{-*}} + \underbrace{\|z\| \cdot \boldsymbol{R}^{-*}e'}_{\boldsymbol{R}^{-*} \cdot \text{err}(\boldsymbol{B}^*z)} + \text{HOT}.$$

Applying the bound $\|\boldsymbol{B}\boldsymbol{R}^{-1}\| \lesssim 1$ (equation (23.3)), we conclude

$$\text{err}(\boldsymbol{R}^{-*}\boldsymbol{B}^*z) = \|z\| \cdot \boldsymbol{R}^{-*}e + \text{HOT}. \tag{23.9}$$

In the Lanczos-based sketch-and-precondition method, we use Lanczos to solve the normal equations

$$My = f \quad \text{for } M := R^{-*}B^*BR^{-1}, \ f := R^{-*}B^*(c - Bx_i).$$

As shown in (23.10), multiplies $Mz = R^{-*}(B^*(B(R^{-1}z)))$ is forward stable in effect precision $\widetilde{u} = \kappa u$, up to higher order terms. Further, by (23.3) and (23.5), the matrix M is well-conditioned, $\text{cond}(M) \leq \text{const}$. Thus, Fact 23.1 implies that Lanczos produces a solution to the normal equations satisfying

$$\left\| \text{fl}(y) - M^{-1}\,\text{fl}(f) \right\| \lesssim \kappa u \|\text{fl}(f)\| + \text{HOT}. \qquad (23.12)$$

23.7 The error formula

We now analyze an iterative refinement step using sketch-and-precondition:

$$x_{i+1} \leftarrow x_i + R^{-1} \underbrace{(R^{-*}B^*BR^{-1})^{-1}}_{\text{applied via Lanczos}} R^{-*}B^*(c - Bx_i). \qquad (23.13)$$

Here, x_i denotes an arbitrary ith iterate. Throughout this section, we let fl and err denote only the floating-point errors incurred during the refinement step (23.13).

Begin by using the error bounds for vector subtraction (23.7b) and matrix multiplication (23.7c) to obtain

$$\|\text{err}(c - Bx_i)\| \lesssim (1 + \|x_i\|)u + \text{HOT}.$$

Next, invoking (23.9) yields

$$\text{err}(R^{-*}B^*(c - Bx_i)) = \underbrace{(1 + \|x_i\|)R^{-*}B^*e}_{R^{-*}B^* \cdot \text{err}(c - Bx_i)} + \underbrace{\|c - Bx_i\| \cdot R^{-*}e'}_{\text{errors from applying } R^{-*}B^*} + \text{HOT}.$$

Employing the bound $\|R^{-*}B^*\| \lesssim 1$ ((23.3)), this equation simplifies

$$\text{err}(R^{-*}B^*(c - Bx_i)) = (1 + \|x_i\|)e + \|c - Bx_i\| \cdot R^{-*}e' + \text{HOT}.$$

Now, introduce the shorthand $M := R^{-*}B^*BR^{-1}$ and apply the Lanczos linear solve bound (23.12) to obtain

$$\text{err}(M^{-1}R^{-*}B^*(c - Bx_i)) = \underbrace{(1 + \|x_i\|)e + \|c - Bx_i\| \cdot M^{-1}R^{-*}e'}_{M^{-1} \cdot \text{err}(R^{-*}B^*(c - Bx_i))}$$

$$+ \underbrace{\kappa \|R^{-*}B^*(c - Bx_i)\|e''}_{\text{errors from applying } M^{-1}} + \text{HOT}. \qquad (23.14)$$

We have simplified by noting that $\|M^{-1}\| \leq$ const. To reduce further, we note that

$$\|R^{-*}B^*(c - Bx_i)\| = \|R^{-*}B^*(c - Bx + B(x - x_i))\|$$
$$= \|R^{-*}B^*(Bx - Bx_i)\| \lesssim \|B(x - x_i)\|.$$

The second equality is the orthogonality of the residual $c - Bx$ to the range of B and the inequality is (23.3). We also note that $M^{-1}R^{-*} = R(B^*B)^{-1}$. Applying all of these observations and combining with (23.14), we obtain

$$\mathrm{err}(M^{-1}R^{-*}B^*(c - Bx_i)) = (1 + \|x_i\| + \kappa\|B(x - x_i)\|)e$$
$$+ \|c - Bx_i\| \cdot R(B^*B)^{-1}e' + \mathrm{HOT}.$$

Now, employ the inversion bound (23.8a) and addition bound (23.7b) to conclude

$$\mathrm{err}(x_{i+1}) = \underbrace{(1 + \|x_i\| + \kappa\|B(x - x_i)\|)R^{-1}e + \|c - Bx_i\| \cdot (B^*B)^{-1}e'}_{R^{-1}\cdot\mathrm{err}(M^{-1}R^{-*}B^*(c - Bx_i))}$$
$$+ \underbrace{\|R^{-1}M^{-1}R^{-*}B^*(c - Bx_i)\| \cdot R^{-1}e''}_{\text{errors from applying } R^{-1}} + \underbrace{\|x\|e'''}_{\text{errors from final addition}} + \mathrm{HOT}.$$

This display constitutes an error bound for a single sketch-and-precondition step. To simplify, note that

$$\left\|R^{-1}M^{-1}R^{-*}B^*(c - Bx_i)\right\| = \left\|B^\dagger(c - Bx_i)\right\| = \|x - x_i\| \leq \|x\| + \|x_i\|.$$

Thus, we have shown

$$\mathrm{err}(x_{i+1}) = (1 + \|x\| + \|x_i\| + \kappa\|B(x - x_i)\|)R^{-1}e$$
$$+ \|c - Bx_i\| \cdot (B^*B)^{-1}e' + \mathrm{HOT}. \quad (23.15)$$

23.8 Forward stability of sketch-and-precondition

Now we show strong forward stability of sketch-and-precondition with the sketch-and-solve initialization. By (23.6), the numerically computed sketch-and-solve solution satisfies

$$\|\mathrm{fl}(x_0)\| \leq \|x\| + \|x - \mathrm{fl}(x_0)\| \leq \|x\| + \kappa\|B(x - \mathrm{fl}(x_0))\| \lesssim \|x\| + \kappa\|c - Bx\|.$$

Here, we used the hypothesis $\kappa u \ll 1$. Similarly,

$$1 = \|c\| \leq \|c - Bx\| + \|x\|.$$

Substituting the previous two displays in the error formula (23.15) gives

$$\text{err}(\boldsymbol{x}_1) = (\|\boldsymbol{x}\| + \kappa \|\boldsymbol{c} - \boldsymbol{B}\boldsymbol{x}\|)\boldsymbol{R}^{-1}\boldsymbol{e} + \|\boldsymbol{c} - \boldsymbol{B}\boldsymbol{x}\| \cdot (\boldsymbol{B}^*\boldsymbol{B})^{-1}\boldsymbol{e}' + \text{HOT}.$$

Finish up by multiplying by $\boldsymbol{B}$, taking norms, and using the bound $\mathrm{s}\|\boldsymbol{B}\boldsymbol{R}^{-1}\| \lesssim 1$ (equation (23.3)) and $\|\boldsymbol{B}(\boldsymbol{B}^*\boldsymbol{B})^{-1}\| = \|\boldsymbol{B}^{\dagger}\| = \kappa$. We obtain

$$\|\boldsymbol{B}(\boldsymbol{x} - \text{fl}(\boldsymbol{x}_1))\| \lesssim \|\boldsymbol{x}\|u + \kappa \|\boldsymbol{c} - \boldsymbol{B}\boldsymbol{x}\|u + \text{HOT}. \qquad (23.16)$$

This bound is strong forward stability, up to the higher-order terms.

23.9   Backward stability of SPIR

Now, we show backward stability of SPIR. Our main tool will be the following characterization [EMN25, Thm. 2.8 & Cor. 2.9]:

Theorem 23.2 (Backward stability: Componentwise errors). *Consider a real least-squares problem* (21.1) *with the normalization* $\|\boldsymbol{B}\| = \|\boldsymbol{c}\| = 1$, *and introduce the SVD* $\boldsymbol{B} = \sum_{i=1}^n \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^*$. *A vector* $\widehat{\boldsymbol{x}}$ *is backward stable solution to the least-squares problem* (21.1) *if and only if*

$$\left|\boldsymbol{v}_i^*(\boldsymbol{x} - \widehat{\boldsymbol{x}})\right| \lesssim \sigma_i(\boldsymbol{B})^{-1} \cdot (1 + \|\widehat{\boldsymbol{x}}\|)u + \sigma_i(\boldsymbol{B})^{-2} \cdot \|\boldsymbol{c} - \boldsymbol{B}\widehat{\boldsymbol{x}}\|u \quad \textit{for each } i = 1, \ldots, n.$$

Consequently, if the error takes the form

$$\boldsymbol{x} - \widehat{\boldsymbol{x}} = (1 + \|\widehat{\boldsymbol{x}}\|) \cdot \text{fl}(\boldsymbol{R})^{-1}\boldsymbol{e} + \|\boldsymbol{c} - \boldsymbol{B}\widehat{\boldsymbol{x}}\| \cdot (\boldsymbol{B}^*\boldsymbol{B})^{-1}\boldsymbol{e}',$$

then $\widehat{\boldsymbol{x}}$ *is backward stable.*

The proof follows by comparing the backward error to the Karlson–Waldén estimate using Fact 22.11, then decomposing the Karlson–Waldén estimate using the singular value decomposition of $\boldsymbol{B}$. The computation is straightforward, and we omit the details here.

To prove backward stability of SPIR, we use the following ingredients: the characterization theorem (Theorem 23.2), the error formula (23.15), and the strong forward stability of sketch-and-precondition (23.16). Begin by using strong forward stability (23.16) of the numerically computed solution $\text{fl}(\boldsymbol{x}_1)$ together with the error formula (23.15) to obtain

$$\text{err}(\boldsymbol{x}_2) = (1 + \|\boldsymbol{x}\|)\boldsymbol{R}^{-1}\boldsymbol{e} + \|\boldsymbol{c} - \boldsymbol{B}\boldsymbol{x}\| \cdot (\boldsymbol{B}^*\boldsymbol{B})^{-1}\boldsymbol{e}' + \text{HOT}.$$

We see that $\mathrm{fl}(\boldsymbol{x}_2)$ and $\|\boldsymbol{x}\|$ differ by an amount of size $O(u)$, so we may replace $\|\boldsymbol{x}\|u$ by $\|\mathrm{fl}(\boldsymbol{x}_2)\|u$ up to higher order terms. Similarly, the $\boldsymbol{x}$ minimizes the least-squares residual norm, so $\|\boldsymbol{c} - \boldsymbol{B}\boldsymbol{x}\| \leq \|\boldsymbol{c} - \boldsymbol{B}\widehat{\boldsymbol{x}}\|$. Thus, we have shown

$$\mathrm{err}(\boldsymbol{x}_2) = (1 + \|\mathrm{fl}(\boldsymbol{x}_2)\|)\boldsymbol{R}^{-1}\boldsymbol{e} + \|\boldsymbol{c} - \boldsymbol{B}\,\mathrm{fl}(\boldsymbol{x}_2)\| \cdot (\boldsymbol{B}^*\boldsymbol{B})^{-1}\boldsymbol{e}' + \mathrm{HOT}.$$

This conclusion is precisely backward stability, up to higher-order terms.

Remark 23.3 (What about the benign rounding error assumption?). The careful reader may notice that our derivation did not use the non-pathological rounding error assumption (Definition 22.6). Thus, we conclude that backward stability *to first order* holds without this assumption. As far as I am aware, this assumption is needed to control higher-order terms and prove the backward stability of SPIR (Theorem 22.7). ◇

Chapter 24

SKETCHING, SOLVERS, AND STABILITY: OPEN PROBLEMS

This chapter presents open problems related to this part of the thesis. First, in Section 24.1, we discuss improving our stability analysis to treat CG or LSQR in place of Lanczos. Second, Section 24.2 investigates open problems in the numerical stability of Krylov methods. We conclude in Section 24.3 by pondering the stability properties of preconditioned iterative methods for square systems of linear equations.

24.1 From Lanczos to conjugate gradient to LSQR

The major limitation of our analysis sketch-and-precondition and SPIR in the last chapter was that we analyzed a Lanczos-based version of the procedure. In practice, we instead use LSQR or perhaps conjugate gradient on the normal equations. To close this gap between theory and practice, we ask: Could we analyze sketch-and-precondition-type methods with conjugate gradient or LSQR?

The path to analyzing sketch-and-precondition with conjugate gradient seems more clear. The conjugate gradient algorithm has already been analyzed many times in finite-precision arithmetic [Gre89; GS92; Gre97a; MS06]. Further, the pseudocodes for conjugate gradient and the Lanczos algorithm are fairly similar. Given this existing body of work and these connections, it is natural to hope that a version of Fact 23.1 should be attainable for conjugate gradient. I am actively working on this question in collaboration with Deeksha Adil, Anne Greenbaum, and Christopher Musco.

Analyzing sketch-and-precondition with LSQR seems harder as a practical matter, as there is no existing rigorous analysis of the LSQR method in finite precision arithmetic (to the best of my knowledge). However, in principle, any tools used to analyze conjugate gradient or Lanczos should be extensible to LSQR, as well.

24.2 Numerical stability of Krylov methods

In the Lanczos stability result (Fact 23.1), the $\lesssim$ notation suppresses prefactors that, apon closer analysis, are quantitatively large. The analysis of [MMS18], which underrides the proof of Fact 23.1, shows that the Lanczos algorithm solves an $n \times n$

positive definite linear system $\boldsymbol{M}\boldsymbol{f}$ up accuracy roughly

$$\|\boldsymbol{M}\| \cdot \left\|\mathrm{fl}(\boldsymbol{M}^{-1}\boldsymbol{f}) - \boldsymbol{M}^{-1}\boldsymbol{f}\right\| \lessapprox \mathrm{poly}(n) \cdot \mathrm{cond}(\boldsymbol{M})^3 \widetilde{u} \log^4(1/\widetilde{u}) \cdot \|\boldsymbol{f}\|. \quad (24.1)$$

Here, as in Fact 23.1, $\widetilde{u} \geq u$ denotes the effective precision at which matvecs with $\boldsymbol{M}$ are computed. The analysis of [DGK98, Thm. 2.2] is stronger, roughly implying that

$$\|\boldsymbol{M}\| \cdot \left\|\mathrm{fl}(\boldsymbol{M}^{-1}\boldsymbol{f}) - \boldsymbol{M}^{-1}\boldsymbol{f}\right\| \lessapprox \mathrm{poly}(n) \cdot \mathrm{cond}(\boldsymbol{M})^{2.25} \widetilde{u} \log^{0.5}(1/\widetilde{u}) \cdot \|\boldsymbol{f}\|. \quad (24.2)$$

(The full version of [DGK98, Thm. 2.2] is stronger (24.2), as it bounds the residual $\left\|\boldsymbol{f} - \boldsymbol{M}\,\mathrm{fl}(\boldsymbol{M}^{-1}\boldsymbol{f})\right\|$.) As a point of comparison, recall that a (forward) stable algorithm for the linear system $\boldsymbol{M}\boldsymbol{y} = \boldsymbol{f}$ achieves a much higher accuracy

$$\|\boldsymbol{M}\| \cdot \left\|\mathrm{fl}(\boldsymbol{M}^{-1}\boldsymbol{f}) - \boldsymbol{M}^{-1}\boldsymbol{f}\right\| \lessapprox \mathrm{poly}(n) \cdot \mathrm{cond}(\boldsymbol{M})u \cdot \|\boldsymbol{f}\|. \quad (24.3)$$

In my numerical experience, I have always found the Lanczos algorithm to be forward stable or nearly so. As such, I believe the the polylogarithmic factors $\mathrm{poly}(\log(1/\widetilde{u}))$ and superlinear powers of $\mathrm{cond}(\boldsymbol{M})$ in (24.1) and (24.2) are parasitic.

In our analysis, the unappealing aspects of (24.1) and (24.2) are hidden by the $\lessapprox$ notation. The condition number of $\boldsymbol{M} = \boldsymbol{R}^{-*}\boldsymbol{B}^*\boldsymbol{B}\boldsymbol{R}^{-1}$ is $O(1)$ and polylogarithmic factors in $\log(1/u) \geq \log(1/\widetilde{u})$ are suppressed under the $\lessapprox$ relation. Still, if one were to develop versions of stability theorems in this thesis with explicit prefactors, those prefactors would pay a steep price for the weak bounds (24.1) and (24.2). As such, developing improved versions of Fact 23.1 with better dependence on $\mathrm{cond}(\boldsymbol{M})$ and $\log(1/\widetilde{u})$ is a natural question for future work. In particular, a precise open question is to determine whether the Lanczos algorithm is forward stable (24.3).

24.3 From least squares to linear systems

On inspection, the analysis in Chapter 23 uses randomness only to *construct* the preconditioner $\boldsymbol{R}$ and the initialization $\boldsymbol{x}_0$. As such, if one had another means of producing a high-enough quality preconditioner $\boldsymbol{R}$ and initialization $\boldsymbol{x}_0$, our analysis would also establish backward stability in this case (with a single step of iterative refinement). Remarks to these effects are made in [EMN25, §6.7].

Unfortunately, finding a good initialization $\boldsymbol{x}_0$ satisfying $\|\boldsymbol{c} - \boldsymbol{B}\boldsymbol{x}_0\| \lessapprox \|\boldsymbol{c} - \boldsymbol{B}\boldsymbol{x}\|$ appears challenging beyond the setting of highly overdetermined least squares. Indeed, for a least-squares problem which is consistent ($\boldsymbol{c} = \boldsymbol{B}\boldsymbol{x}$), finding such a good initialization amounts to solving the problem! As such, randomized preconditioning

represents the only setting of which I am aware for which one has access to both high-quality initialization and high-quality preconditioner.

Still, the techniques we studied in this thesis could be useful beyond the setting of randomized least squares. In particular, it is natural to ask:

> Can we say anything about the solution to *linear systems of equations* by preconditioned iterative methods from trivial initialization $x_0 = \mathbf{0}$?

I investigated this question with Anne Greenbaum and Yuji Nakatsukasa in [EGN25], and we showed that iterative methods on the *preconditioned normal equations* converge to backward stability after sufficiently many iterative refinement. As with this thesis, the analysis only holds for Lanczos, but the numerical results suggest it holds for LSQR as well. I also make the caveat that our analysis requires that our analysis requires that applying the inverse-preconditioner $z \mapsto \mathbf{R}^{-1}z$ is done in a backward stable manner.

This leaves open a fundamental question: What about other iterative methods? Is preconditioned conjugate gradient (with iterative refinement) backward stable, provided with a high-enough quality preconditioner? What about preconditioned GMRES? These are natural and basic questions about the attainable accuracy of some of the most fundamental preconditioned iterative methods of computational mathematics, and they remain open. Some preliminary numerical evidence for preconditioned conjugate gradient is provided in [EGN25, Fig. 4], showing that preconditioned conjugate gradient is backward stable *with, and only with, iterative refinement*.

A p p e n d i x   A

INCREMENTAL QR DECOMPOSITION

In both Parts I and II of this thesis, we are interested in maintaining a QR decomposition of a tall matrix $Y \in \mathbb{K}^{m \times k}$ that evolves by having new columns appended to its end. It is straightforward using appropriate calls to a linear algebra library like LAPACK [ABBB+99], though I am unaware of a reference. For completeness, I provide a description of this "incremental QR decomposition" primitive here. I also provide a MATLAB implementation using *internal undocumented functions*, which provide an interface to the corresponding LAPACK calls. (Warning: MATLAB internal functions are subject to be removed or changed between releases.) In the following discussion, we use names of LAPACK routines for real, double-precision arithmetic.

Suppose the entries of $Y \in \mathbb{K}^{m \times k}$ are stored in column-major order in a buffer Y. New (blocks of) columns are added by appending to the end of the buffer. If the allocated space for the buffer is ever exhausted, a new buffer of twice the size is allocated and the contents of the existing buffer are copied over. To store a QR decomposition using LAPACK, we allocate a second small buffer tau, which also can be dynamically resized as necessary. See Fig. A.1(a) for an illustration.

A (full) QR decomposition

$$Y = Q \begin{bmatrix} R \\ 0 \end{bmatrix} \quad \text{for } Q \in \mathbb{K}^{m \times m} \text{ and } R \in \mathbb{K}^{k \times k}$$

can now be computed in place using the LAPACK command dgeqrf. The matrix R is overwritten over the upper triangular portion of the buffer Y, and the matrix Q is represented implicitly as a product of Householder reflectors, which are stored in the strictly lower triangular parts of the Y buffer and the first k entries of tau. See Fig. A.1(b) for illustration.

New columns or blocks of columns can be adjoined to the matrix Y by appending to the buffer Y. The newly added columns are stored as-is, whereas the existing columns have been overwritten by the QR decomposition. One can continue to add columns in this way for as long as desired. See Fig. A.1(c).
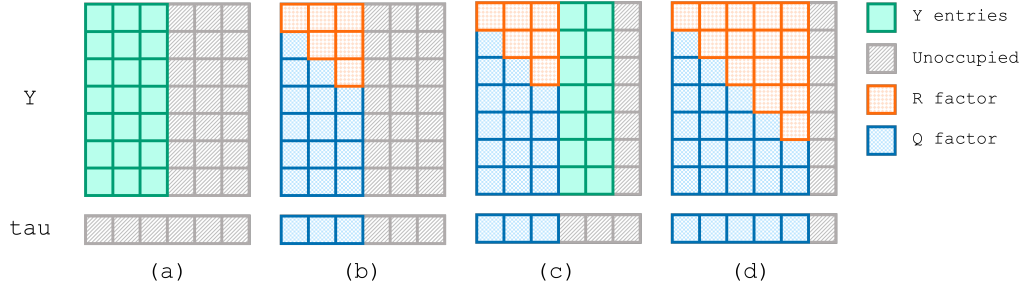
Figure A.1: Data layout for incremental QR decomposition, illustrated with a buffer of size 42 for $\mathtt{Y}$ and size 6 for $\mathtt{tau}$. (a) Initial matrix $Y \in \mathbb{K}^{7\times3}$ stored in first 21 entries of $\mathtt{Y}$ buffer; remaining buffer positions are unoccupied. (b) QR decomposition of Y overwritten on Y in place; upper triangle of Y stores the R factor, and lower triangle (and first three entries of $\mathtt{tau}$ buffer) store the Q factor implicitly using Householder reflectors. (c) Matrix Y is expanded to size 7×5; new columns are appended to the end of the $\mathtt{Y}$ buffer. (d) New columns are brought into QR decomposition format.

Suppose at some future time one wishes to compute a QR decomposition of the matrix Y with its new columns. Denote $Y_1 \in \mathbb{K}^{m\times k}$ the previous matrix with its full QR decomposition

$$Y_1 = Q_1 \begin{bmatrix} R_{11} \\ 0 \end{bmatrix}$$

already computed and stored in place, and denote by $Y_2 \in \mathbb{K}^{m\times\ell}$ the newly added columns, so that $Y = \begin{bmatrix} Y_1 & Y_2 \end{bmatrix}$. As a first step, apply the matrix Q_1^* to Y_2 in place, which can be accomplished using the LAPACK routine $\mathtt{dormqr}$. The result of this operation can be written as

$$Q_1^*Y = Q_1^* \begin{bmatrix} Y_1 & Y_2 \end{bmatrix} =: \begin{bmatrix} R_{11} & R_{12} \\ 0 & \widetilde{Y}_2 \end{bmatrix} \quad \text{for } R_{12} \in \mathbb{K}^{k\times\ell} \text{ and } \widetilde{Y}_2 \in \mathbb{K}^{(m-k)\times\ell}.$$

The matrix Y_2 in the buffer $\mathtt{Y}$ has been overwritten by two matrices, R_{12} and $\widetilde{Y}_2$ stacked on top of each other. The top part of this matrix, R_{12}, shall serve as the upper right block of the R factor of the full matrix Y. The bottom part $\widetilde{Y}_2$ is QR factorized in place, again using the LAPACK routine $\mathtt{dgeqrf}$. (The Q factor requires ℓ additional elements of storage, which are placed into the next ℓ entries of the buffer $\mathtt{tau}$.) Symbolically, this QR decomposition may be written

$$\widetilde{Y}_2 = Q_2 \begin{bmatrix} R_{22} \\ 0 \end{bmatrix} \quad \text{for } Q_2 \in \mathbb{K}^{(m-k)\times(m-k)} \text{ and } R_{22} \in \mathbb{K}^{\ell\times\ell}.$$

Combining, the two previous displays, we have obtained a QR decomposition of

the new Y matrix

$$Y = \begin{bmatrix} Y_1 & Y_2 \end{bmatrix} = Q_1 \begin{bmatrix} I & 0 \\ 0 & Q_2 \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} \\ 0 & R_{22} \\ 0 & 0 \end{bmatrix} =: Q \begin{bmatrix} R \\ 0 \end{bmatrix}.$$

The dimensions are $Q \in \mathbb{K}^{m \times m}$ and $R \in \mathbb{K}^{(k+\ell) \times (k+\ell)}$. We have updated the QR decomposition in place; see Fig. A.1(d) for the data layout at the end of this procedure.

Computational cost. The computational cost of the update procedure consists of $O(mk\ell)$ operations to multiply Y_2 by Q_1^*, and $O(m\ell^2)$ operations to compute the QR decomposition of $\widetilde{Y}_2$ in place. The total cost of updating the QR decomposition of an $m \times k$ matrix with ℓ column appends is $O(m\ell(k + \ell))$ operations.

This incremental QR procedure is very efficient. To see this, suppose we aggregate columns in increments of sizes $\ell_1, \ldots, \ell_t$ resulting in a total of $p = \sum_{i=1}^{t} \ell_i$ columns. Then the total cost of computing the QR decomposition is

$$O\left(\sum_{i=1}^{t} m\ell_i(\ell_i + p) \right) = O(mp^2).$$

The total cost of the incremental QR decomposition, $O(mp^2)$, is the same asymptotically as computing the QR decomposition of an $m \times p$ matrix Y all at once. However, the practical speed of this procedure very much depends on the block sizes ℓ_i; many small column updates are much slower to process than one larger one.

Inverting the R factor. For the applications of the incremental QR decomposition in Part II of this thesis, we need to have access to the *inverse* of the R matrix. While computing the inverse of a matrix explicitly is usually heresy in numerical linear algebra [Hig02, §14.1], this instance may be an exception where matrix inversion is well-justified. (The trace estimates produced by XTRACE do reach machine accuracy—see Fig. 14.1—so using explicit inversion in that application does not appear to resulting in numerical issues!)

The inverse of the R factor can be computed in place using the LAPACK routine `dtrtri`. To perform the update steps, observe that we have the block matrix inverse identity

$$\begin{bmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{bmatrix}^{-1} = \begin{bmatrix} R_{11}^{-1} & -R_{11}^{-1} R_{12} R_{22}^{-1} \\ 0 & R_{22}^{-1} \end{bmatrix}.$$

The inverses R_{11}^{-1} and R_{22}^{-1}, after which the triple product $-R_{11}^{-1}R_{12}R_{22}^{-1}$ can be computed in place using the BLAS routine `dtrmm`.

Working with the Q factor. This implementation presents the Q factor implicitly, represented in the lower triangular portion of the buffer `Y` and the buffer `tau`. For most purposes, this representation is sufficient, as one can compute matrix products with Q and Q^* using the LAPACK routine `dormqr`. If desired, an explicit representation the economy Q factor can be extracted using the LAPACK routine `dorgqr`; the cost is $O(mp^2)$ operations, where p denotes the current number of columns. For computational efficiency, one should make sure to only extract the Q factor only once, when the matrix Y has reached its maximal size.

The design of LAPACK. Many of LAPACK's design decisions, such as the infamous "leading dimension" input for matrix algorithms and the representation of QR decomposition via packed, in-place formats are confusing and unintuitive for new users. The effortlessness by which the incremental QR decomposition primitive can be implemented using LAPACK routines speaks to the wisdom of these aspects of LAPACK. Indeed, the efficient implementation described here would not be possible, or would be much less efficient, without LAPACK's in-place QR decomposition and leading dimension argument, allowing for seamless execution of matrix routines in place on submatrices. As a technical community, we owe a great debt to the designers of well-designed pieces of computational mathematics software like LAPACK.

Implementation. A MATLAB implementation of the incremental QR primitive is provided in Program A.1. Instead of LAPACK routines, this implementation uses undocumented MATLAB functions, which provide an interface to the low-level LAPACK routines. This implementation works in MATLAB 2023b, but their is no guarantee the relevant private functions are not removed or changed in upcoming MATLAB releases. This program defines an `incremental_qr` class, whose usage is demonstrated in the following code segment:

```
iqr = incremental_qr(Y); % Initialize incremental_qr object
iqr.addcols(Ynew);        % Append Ynew to matrix, update QR
Qx = iqr.applyQ(x);       % Compute Q*x
Qtx = iqr.applyQt(x);     % Compute Q'*x
y = iqr.projectOut(x);    % Compute x - Q*Q'*x
Q = iqr.getQ();           % Extract economy Q factor
R = iqr.getR();           % Extrace square R factor
```

Program A.1 hhqr.m. Compute a Householder QR decomposition of input matrix, represented in compact format. Warning: Uses *undocumented internal MATLAB functions*.

```
function [hous,tau] = hhqr(Y)
% Input:  Matrix Y
% Output: Compact representation [hous,tau] for QR factorization of Y

[hous,tau] = matlab.internal.decomposition.compactQR(Y);

end
```

Program A.2 apply_Qt.m. Apply the adjoint of (full) Q matrix for a compactly represented Householder QR decomposition. Warning: Uses *undocumented internal MATLAB functions*.

```
function QtX = apply_Qt(hous,tau,X)
% Input:  Compact representation [hous,tau] for QR factorization,
%         input matrix X
% Output: Product QtX = Q'*X

QtX = matlab.internal.decomposition.applyHouseholder(hous, tau,...
    X, true, size(hous,2));

end
```

Program A.3 get_Q.m. Get the (thin) Q matrix for a compactly represented Householder QR decomposition. Warning: Uses *undocumented internal MATLAB functions*.

```
function Q = get_Q(hous,tau)
% Input:  Compact representation [hous,tau] for QR factorization
% Output: Thin QR factor Q

[m,k] = size(hous);
Q = matlab.internal.decomposition.applyHouseholder(hous, tau,...
    [eye(k);zeros(m-k,k)], false, k);

end
```

Some of the functionality of the incremental_qr class is broken out into subroutines hhqr, apply_Qt, and get_Q, which are provided in Programs A.1 to A.3. These subroutines are used in the bespoke incremental QR implementation in the Householder reflector-based accelerated randomly pivoted QR implementation (Program 9.4).

Program A.1: An implementation of the incremental QR primitive described in Appendix A. Warning: This implementation is based off of *undocumented internal MATLAB functions* that are subject to be removed or change between releases. Subroutines hhqr, apply_Qt, and get_Q are provided in Programs A.1 to A.3.

```matlab
classdef incremental_qr < handle
% Implementation of the incremental QR decomposition of a tall matrix
% with an increasing number of columns

    properties
        H   % Matrix to store evolving QR decomposition
        tau % Vector to store scalars for Householder reflections
        k   % Number of columns in current matrix
        n   % Height of matrix
    end

    methods
        function obj = incremental_qr(Y,varargin)
            % Initialize the QR decomposition with matrix Y
            [obj.H, obj.tau] = hhqr(Y);
            [obj.n,obj.k] = size(Y);
            if ~isempty(varargin) % Optionally: Set size
                obj.H = [obj.H zeros(obj.n,varargin{1}-obj.k)];
            end
        end

        function obj = addcols(obj, Ynew)
            % Add new columns to the existing QR decomposition
            l = size(Ynew,2);

            % Double and copy if necessary
            if obj.k + l > size(obj.H,2)
                obj.H = [obj.H zeros(size(obj.H))];
                obj.tau = [obj.tau;zeros(size(obj.tau))];
                obj.addcols(Ynew);
                return
            end

            Ynew = obj.applyQtfull(Ynew);   % Apply Qt to Ynew
```

```
        Ybot = Ynew(obj.k+1:end,:);      % Extract bottom of Ynew


        % Form QR decomposition of bottom
        [Ynew(obj.k+1:end,:),obj.tau(obj.k+1:obj.k+l)]...
            = hhqr(Ybot);


        obj.H(:,obj.k+1:obj.k+l) = Ynew; % Write to buffer
        obj.k = obj.k + l; % Increase size of k
    end


    function Qx = applyQ(obj, x)
        % Apply the Q matrix to a vector x
        Qx = matlab.internal.decomposition.applyHouseholder(...
            obj.H, obj.tau, [x;zeros(obj.n-obj.k,...
            size(x,2))], false, obj.k);
    end


    function Qtx = applyQt(obj, x)
        % Apply the transpose of the Q matrix to a vector x
        Qtx = apply_Qt(obj.H.obj.tau,x);
        Qtx = Qtx(1:obj.k,:);
    end


    function y = projectOut(obj, x)
        % Compute y = (I-Q*Q')*x
        y = apply_Qt(obj.H.obj.tau,x);
        y(1:obj.k,:) = 0;
        y = matlab.internal.decomposition.applyHouseholder(...
            obj.H, obj.tau, y, false, obj.k);
    end


    function Q = getQ(obj)
        % Return the Q matrix from the current QR decomposition
        Q = get_Q(obj.H,obj.tau);
    end


    function R = getR(obj)
        % Return the R matrix from the current QR decomposition
```

```matlab
            R = triu(obj.H);  % Extract the upper triangular part
            R = R(1:obj.k,1:obj.k);
        end
    end
end
```

A p p e n d i x   B

WHICH SKETCH SHOULD I USE?

> *Our implementation of CQRRPT uses structured sparse sketching operators since these can provide exceptional speed without sacrificing reliability of the algorithm.*
>
> Maksim Melnichenko, Oleg Balabanov, Riley Murray, James Demmel, Michael W. Mahoney, and Piotr Luszczek, *CholeskyQR with randomization and pivoting for tall matrices (CQRRPT)* [MBMD+25]

This appendix surveys the various constructions sketching matrices and compares them. Echoing [DM23b], my conclusion is that, among the available options, *sparse sign embeddings* (Appendix B.6) are the fastest and most reliable for general-purpose use.

Sources. This appendix is a significantly expanded version of a blog post I wrote on this topic [Epp23c], and it is informed by my numerical experience writing the papers [Epp24b; EMN25]. The original inspiration for this line of research was the numerical experiments of [DM23b], which first demonstrated to me the significant speed advantage of sparse random embeddings over alternatives.

Outline. Appendix B.1 begins by describing, in more detail, the kind of properties we desire for sketching matrices to be useful for applications in matrix computations. The next sections provide a tour through standard options for sketching matrices: Gaussian embeddings (Appendix B.2), iid embeddings (Appendix B.3), subsampled randomized trigonometric transforms (Appendix B.4), and iid sparse embeddings (Appendix B.5). Each of these constructions is seen to have both merits and demerits. We conclude our tour with sparse sign embeddings in Appendix B.6, which we observe to have a balance of speed and accuracy (i.e., low distortion) not matched by other types of dimensionality maps. We end with some concluding thoughts in Appendix B.7 and a postscript on recent developments in Appendix B.8.

B.1 What properties do we want sketching matrices to have?

Before studying different types of sketching matrices, let us revisit the question: What properties does a sketching matrix need for use in matrix computations?

This section answers this question in three subsections. First, we redefine the subspace embedding to allow different "upper" and "lower" distortion parameters, and we investigate the asymmetric importance of these parameters. Next, we discuss the distribution of singular values for a sketched matrix. Finally, we discuss the differences between subspace embeddings and Johnson–Lindenstrauss embeddings.

Redefining subspace embeddings and the sketching asymmetry principle

In Chapter 21, we defined a subspace embedding $S \in \mathbb{K}^{m \times d}$ for a matrix $F \in \mathbb{K}^{m \times n}$ as a matrix for which

$$(1 - \eta)\|Fz\| \leq \|S^*(Fz)\| \leq (1 + \eta)\|Fz\| \quad \text{for all } z \in \mathbb{K}^n. \tag{B.1}$$

The parameter $\eta \in [0, 1]$, called the *distortion*, measures the quality of the embedding. This definition is somewhat lacking in that it uses the same parameter η both upper and lower bound $\|S^*(Fz)\|$. In this appendix, we will use the following two-parameter version of the subspace embedding property:

Definition B.1 (Subspace embeddings, again). A matrix $S \in \mathbb{K}^{m \times d}$ is said to be a *subspace embedding* for a matrix $F \in \mathbb{K}^{m \times n}$ with *lower distortion* $\eta_- \in [0, 1)$ and *upper distortion* $\eta_+ \geq 0$ provided that

$$(1 - \eta_-)\|Fz\| \leq \|(S^*F)z\| \leq (1 + \eta_+)\|Fz\| \quad \text{for every } z \in \mathbb{K}^n.$$

If $\eta := \max\{\eta_-, \eta_+\} < 1$, we say that S is a subspace embedding for F with *distortion* η.

This simple change in definition reframes our perspective on what properties we should desire for a sketching matrix, and we have new versions of Theorem 21.8 and Proposition 21.12 consistent with this new definition:

Theorem B.2 (Sketch-and-solve, again). *Let S be a subspace embedding for $[B \ \ c]$ with distortions η_-, η_+, and let $\widehat{x}$ be the sketch-and-solve solution. Then*

$$\|c - B\widehat{x}\| \leq \max\left\{\frac{1 + \eta_+}{1 - \eta_-}, \frac{4.5 \max\{\eta_+, \eta_-\}^2}{(1 - \eta_-)^4}\right\} \cdot \|c - Bx\|.$$

Proposition B.3 (Randomized preconditioning, again). *Let S be a subspace embedding for B with distortions η_-, η_+. Construct the sketched matrix S^*B, and let*

$$S^*B = UM$$

be any orthonormal decomposition of S^*B, as in Proposition 21.12. Then

$$\sigma_{\max}(BM^{-1}) \le \frac{1}{1 - \eta_-}, \quad \sigma_{\min}(BM^{-1}) \ge \frac{1}{1 + \eta_+}, \quad \mathrm{cond}(BM^{-1}) \le \frac{1 + \eta_+}{1 - \eta_-}.$$

We see that the bounds on the sketch-and-solve residual norm and the condition number diverge as $\eta_- \uparrow 1$, but remain bounded when $\eta_+ \uparrow 1$. This demonstrates a fundamental asymmetry in the importance of the lower and upper distortions:

> **The sketching asymmetry principle.** For most applications of sketching in linear algebra, the most important requirement for a sketching matrix is that the *lower distortion is bounded away from 1*, e.g., $\eta_- \le 0.9$. Sketching algorithms typically produce meaningful results even with weak control on the upper distortion η_+, say, $\eta_+ \le O(\log n)$.

The sketching asymmetry principle is a recent perspective, as most foundational works on sketching and subspace embeddings are concerned with establishing the subspace embedding property with a single distortion parameter η, which is sometimes defined differently than we have (e.g., [CDDR24, Def. 1.1]). The sketching asymmetry principle was developed and advocated for in recent work of Joel Tropp [Tro25], who suggests the name *subspace injection* for a sketching matrix satisfying $\eta_- < 1$.

The distribution of singular values

As the name suggests, a subspace embedding for F preserves the lengths of all vectors in the *subspace* range(F). The distortions η_-, η_+ describe the minimum $(1 - \eta_-)$ and maximum $(1 + \eta_+)$ factors by which S^* can rescale vectors in this subspace. As the following result shows, these factors must always bound an interval containing there singular values of the matrix S^*Q, where $Q = \mathbf{Orth}(F)$:

Proposition B.4 (Singular values to subspace embedding)**.** *Instate the notation of Proposition B.3. Let $F \in \mathbb{K}^{m \times n}$ be a matrix, and let Q be* any *matrix whose columns for an orthonormal basis for* range(F). *Then $S \in \mathbb{K}^{m \times d}$ is a subspace embedding for F with distortions η_-, η_+ if and only if*

$$1 - \eta_- \le \sigma_{\min}(S^*Q) \le \sigma_{\max}(S^*Q) \le 1 + \eta_+.$$

This result is standard, and its proof is straightforward; see, e.g., [KT24, Prop. 5.2].

Can the *distribution* of the singular values of S^*Q tell us anything about sketching in matrix computations? Perhaps. We have the following result:

Proposition B.5 (Singular values and randomized preconditioning)**.** *Instate the notation of Proposition B.3. The singular values of the preconditioned matrix BM^{-1} are the reciprocals of the singular values of S^*Q.*

Proof. Let $B = Q(B^*B)^{1/2}$ be the polar decomposition, and let $S^*B = UM$ be any orthonormal decomposition. Now, taking a polar decomposition $M = V(B^*S^*SB)^{1/2}$, we have

$$BM^{-1} = Q(B^*B)^{1/2}(B^*SS^*B)^{-1/2}V^*.$$

Therefore, the squared singular values of BM^{-1} are

$$\sigma(BM^{-1})^2 = \lambda\left((B^*B)^{1/2}(B^*SS^*B)^{-1}(B^*B)^{1/2}\right).$$

(Recall that we extend nonlinear functions to vectors entrywise.) Since $S^*B = S^*Q(B^*B)^{1/2}$, we have the simplification:

$$\sigma(BM^{-1})^2 = \lambda\left((Q^*SS^*Q)^{-1}\right) = \sigma(S^*Q)^{-2}.$$

Conclude by observing that the eigenvalues of $(Q^*SS^*Q)^{-1}$ are the inverses of the eigenvalues of Q^*SS^*Q, which are the inverse-squares of the singular values of S^*Q. $\qquad\square$

This result shows that the singular values of S^*Q determine the singular values of the preconditioned matrix BM^{-1}. In principle, knowing the singular values of BM^{-1} can lead to bounds on the convergence of sketch-and-precondition-type algorithms that are more precise than the simple bounds we derived in Corollary 21.13. See [Gre97b, §3.1] for a discussion of such *spectrum-adaptive* bounds for Krylov iterative methods. (Be aware that these spectrum-adaptive are valid in exact arithmetic but often fail in finite-precision arithmetic.)

In my experience, I understanding the precise distribution of the singular values S^*Q is usually not *that* informative for studying the behavior of linear algebraic algorithms. Still, plotting the distribution of singular values of S^*Q can be a valuable way of understanding the behavior of sketching matrices, particularly those that *lack* the subspace embedding property for given parameters η_+, η_-. Are there just a few stray singular values that escape the interval $[1 - \eta_-, 1 + \eta_+]$? A small clump? Or

does the bulk of singular values extend past the endpoints of the interval? These are all different ways an embedding can fail to have the subspace embedding property, and visualizing the singular values can help diagnose *how* a sketching matrix fails to have the subspace embedding property.

Subspace embeddings vs. Johnson–Lindenstrauss embeddings

In matrix computations, the term "sketching" is generally accepted to have a very broad definition. To some, *any* form of *linear* dimensionality reduction is an instance of sketching. Typically, we expect that the sketching process should "preserve lengths" of the matrix or vectors being sketched. The subspace embedding property (Definition B.1) is one way of formalizing this length-preservation property, but there are others. In particular, we have the following standard definition:

Definition B.6 (Johnson–Lindenstrauss embedding)**.** A matrix $S \in \mathbb{K}^{m \times d}$ is said to be a *Johnson–Lindenstrauss* embedding with distortion $\eta \in [0, 1)$ for a *finite* subset $\mathsf{E} \subseteq \mathbb{K}^m$ if

$$(1 - \eta)\|v\| \leq \|S^*v\| \leq (1 + \eta)\|v\| \quad \text{for every } v \in \mathsf{E}.$$

The Johnson–Lindenstrauss lemma [JL84] states that, e.g., Gaussian embeddings (Appendix B.2) are *oblivious* Johnson–Lindenstrauss embeddings with embedding dimension $d = O(\eta^{-2} \log |\mathsf{E}|)$ (with failure probability, say, $1/n$) [Woo14, Lem. 18]. We see the embedding dimension need only be *logarithmic* in the *cardinality* of the set of points E.

Both the Johnson–Lindenstrauss embedding property and subspace embedding property are useful conditions for a sketching matrix to satisfy. But it is important not to confuse them. Johnson–Lindenstrauss embeddings preserve lengths of vectors in a *finite set* E and the embedding dimension $d \sim (\log |\mathsf{E}|)/\eta^2$ must be *logarithmic* in the *cardinality* $|\mathsf{E}|$. Subspace embeddings preserve lengths of vectors in a *finite-dimensional subspace* $\operatorname{range}(F)$ and the embedding dimension $d \sim \dim \operatorname{range}(F)/\eta^2$ must be *linear* in the *dimension* $\dim \operatorname{range}(F)$. In Part III of this thesis, we are exclusively concerned with the subspace embedding property.

B.2 Gaussian embeddings

To understand sketching, the best place to start is with Gaussian embeddings, which admit a beautiful and mathematically precise theory. While Gaussian embeddings are often less computationally efficient than other types of embeddings, they set the standard to which other types of dimensionality reduction maps will aspire.

Definition B.7 (Gaussian embedding). A *Gaussian embedding* (over $\mathbb{K}$) is a matrix $S \in \mathbb{K}^{m \times d}$ with iid $\text{Normal}_{\mathbb{K}}(0, 1/d)$ entries.

The Gaussian distribution has many beautiful properties. These properties can be applied to give very sharp quantitative analysis of Gaussian embeddings. In particular, we will employ the following fundamental fact:

> For any matrix $Q \in \mathbb{K}^{m \times n}$ with orthonormal columns and a Gaussian matrix $S \in \mathbb{K}^{m \times d}$ over $\mathbb{K}$, the sketch $S^* Q \in \mathbb{K}^{d \times n}$ has iid $\text{Normal}_{\mathbb{K}}(0, 1/d)$ entries.

On the basis of this result, analysis of Gaussian embeddings reduces to questions about the singular values of rectangular matrices with iid Gaussian entries, which are among the most well-studied in random matrix theory.

Asymptotic theory. In the large-data limit $n, d \to \infty$, the behavior of Gaussian embeddings is captured by standard asymptotic results from random matrix theory. We have the following result:

Fact B.8 (Gaussian embeddings, distribution of singular values: large-data limit).
Consider a family of matrices $Q_i \in \mathbb{K}^{m_i \times n_i}$ and Gaussian embeddings $S_i \in \mathbb{K}^{m_i \times d_i}$ over $\mathbb{K}$ with dimensions $m_i, n_i, d_i \to \infty$ tending to infinity and with limiting aspect ratio $n_i / d_i \to \varrho \in (0, 1)$. Then the distribution of singular values of $S_i^* Q_i$ converges (weakly in probability) to the Marčenko–Pastur singular value distribution, *a continuous probability distribution on $\mathbb{R}_+$ with density*

$$f_{\text{mp}}(\sigma) = \frac{\sqrt{(\sigma_+^2 - \sigma^2)_+ (\sigma^2 - \sigma_-^2)_+}}{\pi \varrho \sigma}.$$

Here,

$$\sigma_\pm = 1 \pm \sqrt{\varrho}$$

and $a_+ := \max\{a, 0\}$ denotes the positive part of a real number.

Fact B.8 is a deep fact of random matrix theory, proven (in a somewhat different form) in Marčenko and Pastur's seminal paper [MP67]. Its proof requires a significant mathematical apparatus.

This result demonstrates that the *distribution* of singular values converges to a continuous distribution supported on the interval $[\sigma_-, \sigma_+]$, but it does not rule out the possibility of a few extraneous singular values exiting this interval. The following result [BY93] forbids it.
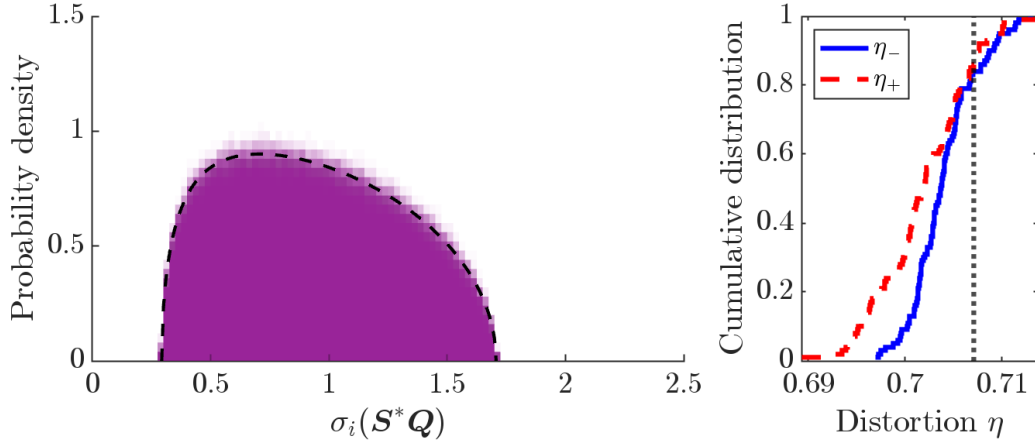
Figure B.1: *Left:* Superimposed singular value distributions $\boldsymbol{\sigma}(\boldsymbol{S}^*\boldsymbol{Q})$ for 100 independent realizations of a Gaussian embedding applied to a matrix $\boldsymbol{Q} \in \mathbb{R}^{m \times n}$ with dimension $n = 1000$. The embedding dimension is $d = 2n$. The Marčenko–Pastur singular value distribution is shown as a black dashed line for reference. *Right:* Empirical cumulative distribution function of lower and upper distortions. The asymptotic limiting value $\eta = \sqrt{n/d}$ is shown in black.

Fact B.9 (Gaussian embeddings, extreme singular values: large-data limit). *Assume the setting of Fact B.8 and assume in addition that $\boldsymbol{S}_i$ are independent. Then $\sigma_{\max}(\boldsymbol{S}_i^*\boldsymbol{Q}_i) \to \sigma_+$ and $\sigma_{\min}(\boldsymbol{S}_i^*\boldsymbol{Q}_i) \to \sigma_-$, almost surely as $i \to \infty$. In particular, the upper and lower distortions η_+, η_- both converge almost surely to $\sqrt{\varrho} = \sqrt{n/d}$.*

Rearranging the relation $\eta \approx \sqrt{n/d}$, we derive the relation

$$d \approx n/\eta^2.$$

This gives us guidance for how to set the embedding dimension d for a Gaussian embedding to achieve a desired distortion η.

Empirical results. The asymptotic theory for Gaussian embeddings accurately predicts the empirical results. The left panel of Fig. B.1 shows the distribution of singular values $\boldsymbol{\sigma}(\boldsymbol{S}^*\boldsymbol{Q})$ for a Gaussian embedding over 100 independent random trials. We set $n := 1000$ and $d := 2n$. (The value of m and the matrix $\boldsymbol{Q}$ are immaterial, owing to the boxed fact above.) Each trial is plotted as a single transluscent histogram, and the histograms for all 100 trials are superimposed. The empirically computed singular value distributions show remarkable fidelty to the Marčenko–Pastur singular value distribution, marked as a dotted black line. Not one of the 100 random realizations shows a significant excursion from the the bounds of the Marčenko–Pastur density curve.

The right panel plots the cumulative distribution of the distortion parameters η_+ and η_- for these 100 trials The distortion parameters are consistently close to the almost-sure limit value $\eta = \sqrt{n/d} = 1/\sqrt{2}$, marked as a dotted vertical line.

Nonasymptotic results. In practice, we do not apply sketching matrices to a series of problems asymptotically growing to infinity; we apply sketching to a given problem of a fixed size. As such, we value theoretical results that predict the performance of Gaussian embeddings for a specific problem size.

Fact B.10 (Gaussian embeddings, extreme singular values: nonasymptotic results)**.** *Let $S \in \mathbb{R}^{m \times d}$ be a* real *Gaussian embedding, let $B \in \mathbb{R}^{m \times n}$ be a* real *matrix, and let η_+, η_- be the distortions. Then*

$$\mathbb{E}[\eta_+], \mathbb{E}[\eta_-] \leq \sqrt{n/d}.$$

Furthermore, we have the tail bound

$$\mathbb{P}\left\{\eta_+ \geq \sqrt{n/d} + t\right\} \leq \exp\left(-\frac{dt^2}{2}\right) \quad and \quad \mathbb{P}\left\{\eta_- \geq \sqrt{n/d} + t\right\} \leq \exp\left(-\frac{dt^2}{2}\right).$$

This result is due to [DS01]. See [AS17, Cor. 6.38] for an exposition and [MT20, Thm. 8.4] for a generalization. It's proof requires many techniques. The bound on $\mathbb{E}[\eta_+]$ is a direct consequence of Chevet's theorem [Che78], and the bound on $\mathbb{E}[\eta_-]$ follows from Gordon's inequalities [Gor85]. The tail bounds follow from concentration for Lipschitz continuous functions of Gaussian random variables [van14, Thm. 3.25]. I recommend the references [van14; AS17; Ver18; Tro21] for learning about this family of ideas. A similar (but slightly worse) result is also known for complex Gaussian embeddings [AS17, Prop. 6.3.3].

Universality. As we will see in the rest of this chapter, the most effective random embeddings tend to exhibit performance similar to Gaussian embeddings, a manifestation of the probabilistic phenomenon of universality. Providing mathematical support to this type of universality phenomenon has been an active subject for research (see, e.g., [OT18; DL19; Bv24; CDD25a; CDDR24; Tro25]), but even the best-known mathematical tools are not precise enough to fully explain empirically observed universality phenomena. Notwithstanding these theoretical gaps, Martinsson and Tropp recommend using the Gaussian theory to inform parameter decisions [MT20, §9.7]:

> Even so, we would like to have a priori predictions about how our algorithms will behave. Beyond that, we need reliable methods for se-

lecting algorithm parameters, especially in the streaming setting where
we cannot review the data and repeat the computation.

Here is one answer to these concerns. As a practical matter, we can
simply invoke the lessons from the Gaussian theory, even when we are
using a different type of random embedding. The universality result [of
[OT18]] gives a rationale for this approach in one special case. We also
recommend undertaking computational experiments to verify that the
Gaussian theory gives an adequate description of the observed behavior
of an algorithm.

I fully agree with this viewpoint, and the experiments provided in this chapter
should provide insights to practitioners interested in the differences between types
of random embeddings.

B.3 IID embeddings

One simple way of constructing a non-Gaussian sketching matrix is to replace the
Gaussian distribution with another random distribution, such as the (scaled) random
sign distribution $\mathrm{U_{NIF}}\{\pm d^{-1/2}\}$. As long as the new distribution has mean zero,
variance $1/d$, and is sufficiently well-behaved (e.g., exhibits rapidly decaying sub-
gaussian tails), the performance is generally comparable to a Gaussian embedding.
Figure B.2 shows the behavior of a sketching matrix with iid $\mathrm{U_{NIF}}\{\pm d^{-1/2}\}$ entries
applied to the matrix

$$Q := \begin{bmatrix} \mathbf{I}_n \\ \mathbf{0} \end{bmatrix} \in \mathbb{R}^{m \times n} \quad \text{for } m := 10^5, \ n := 10^3; \tag{B.2}$$

we use embedding dimension $d = 2n$. (This matrix presents a hard case for certain
other sketching matrices, as we will see below.) The results are essentially identical
to those for Gaussian embeddings in Fig. B.1, providing a clear demonstration of
the universality phenomenon.

B.4 Subsampled trigonometric transforms

Subsampled randomized trigonometric transforms (SRTTs) are widely viewed as
among the most effective types of sketching matrices for use in matrix computations.
Speaking to other researchers, one hears claims such as "SRTTs behave exactly like
Gaussian embeddings". Do these claims hold up to scrutiny? We shall see.
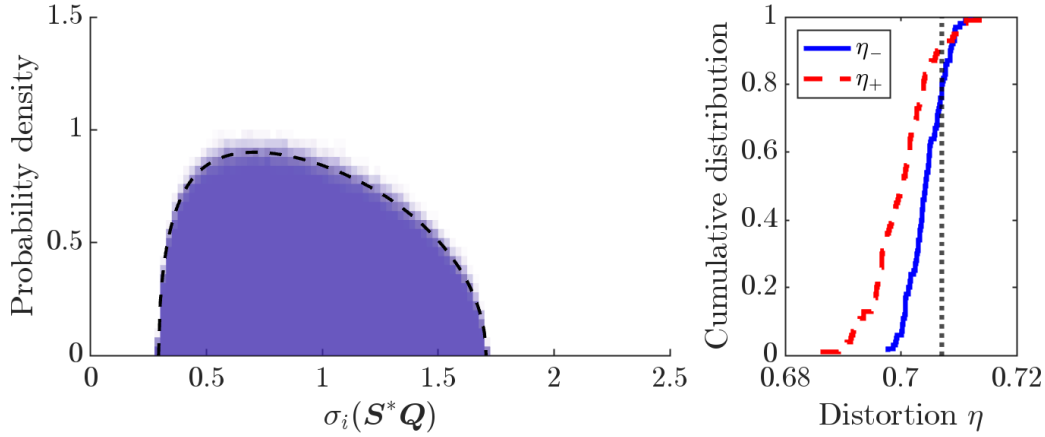
Let us begin with the definition:

Figure B.2: *Left:* Superimposed singular value distributions $\boldsymbol{\sigma}(\boldsymbol{S}^*\boldsymbol{Q})$ for 100 independent realizations of a iid scaled sign embedding applied to the adversarial matrix (B.2) with embedding dimension $d = 2n$. The Marčenko–Pastur singular value distribution is shown as a black dashed line for reference. *Right:* Empirical cumulative distribution function of lower and upper distortions. The asymptotic limiting value $\eta = \sqrt{n/d}$ is shown in black.

Definition B.11 (Subsampled randomized trigonometric transform)**.** A (standard) *subsampled randomized trigonometric transform* is a sketching matrix $\boldsymbol{S} \in \mathbb{K}^{m \times d}$ defined as a scaled product of three matrices.

$$\boldsymbol{S}^* = \sqrt{\frac{m}{d}}\boldsymbol{RFD}.$$

These matrices have the following definitions:

- $\boldsymbol{D} = \mathbf{Diag}(\boldsymbol{\varepsilon})$ is a diagonal matrix with entries $\varepsilon_1, \ldots, \varepsilon_m$ drawn iid from either the random sign $\mathrm{U_{NIF}}\{\pm 1\}$ or random phase $\mathrm{U_{NIF}}\,\mathbb{T}(\mathbb{C})$ distributions.

- $\boldsymbol{F}$ is a unitary fast trigonometric transform for which $\boldsymbol{Fz}$ can be computed in $O(m \log m)$ operations. Examples include the discrete Fourier transform, the (type-II) discrete cosine transform, the (Walsh–)Hadamard transform, or the Hartley transform.

- $\boldsymbol{R}$ is a restriction to d coordinates, sampled uniformly *without replacement*.

This construction is due to Ailon and Chazelle [AC06] (see also [WLRT08]). SRTTs are often called SRFTs or SRHTs when $\boldsymbol{F}$ is the discrete Fourier transform or (Walsh–)Hadamard transform. In fact, the name SRFT is often used even when with other types of trigonometric transforms. In this work, we exclusively use the

discrete cosine transform as our trigonometric transform F and the random signs distribution for D.

Runtime. The operation count for sketching using SRTTs is smaller than for Gaussian embeddings. Since we use a *fast* trigonmetric transform F, the sketch $S^*v = \sqrt{n/d} \cdot R(F(Dv))$ can be computed in $O(m \log m)$ operations. Compare with $O(dm)$ operations for Gaussian embeddings. In principle, the runtime for SRTTs can be reduced to $O(m \log d)$ operations [WLRT08].

Fast trigonometric transforms are notoriously difficult to implement [Van92]. Even with a world-class implementation, the computational throughput of fast trigono-metric transforms (as measured by floating point operations per second) is much less than for matrix multiplication. For sufficiently large problems, SRTTs are sub-stantially faster than Gaussian embeddings, but the speedups can be less impressive than one might hope. In my experience, sparse sketching matrices (Appendices B.5 and B.6) are typically significantly faster than SRTTs.

Analysis. Guarantees for the SRHT were proven by Tropp [Tro11]. Here is a simplified version of his result:

Fact B.12 (Standard SRHTs: Subspace embedding property)**.** *Standard SRHTs of size $m \times d$ are oblivious subspace embeddings for dimension n with distortion $\eta = 0.6$ and failure probability $\delta = O(1/n)$ provided that*

$$d \geq \text{const} \cdot (n + \log m) \log n.$$

The full versions of Tropp's results have explicit constants. The analysis of [Tro11] can be extended to prove subspace embedding bounds for general distortions $\eta > 0$ (such a resulted is stated in [Woo14, Thm. 7]) and adapted to the random phase distribution for D or other types of trigonometric transforms.

Tropp's result suggests (correctly) that an embedding dimension of $d \sim n \log n$ is *necessary* for standard SRTTs to achieve the subspace embedding property (for constant $\eta < 1$). Compare with Gaussian embeddings, which allow $d \sim n$, with no need for a logarithmic oversampling factor.

Bad examples. If we want to use sketching matrices as primitives in general-purpose software, then we must demand that they work reliably when applied to *any* input matrix. As such, even a single bad example can dissuade us from using a particular type of dimensionality reduction map.
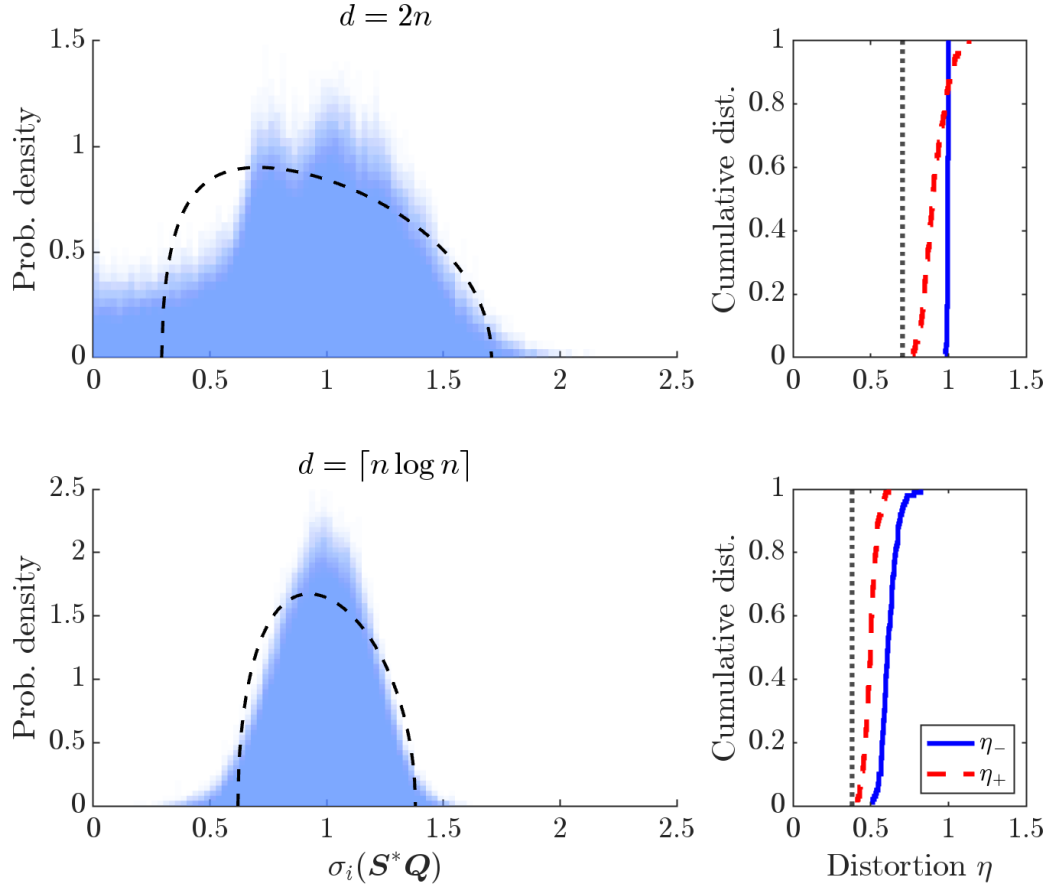
Figure B.3: *Left:* Superimposed singular value distributions $\sigma(S^*Q)$ for 100 independent realizations of standard SRTT applied to the adversarial matrix (B.2) with embedding dimension $d = 2n$ (*top*) and $d = \lceil n \log n \rceil$ (*bottom*). The Marčenko–Pastur singular value distribution is shown as a black dashed line for reference. *Right:* Empirical cumulative distribution function of lower and upper distortions. The asymptotic limiting value $\eta = \sqrt{n/d}$ is shown in black.

With that context, we evaluate the adversarial example matrix Q defined in (B.2). Results with embedding dimension $d = 2n$ are shown in the top panels of Fig. B.3. We see that, on this difficult example, the embedding has lower distortion $\eta_- \approx 1$ with high probability. Thus, the standard SRTT *with $d \sim n$* is not effective for this type of matrix.

Consistent with Fact B.12, we may remedy this issue by setting $d \sim n \log n$. The bottom panels of Fig. B.3 show the same experiments with $d = \lceil n \log n \rceil$. We now see that we reliably obtain an embedding with $\eta < 0.9$, although the singular value statistics still deviate significantly from the appropriate Marčenko–Pastur singular value distribution.

One can also confirm the failures of standard SRTTs with $d \ll n \log n$ theoretically. The following result appears in [Tro11, §3.3]:

Fact B.13 (Standard SRHTs: Subspace embedding property). *For $n \geq 1$, construct the orthnormal matrix $\mathbf{Q} := \mathbf{I}_n \otimes \mathbf{e}_1 \in \mathbb{R}^{n^2 \times n}$, and consider the action of a standard SRHT with embedding dimension d on $\mathbf{Q}$. Then $\eta_- = 1$ with high probability unless $d \geq \Omega(n \log n)$.*

This failure mode is a consequence of the coupon collector phenomenon [MR95, §3.6]; see [Tro11, §3.3] for details.

The bad examples for the SRTT are unfortunate, and their existence appears to not be widely known. SRTTs are routinely used with a small embedding dimension (say, $d = 2n$ and $d = 3n$), accompanied implicitly or explicitly by claims that they are "just as good a Gaussian embedding".

Rerandomization. The performance of subsampled randomized trigonometric transforms can be greatly improved by adding an additional layer of randomness. We make the following definition:

Definition B.14 (Rerandomized subsampled randomized trigonometric transform). A *rerandomized subsampled randomized trigonometric transform* is a sketching matrix $\mathbf{S} \in \mathbb{K}^{m \times d}$ defined so that

$$\mathbf{S}^* = \sqrt{\frac{m}{d}} \mathbf{R} \mathbf{F} \mathbf{D}_1 \mathbf{F} \mathbf{D}_2,$$

where $\mathbf{F}$ and $\mathbf{R}$ are as in Definition B.11 and $\mathbf{D}_1$ and $\mathbf{D}_2$ are *independent* random diagonal matrices populated with random signs or random phases.

A version of the rerandomization trick using random Givens rotations appears in [RT08]. The Blendenpik software from the paper [AMT10] contains a version of rerandomization more similar to Definition B.14. Another version of rerandomization is advocated in the paper [TYUC17a].

Empirically, the rerandomized SRTT appears to achieve the touted "just as good as Gaussian" performance of SRTTs. Figure B.4 shows the results of rerandomized SRTTs applied to the adversarial matrix (B.2). The singular values are observed to follow Marčenko–Pastur statistics, and the distortion is quite similar to Gaussian embeddings—or even slightly better.
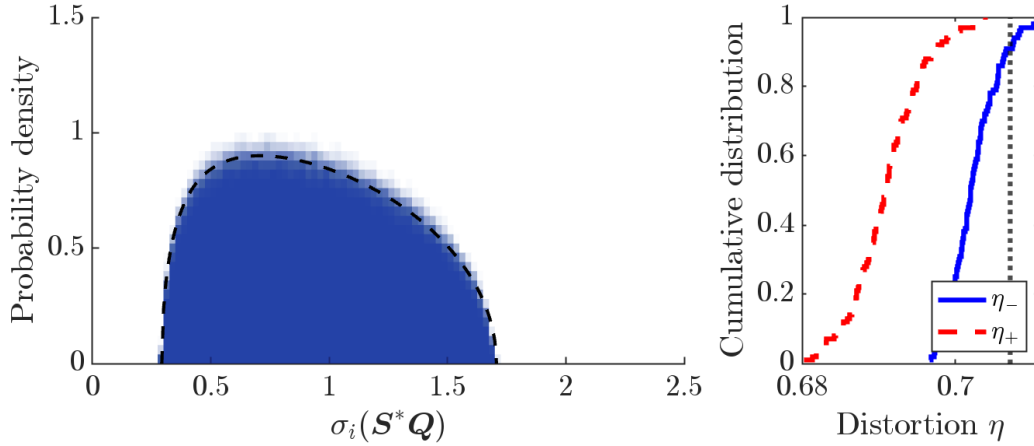
Figure B.4: *Left:* Superimposed singular value distributions $\sigma(S^*Q)$ for 100 independent realizations of a rerandomized SRTT applied to the adversarial matrix (B.2) with embedding dimension $d = 2n$. The Marčenko–Pastur singular value distribution is shown as a black dashed line for reference. *Right:* Empirical cumulative distribution function of lower and upper distortions. The asymptotic limiting value $\eta = \sqrt{n/d}$ is shown in black.

Unfortunately, theoretical guarantees for the near-Gaussian behavior of rerandomized SRTTs are not known. This is a natural subject for future work.

Remark B.15 (Adding even more randomness)*.* Another variant of SRTTs, adding further randomness, is to use random *signed permutation matrices* instead of diagonal matrices. This idea can mollify, but not fix, the issues with the standard SRTT (Definition B.11). It is also compatible with the rerandomized SRTT (Definition B.14), but I am unaware of an example where this trick substantially improves on rerandomized SRTTs. ◇

Implementation. A MATLAB class implementing SRTTs is shown in Program B.1. We use random signs for the diagonal matrices and discrete cosine transforms as the fast trigonometric transform. The first arguments to the constructor are the dimensions d and m, and the third argument is the number of rounds of randomization, set to 1 for standard SRTTs (Definition B.11) or 2 for rerandomized SRTTs (Definition B.14). We have overloaded the multiplication routine (`mtimes`), so S^*v can be computed as `St * v`.

B.5 IID sparse embeddings

With rerandomization, SRTTs appear to have distortions and singular value distributions that closely match Gaussian embeddings, even when applied to challenging instances. Despite their $O(m \log m)$ runtimes, these embeddings are often not as

Program B.1 `srtt.m`. Class implementing subsampled randomized trigonmetric transforms. The argument `rounds` sets the number of rounds of randomization (1 for standard SRTTs, 2 for rerandomized SRTTs). The subroutine `random_signs` is provided in Program F.2.

```
classdef srtt
    properties
        signs  % Matrix of random signs
        idx    % Indexes for subsampling
    end

    methods
        function St = srtt(d, m, rounds)
            St.signs = random_signs(m, rounds);  % Generate signs
            St.idx = randsample(m, d, false);    % Subsample indices
        end

        function y = mtimes(St, x)
            for i = 1:size(St.signs,2)
                x = dct(St.signs(:,i) .* x);      % Random trig trans
            end
            y = x(St.idx,:);                      % Subsample
            y = sqrt(length(x)/length(y)) * y;    % Rescale
        end
    end
end
```

fast as one would hope; see Fig. B.7.

Another way to obtain a sketching matrix with an efficient apply operation $B \mapsto S^* B$ is through *sparse* constructions. In this section, we will explore one class of sparse random embeddings with *iid* entries. These embeddings already have some appealing properties, and they are (comparatively) easy to analyze. In next section, we will consider another class of sparse random embeddings that have even better properties. We make the following definition:

Definition B.16 (IID sparse embedding). An *iid sparse embedding* $S \in \mathbb{R}^{m \times d}$ with *expected sparsity* $\zeta \in [0, d]$ is a random matrix with iid entries drawn from the *scaled sparse sign distribution*

$$
s = \begin{cases}
+\zeta^{-1/2} & \text{with probability } \zeta/2d, \\
0, & \text{with probability } 1 - \zeta/d, \\
-\zeta^{-1/2} & \text{with probability } \zeta/2d.
\end{cases}
$$

As the name suggests, the parameter ζ is equal to the expected number of nonzero entries in each row of S. As such, the expected number of nonzero entries in the entire matrix is $\text{nnz}(S) = \zeta m$, and the expected fraction of nonzero entries is $\text{nnz}(S)/md = \zeta/d$.

Runtime and implementation. We will not focus on how to implement and analyze the runtime of iid sparse embeddings, because sparse sign embeddings (Appendix B.6) are more effective in practice.

IID sparse maps can be generated efficiently by (1) randomly generating the number $\text{nnz}(S)$ (which has a binomial distribution with parameters nd and ζ/d), (2) generating the positions of those entries, and (3) then generating the values $\pm\zeta^{-1/2}$. There are several ways to implement step (2); one way is to randomly generate random positions $(i, j) \sim \text{UNIF}(\{1, \ldots, d\} \times \{1, \ldots, m\})$ and add them to a hashtable until $\text{nnz}(S)$ distinct positions are found. (In the unusual case where $\text{nnz}(S) \geq md/2$, one should instead select random positions to *disinclude*). The expected runtime is $O(\zeta m)$. Once generated, iid sparse maps can be applied to a vector in $O(\text{nnz}(S)+d)$ operations, so the expected runtime is $O(\zeta m + d)$ operations.

Analysis. The subspace *injection* properties of iid sparse embeddings were studied in a recent paper of Tropp [Tro25, Thm. 6.3]. Here is a simplified version of his result:

Fact B.17 (IID sparse embeddings: Subspace injection). *Let $B \in \mathbb{R}^{m \times n}$ be a* real *matrix, and fix $\eta_- > 0$. An iid sparse embeddings $S \in \mathbb{R}^{m \times d}$ with expected sparsity ζ satisfies the (oblivious) subspace injection property*

$$\|S^*(Bz)\| \geq (1 - \eta_-)\|Bz\| \quad \text{for every } z \in \mathbb{R}^n$$

with probability at least $1 - \delta$ provided that

$$d \geq \frac{16 \max\{n, 6 \log(2n/\delta)\}}{\eta_-^2} \quad \text{and} \quad \zeta \geq \frac{32 \log(n/\delta)}{\eta_-^2}.$$

This result shows that an embedding dimension of $d \sim n$ and sparsity $\zeta \sim \log n$ to obtain a nontrivial (lower) distortion $\eta_- < 1$. Tropp's result has explicit constants; however, the bounds are pessemistic.

Remark B.18 (Coherence–sparsity tradeoff). The full version of Tropp's bound establishes a *coherence–sparsity tradeoff* for iid sparse embeddings. The smaller the *coherence* of B (defined to be its largest leverage score, Definition 3.15), the

smaller the sparsity parameter ζ can be. Indeed, for the most incoherent matrices, Tropp's result shows that an expected sparsity of $\zeta = O(\log(n)/m)$ is sufficient to obtain the subspace injection property. In this extreme case, the embedding has just $\mathbb{E}[\text{nnz}(S)] = O(\log n)$ nonzero entries! These results allow the practitioner to employ very aggressive choices for the parameter ζ if they have strong assurances that the matrix is incoherent. On the other hand, for general use, it is advisable to use the "safe" parameter settings $d \sim n$ and $\zeta \sim \log n$. $\diamond$

Limitations. A first limitation of iid sparse embeddings is that, for a worst-case subspace, they provably require an expected sparsity parameter $\zeta \sim \log n$ to achieve the subspace injection condition $\eta_- < 1$.

Proposition B.19 (IID sparse embeddings: Logarithmic sparsity is necessary)**.** *Consider a iid sparse embedding $S \in \mathbb{R}^{m \times d}$ with expected sparsity $\zeta \leq 0.5d$. There exists a matrix Q for which S is not a subspace injection for Q (i.e., the lower distortion has the trivial value $\eta_- = 1$) with 50% probability unless $\zeta \geq 0.5 \log n$.*

Proof. Consider the adversarial matrix $Q \in \mathbb{R}^{m \times n}$ from (B.2). The matrix Q has the effect of selecting the first n columns of S^*, so $S^*Q \in \mathbb{R}^{d \times n}$ is an iid sparse embedding with expected sparsity ζ.

To show that S has lower distortion $\eta_- = 1$ with at least 50% probability, it is sufficient to show that S^*Q has a zero column with at least 50% probability. Each column of S^*Q has the following probability of being fully zero:

$$p := \left(1 - \frac{\zeta}{d}\right)^d \geq \exp(-2\zeta).$$

Here, we used the numerical inequality $1 - \alpha \geq \exp(-2\alpha)$, which is valid for all $\alpha \in [0, 0.5]$. Thus, the probability that some column of S^*Q is zero is $1 - (1 - p)^n$. For this probability to be at most $1/2$, we must have $p \leq 1 - 2^{-1/n}$. Observe that

$$1 - 2^{-1/n} = 1 - \exp\left(-\frac{\log 2}{n}\right) \leq \frac{\log 2}{n}.$$

Thus, by the chain of inequalities $\exp(-2\zeta) \leq p \leq 1 - 2^{-1/n} \leq (\log 2)/n$ and some algebra, we deduce

$$\zeta \geq \frac{1}{2} \log\left(\frac{n}{\log 2}\right) > \frac{1}{2} \log n.$$

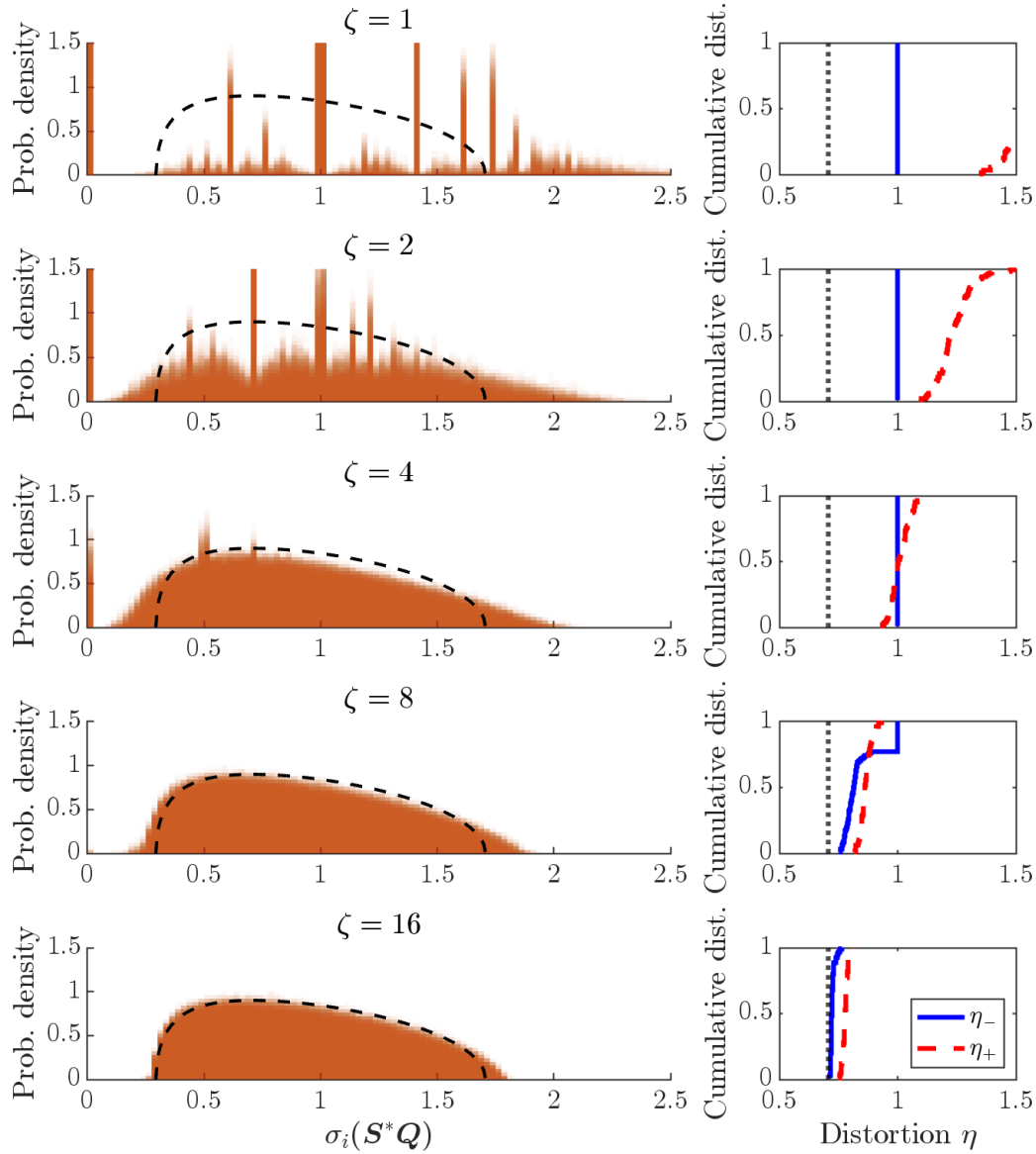The stated result is proven. $\qquad\square$

Figure B.5: *Left:* Superimposed singular value distributions $\sigma(S^*Q)$ for 100 independent realizations of a iid sparse embedding applied to the adversarial matrix (B.2) with embedding dimension $d = 2n$ for expected sparsity levels $\zeta \in \{1, 2, 4, 8, 16\}$ (*top* to *bottom*). The Marčenko–Pastur singular value distribution is shown as the black dashed line. *Right:* Empirical cumulative distribution function of lower and upper distortions. The asymptotic limiting value $\eta = \sqrt{n/d}$ is shown in black.

A second limitation of iid sparse embeddings, less significant in my eyes, is that it appears to require $\zeta \sim 1/\eta_-^2$ to achieve lower distortion η_- [CDD25a, Rem. 3.6]. In contrast, sign embeddings (Appendix B.6) allow $\zeta \sim 1/\eta$ to obtain distortion η.

Empirical results. Figure B.5 shows the results for iid sparse embeddings on the difficult instance (B.2). We set $d := 2n$ and we test sparsity parameters $\zeta = 2^i$ for

$0 \le i \le 4$. We observe gradual convergence towards the Marčenko–Pastur singular value distribution as ζ increases. To achieve a high probability of success, we require expected sparsity $\zeta > 8$ (for this problem size).

B.6 Sparse sign embeddings

The weakness of iid sparse embeddings is in the name—the entries are generated iid. Because of this, it is possible—likely, even—that full rows of S are zero unless ζ is chosen sufficiently large. We can remedy this issue by using embeddings that have a fixed number of nonzeros per row. This motivates the following definition:

Definition B.20 (Sparse sign embedding). A *sparse sign embedding $S \in \mathbb{R}^{m\times d}$* with *sparsity ζ* is a random matrix constructed as follows. Each row is independent and possesses exactly ζ nonzero entries. The nonzero entries are placed in uniformly random positions (selected *without replacement*) and have uniform $\pm\zeta^{-1/2}$ values.

See Fig. 21.1 in the main body for an illustration.

Sparse sign embeddings go by many names: sparse Johnson–Lindenstrauss embeddings (SJLTs, [KN12]), oblivious sparse norm-approximating projections (OSNAPs, [NN13]), sparse sign embeddings [MT20], hashing matrices [CFS21], short-axis sketching operators (SASOs, [MDME+23]), and sparse maps [NT24]. (We note, however, that some of these terms refer to a broader class of embeddings, with sparse sign embeddings—as defined above—serving as just one instance.) As best I can tell, this construction is due to Kane and Nelson [KN12]. The case $\zeta = 1$ of the sparse sign embedding is known as a CountSketch, which was introduced in the paper [CCF04].

Let us begin this time with empirical results. Figure B.6 shows the results for sparse sign embeddings on the difficult example (B.2). We set $d := 2n$ and we test sparsity parameters $\zeta = 2^i$ for $0 \le i \le 4$. We see that, for $\zeta \ge 4$, the sparse sign embedding is completely reliable, showing only slight excursions from the Marčenko–Pastur singular value distribution and producing a lower distortion $\eta_- \le 0.8$. Compare with the $\zeta = 4$ and $\zeta = 8$ panels of Fig. B.5, which show nonzero probbabilities of complete failure, $\eta_- = 1$. This suggests that sparse sign embeddings "work" for smaller values of the sparsity parameter ζ than for iid sparse embeddings.

For smaller values of ζ, the performance of sparse sign embeddings deteriorates. The CountSketch matrix (i.e., $\zeta = 1$) fails catastrophically (i.e., $\eta_- = 1$) in every trial in this example. The $\zeta = 2$ embedding also fails catastrophically about 50%
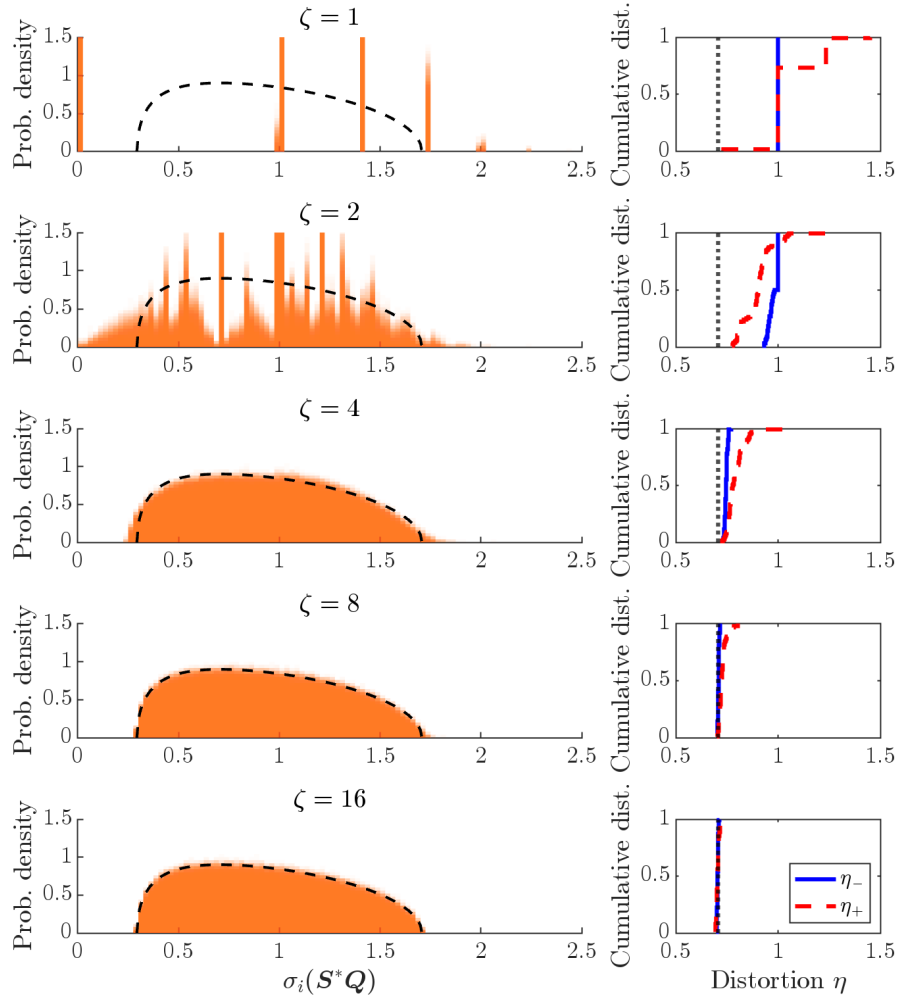
Figure B.6: *Left:* Superimposed singular value distributions $\boldsymbol{\sigma}(\boldsymbol{S}^*\boldsymbol{Q})$ for 100 independent realizations of a sparse sign embedding applied to the adversarial matrix (B.2) with embedding dimension $d = 2n$ for expected sparsity levels $\zeta \in \{1, 2, 4, 8, 16\}$ (*top* to *bottom*). The Marčenko–Pastur singular value distribution is shown as a black dashed line for reference. *Right:* Empirical cumulative distribution function of lower and upper distortions. The asymptotic limiting value $\eta = \sqrt{n/d}$ is shown in black.

of the time. Based on these examples, $\zeta = 3$ is the smallest value I would accept for general-purpose use (with embedding dimension $d \sim n$). The CountSketch embedding ($\zeta = 1$) achieves the subspace embedding property only with a *quadratic* embedding dimension $d \sim n^2$ [Woo14, Thm. 23]!

The experiments in thesis add to a growing body of evidence [TYUC19; DM23b; Epp23c; MBMD+25; EMN25; CNRS+25; CEMT25] that sparse sign embeddings are fast and completely reliable for linear algebraic computations. The insights of this literature may be crystallized into a hypothesis:

> Of the existing constructions, sparse sign embeddings are the most effective oblivious subspace embedding for general-purpose use in matrix computations. They are reliable (i.e., achieve η_- bounded away from 1) when implemented with an embedding dimension as low as $d = 2n$ and *constant* sparsity parameter $\zeta = 4$ or $\zeta = 8$.

In the rest of this section, I will support this claim with evidence.

Implementation and runtime comparison

Making sparse sign embeddings fast in practice requires careful implementation and a good sparse linear algebra library. Often, to obtain the greatest speed in a high-level programming language, such as MATLAB or Python, the construction of the sparse sign embedding should be written in a low-level programming language, like C or C++.

The challenge of implementing sparse sign embeddings is selecting the locations of the nonzero entries in each row *without replacement*. One simple approach, appropriate when $1 \approx \zeta \ll d$ is to use *rejection sampling*. Generate a stream indices $i \sim \text{UNIF}\{1, \ldots, d\}$, and check whether each index has already been picked, stopping when ζ indices have been selected. This approach is illustrated in the following code segment.

```
positions = zeros(zeta,1); i = 0;
while i <= zeta                               % Pick zeta positions
    idx = randsample(d,1);                    % Random index in [1,d]
    for j = 1:(i-1)                           % Check for duplicates
        if positions(j) == idx; continue; end % Reject duplicate
    positions(i) = idx; i = i + 1;            % Accept new position
end
```
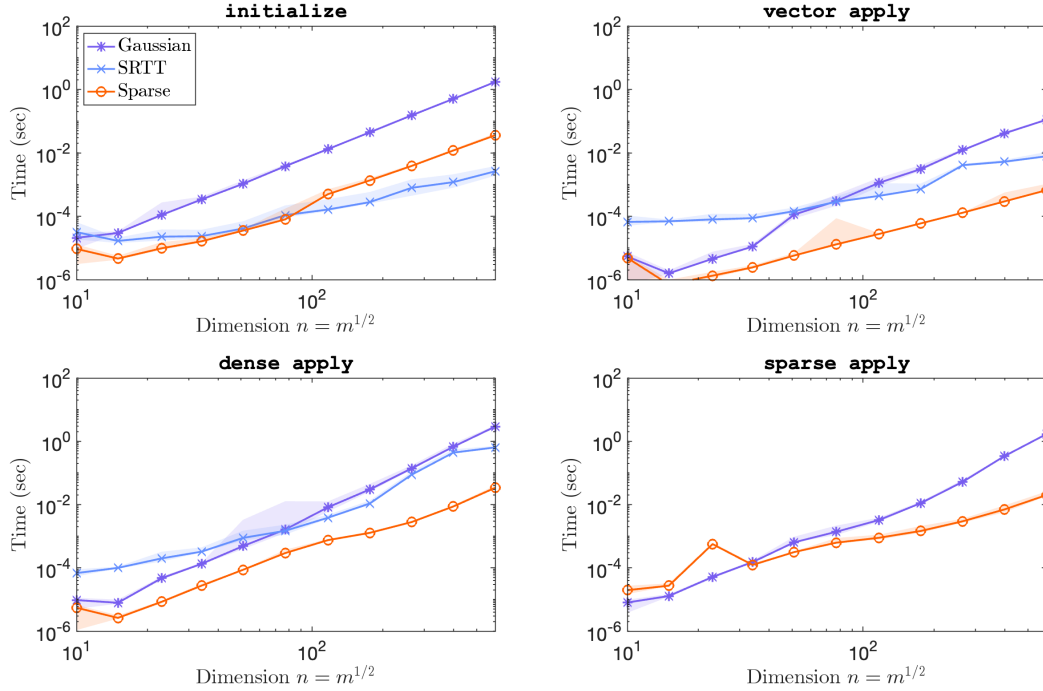
Figure B.7: Runtimes for initialization (*top left*), matrix–vector apply (*top right*), dense matrix–matrix apply (*bottom left*), and sparse matrix–matrix apply (*bottom right*) for Gaussian embeddings (purple asterisks), standard SRTTs (blue crosses), and sparse sign embeddings (orange circles). Lines show median of 100 trials, and shaded regions show 10% and 90% quantiles.

Assuming $\zeta \leq d/2$, the expected cost of generating the nonzero locationsis $O(\zeta^2)$ operations per row. For small values of ζ and with a low-level implementation in C, the overhead of this rejection sampling approach is minimal, and it produces sparse sign embeddings rapidly in practice. C code for generating sparse sign embeddings with this construction appears in Appendix E.

More efficient implementations of sparse sign embeddings have been developed as part of the RandBLAS/RandLAPACK software project. In particular, using a careful implementation of Fisher–Yates sampling, the RandBLAS software builds sparse sign embeddings $O(\zeta m + d)$ operations, which is nearly optimal. See [MDME+23, App. A] for details.

Figure B.7 compares the runtimes of Gaussian embeddings, (standard) subsampled trigonometric transforms, and sparse sign embeddings (with sparsity $\zeta = 4$). We consider a family of problems with dimensions $n = m^{1/2} \in [10, 600]$, and we set $d := 2n$ for each value of n. We evaluate four components of the runtime:

time to construct the embedding, i.e., generate the necessary random variables (*top left*), time to compute a matrix–vector product S^*c (*top right*), time to compute the product S^*B with a dense matrix $B \in \mathbb{R}^{m \times n}$ (*bottom left*), and time to multiply with a sparse matrix $B \in \mathbb{R}^{m \times n}$ with 10 nonzeros per column (*bottom right*). (For the last of these, we omit the SRTT because it does not possess a fast multiply operation with sparse matrices.) Sparse sign embeddings are definitively the fastest on all metrics except initialization. In particular, for $n = 600$, the dense matrix–matrix product operation for sparse sign embeddings is 19× faster than SRTTs and 86× faster than Gaussian embeddings.

Analysis

The best analysis of sparse sign embeddings remains the work of Cohen [Coh16] (Fact 21.10). I restate his theorem below.

Fact B.21 (Sparse sign embeddings: Subspace embedding property)**.** *Sparse sign embeddings are* oblivious *subspace embeddings with dimension n, distortion $\eta \in (0, 1)$, and failure probability $\delta \in (0, 1)$ provided*

$$d \geq \Omega\left(\frac{n \log(n/\delta)}{\eta^2}\right) \quad and \quad \zeta \geq \Omega\left(\frac{\log(n/\delta)}{\eta}\right).$$

In particular, Cohen's results show that *logarithmic sparsity* $\zeta = O(\log n)$ and a *log-linear embedding dimension* $d = O(n \log n)$ suffice to obtain the subspace embedding guarantee.

Cohen's analysis helps explain the success of sparse embeddings. However, it falls short of explaining the near-Gaussian behavior of sparse sign embeddings when implemented with a *linear embedding dimension* $d = O(n)$ and constant sparsity $\zeta = 4$. Closing this gap remained an open problem in the years after Cohen's analysis.

While no further progress has been made on analyzing sparse sign embeddings, recent work has improved our understanding of related constructions. We saw above the work of Tropp (Fact B.17, [Tro25]), which considered iid sparse embeddings. Another construction is given as follows:

Definition B.22 (SparseStack)**.** Let $d, m, \zeta \geq 1$ be integer parameters such that ζ divides evenly into d. An *SparseStack* is a sparse matrix with exactly one nonzero entry per *subcolumn* $S^*((i - 1)\zeta + 1 : i\zeta, j)$ of S^* for $1 \leq i \leq d/\zeta$ and $1 \leq j \leq m$.

This entry is placed in a uniformly random position and has a uniformly random value $\pm 1/\sqrt{\zeta}$.

SparseStacks were introduced alongside sparse sign embeddings in the original paper of Kane and Nelson [KN12]. The SparseStack model is studied in [CDDR24; CDD25a]. The paper [CDD25a] contains the following result.

Fact B.23 (SparseStack). *SparseStacks are* oblivious *subspace embeddings with dimension n, distortion $\eta \in (0,1)$, and failure probability $\delta \in (0,1)$ provided*

$$\Omega\left(\frac{n + \log(1/\delta)}{\eta^2}\right) \le d \le O(e^n) \quad and \quad \zeta \ge \Omega\left(\frac{\log^2(n/(\eta\delta))}{\eta} + \log^3(n/(\eta\delta))\right).$$

This bound shows that an embedding dimension of $d \sim n$ suffices for SparseStacks to achieve the subspace embedding property provided that the sparsity level is $\zeta \sim \log^3 n$. Compare with Tropp's bound (Fact B.17), which states that $d \sim n$ and $\zeta \sim \log n$ imply the *subspace injection* property for iid sparse maps.

To summarize, no theoretical results yet explain the success of sparse sign embeddings with the aggressive parameters $d = 2n$ and a $\zeta = 4$. The bound Fact B.21 of Cohen [Coh16], showing $d \sim n \log n$ and $\zeta \sim \log n$, almost reaches the parameter settings we use in practice, and recent works [CDDR24; CDD25a; Tro25] (analyzing slightly types of sparse embeddings) show that a linear embedding dimension $d \sim n$ suffices, provided one has a large enough sparsity parameter (e.g., $\zeta \sim \log n$ in [Tro25]).

Impossibility results: What they say and what they don't say

Particularly sophisticated readers may be aware that there are impossibility results showing that sparse embeddings with $\zeta = 4$, as we recommend. Cohen [Coh16, §1], summarizing [NN14], writes:

> Lower bounds ([NN14]), on the other hand, *suggest* that the true tradeoff allows $d = O(B(n + \log(1/\delta))/\eta^2)$ with $\zeta = O(\log_B(n/\delta)/\eta)$.

I have rephrased Cohen's notation to be consistent with ours, and the emphasis is added. Cohen suggests that [NN14] establishes a *sparsity–embedding dimension tradeoff*, controlled by a continuous parameter $B > 0$. In particular, to obtain a linear embedding dimension $d \sim n$, Cohen suggests that a logarithmic embedding dimension $\zeta \sim \log n$ is *necessary* to achieve the subspace embedding property. If true, this

provide given strong theoretical evidence that constant-sparsity embeddings could fail for some instances.

A careful reading of the paper [NN14] yields a more nuanced conclusion. Specifically, upon close inspections, the arguments of [NN14, §3] yield the following result.

Fact B.24 (Sparse embeddings: Lower bound)**.** *Assume $m \geq 100n^2$. Any sparse oblivious subspace embedding $S \in \mathbb{R}^{m \times 2n}$ with embedding dimension $d = 2n$, distortions η_+, η_-, and failure probability $\delta = 0.02$ must have*

$$\zeta \geq \text{const} \cdot \left(\frac{1 - \eta_-}{1 + \eta_+} \right)^2 \frac{\log n}{\log \log n}$$

We make a few observations. First, to achieve a subspace embedding with $\eta < 1$ and embedding dimension $d = 2n$, this bound shows we need a sparsity level of $\zeta \sim (\log n)/\log \log n$. This is slightly weaker than the suggested $\zeta = \Omega(\log n)$ lower bound in Cohen's quote above. Second, and more importantly, Fact B.24 leaves open the possibility of constant sparsity embeddings $\zeta = \text{const}$ satisfying the subspace embedding property with

$$\eta_- = \text{const} < 1 \quad \text{and} \quad \eta_+ = O\left(\sqrt{\frac{\log n}{\log \log n}} \right). \tag{B.3}$$

Under the sketching asymmetry principle, an embedding with these parameters will produce meaningful results in most applications. Recall: For a sketching matrices to be useful, the lower distortion η_- must be away from 1, but we can tolerate the upper distortion η_+ to be logarithmically large

Let us clarify that Fact B.24 does not *prove* that sparse sign embeddings with parameters $d = 2n$ and $\zeta = \text{const}$ achieve the distortion scaling in (B.3); it just states that these distortions are consistent with the best-known information-theoretic lower bounds from [NN14, §3]. I conjecture that sparse sign embeddings attain these bounds.

Conjecture B.25 (Sparse sign embeddings: Linear embedding dimension and constant sparsity)**.** *Sparse sign embeddings with parameters $d = 2n$ and $\zeta = 4$ are oblivious subspace embeddings with dimension n, failure probability $\delta = 0.01$, and distortions given by* (B.3)*.*

As some evidence for this c,kaim, we can use the techniques of [NN14] to show that the upper distortion satisfies $\eta_+ \geq \Omega((\log n)/\log \log n)$ with high probability.

Proposition B.26 (Sparse sign embeddings: Upper distortion)**.** *Consider a sparse sign embedding $S \in \mathbb{R}^{m \times d}$ with parameters $d = 2n$ and $\zeta = $ const. Then, there exists a matrix Q for which the upper distortion of S is*

$$\eta_+ \geq \text{const} \cdot \sqrt{\frac{\log n}{\log \log n}}$$

with 99% probability.

Proof sketch. The idea is similar to the analysis of [NN14]. Set Q to be the "the adversarial matrix" (B.2). Then S^*Q is a sparse random matrix with ζ entries in each column in uniformly random positions, with random values $\pm\zeta^{-1/2}$. Think of each row of S^*Q as a bin and each nonzero entry as a ball dropped into a uniformly random bin (with the constraint that, within each column, no two balls are dropped in the same bin). Up to the dependencies within each column (which become insignificant in the limit $n \to \infty$, since $\zeta = $ const), this is an instance of the classical balls-and-bins problem, for which it is known the heaviest bin has at least const $\cdot (\log n)/\log \log n$) balls with high probability (see, e.g., [NN14, Lem. 14]). Thus, with high probability,

$$\eta_+ = \sigma_{\max}(S^*Q) \geq \max_{1 \leq i \leq d} \|(S^*Q)(i,:)\|$$

$$\geq \sqrt{\text{const} \cdot \frac{\log n}{\log \log n} \cdot \left(\pm\zeta^{-1/2}\right)^2} = \text{const} \cdot \sqrt{\frac{\log n}{\log \log n}}.$$

The argument is complete. $\square$

B.7   Conclusions

In my experience working, the question "Which sketch should you use?" is shrouded in folklore and accounts of personal experience. Despite the importance of sketching to modern randomized matrix computations, there seems to be little consensus on which types of sketching matrices to use in practice. In this chapter, I have evaluated different sketching matrices on a challenging test matrix and summarized the relevant theoretical literature. On the basis of this evidence, I and others believe sparse sign embeddings are a strong candidate for the sketching matrix of choice for most applications. Still, there remain open questions, and it is worth further investigations into the theoretical properties and empirical behavior of sparse sign embeddings and other sketching matrices.

B.8 Postscript: Recent developments

Subsequent to my defense and initial drafting of this thesis, Chris Camaño, Raphael Meyer, Joel Tropp, and myself released a paper [CEMT25] revisiting the theory of sketching. In particular, it shows that subspace injection matrices satisfying the isotropy condition

$$\mathbb{E}[\|S^*v\|^2] = \|v\|^2 \quad \text{for all } v \in \mathbb{K}^n$$

produce sketch-and-solve solutions $\widehat{x} \in \mathbb{K}^n$ that are accurate up to a constant factor

$$\|c - B\widehat{x}\| \le \text{const} \cdot \|c - Bx\|.$$

It also analyzes the use of such random matrices in other randomized linear algebra primitives, including the randomized SVD (Section 2.3), Nyström approximation (Section 2.5), and generalized Nyström approximation (Section 10.2). This paper also provides more evidence for the reliability of sparse sketching matrices, and it discusses how SparseStack embeddings (Definition B.22) may be easier to construct than sparse sign embeddings (Definition B.20). Perhaps it is the SparseStack, not the sparse sign embedding, that should be the sketching matrix of choice for most applications!

As another late-breaking update, Chenakkod, Dereziński, and Dong released a new preprint [CDD25b] containing an even sharper analysis of SparseStack embeddings. They show that SparseStacks are oblivious subspace embeddings with parameters

$$d = O\left(\frac{n}{\eta^2}\right) \quad \text{and} \quad \zeta = O\left(\frac{\log^{1+O(1/\log\log\log\log n)}(n)}{\eta}\right).$$

This result improves on Fact B.23, and it achieves the lower bound of [NN14] *for oblivious subspace embeddings* up to *subpolylogarithmic* factors.

A p p e n d i x C

ANALYSIS OF SKETCH-AND-SOLVE

The classical result for sketch-and-solve, cited in many popular surveys and expository works [MT20; MDME+23; KT24] and many of my own works, is as follows:

Proposition C.1 (Sketch-and-solve: Simple bound). *Let S be a subspace embedding for $[B \quad c]$ with lower and upper distortions η_-, η_+ (Definition B.1), and let $\widehat{x}$ be the corresponding sketch-and-solve solution. Then*

$$\|c - B\widehat{x}\| \leq \frac{1 + \eta_+}{1 - \eta_-} \cdot \|c - Bx\|.$$

In particular, using a sparse sign embedding with Cohen's parameter settings (Fact B.21), sketch-and-solve produces $(1 + \varepsilon)$-approximate least-squares solution with 99% probability provided

$$d = O\left(\frac{n \log n}{\varepsilon^2}\right).$$

The runtime is

$$O\left(mn \log n + \frac{n^3 \log n}{\varepsilon^2}\right) \text{ operations.}$$

Proof. We make the following chain of inequalities

$$\|c - B\widehat{x}\| \leq \frac{1}{1 - \eta_-} \cdot \|S^*(c - B\widehat{x})\| \leq \frac{1}{1 - \eta_-} \cdot \|S^*(c - Bx)\| \leq \frac{1 + \eta_+}{1 - \eta_-} \cdot \|c - Bx\|.$$

The first and third inequality are subspace embedding property, and the second inequality is the optimality of $\widehat{x}$ as a solution to the sketched least-squares problem $\mathrm{argmin}_{z \in \mathbb{K}^n} \|S^*(c - B\widehat{x})\|$. □

This bound suggests that the residual norm of the sketch-and-solve solution is within a factor $1 + O(\eta)$ of the optimal residual norm and that an embedding dimension $d \sim 1/\varepsilon^2$ is necessary to obtain a $(1 + \varepsilon)$-approximate least-squares solution. In fact, this scaling can be improved using a sharper analysis:

Theorem C.2 (Sketch-and-solve: Sharper bound). *Let S be a subspace embedding for $[B \quad c]$ with lower and upper distortion η_-, η_+ (Definition B.1) and let $\widehat{x} =$*

$(S^*B)^\dagger(S^*c)$ *be the sketch-and-solve solution. Assume* $\eta \coloneqq \max\{\eta_+, \eta_-\} \leq 1$.
Then

$$\|c - B\widehat{x}\| \leq \left(1 + \frac{4.5\eta^2}{(1 - \eta_-)^4}\right) \|c - Bx\|.$$

In particular, using a sparse sign embedding with Cohen's parameter settings (Fact B.21), sketch-and-solve produces $(1 + \varepsilon)$*-approximate least-squares solution with 99% probability provided*

$$d = O\left(\frac{n \log n}{\varepsilon}\right).$$

The runtime is

$$O\left(mn \log n + \frac{n^3 \log n}{\varepsilon}\right) \text{ operations.}$$

This result establishes represents a significant improvement over Proposition C.1 in the limit where $\eta \to 0$ (equivalently $d \to \infty$), and the resulting $1 + O(\eta^2)$ scaling is sharp, as demonstrated by exact computations for Gaussian embeddings [BP20; Epp24d]. The sharp $d \sim 1/\varepsilon$ scaling for sketch-and-solve has been known since the work of Drineas, Mahoney, Muthukrishnan, and Sarlós [DMMS11], originally released in 2007; see also the expository notes [LK22; Mey23]. However, these earlier bounds pass through a different technical argument and require hypotheses on the sketching matrix S beyond the subspace embedding property. This bound, a version of which was published in the blog post [Epp25], is the only result I know which established the correct $1 + O(\eta^2)$ scaling with the subspace embedding property as the sole hypothesis.

We proceed now with its proof.

Proof of Theorem C.2. We shall bound $\|B(\widehat{x} - x)\|$. To do so, we use the representation

$$B(\widehat{x} - x) = B(S^*B)^\dagger S^*(c - Bx).$$

Letting $B = QR$ be an (economy-size) QR decomposition, we have

$$B(\widehat{x} - x) = Q(S^*Q)^\dagger S^*(c - Bx) = Q(Q^*SS^*Q)^{-1}Q^*SS^*(c - Bx).$$

Thus, introducing $\overline{r} \coloneqq (c - Bx)/\|c - Bx\|$, we have

$$\|B(\widehat{x} - x)\| \leq \left\|(Q^*SS^*Q)^{-1}\right\| \cdot \|Q^*SS^*\overline{r}\| \cdot \|c - Bx\|. \tag{C.1}$$

We now bound the right-hand side of (C.1). By Proposition B.4, we have

$$\left\|(\boldsymbol{Q}^*\boldsymbol{S}\boldsymbol{S}^*\boldsymbol{Q})^{-1}\right\| = \frac{1}{\sigma_{\min}(\boldsymbol{S}^*\boldsymbol{Q})^2} \leq \frac{1}{(1-\eta_-)^2}. \tag{C.2}$$

Bounding $\|\boldsymbol{Q}^*\boldsymbol{S}\boldsymbol{S}^*\overline{\boldsymbol{r}}\|$ requires a more sophisticated argument. Observe that

$$\|\boldsymbol{Q}^*\boldsymbol{S}\boldsymbol{S}^*\overline{\boldsymbol{r}}\| \text{ is a submatrix of } [\boldsymbol{Q} \quad \overline{\boldsymbol{r}}]^*\boldsymbol{S}\boldsymbol{S}^*[\boldsymbol{Q} \quad \overline{\boldsymbol{r}}] - \mathbf{I}.$$

The columns of the matrix $[\boldsymbol{Q} \quad \overline{\boldsymbol{r}}]$ form an orthonormal basis for $\mathrm{range}([\boldsymbol{B} \quad \boldsymbol{c}])$. Therefore, by Proposition B.4 again, we have

$$\|\boldsymbol{Q}^*\boldsymbol{S}\boldsymbol{S}^*\overline{\boldsymbol{r}}\| \leq \left\|[\boldsymbol{Q} \quad \overline{\boldsymbol{r}}]^*\boldsymbol{S}\boldsymbol{S}^*[\boldsymbol{Q} \quad \overline{\boldsymbol{r}}] - \mathbf{I}\right\| \leq \max\{(1+\eta_+)^2-1, 1-(1-\eta_-)^2\}. \tag{C.3}$$

Combining (C.2) and (C.3) and substituting in (C.1) yields

$$\|\boldsymbol{B}(\widehat{\boldsymbol{x}} - \boldsymbol{x})\| \leq \frac{2\eta + \eta^2}{(1-\eta_-)^2}\|\boldsymbol{c} - \boldsymbol{B}\boldsymbol{x}\|.$$

Combining with the Pythagorean identity,

$$\|\boldsymbol{c} - \boldsymbol{B}\widehat{\boldsymbol{x}}\|^2 = \|\boldsymbol{c} - \boldsymbol{B}\boldsymbol{x}\|^2 + \|\boldsymbol{B}(\widehat{\boldsymbol{x}} - \boldsymbol{x})\|^2 \leq \left(1 + \frac{(2\eta + \eta^2)^2}{(1-\eta_-)^4}\right)\|\boldsymbol{c} - \boldsymbol{B}\boldsymbol{x}\|^2.$$

Finally, with the hypothesis, $\eta \leq 1$, we have $(2\eta + \eta^2)^2 \leq 9\eta^2$. Take square roots and use the identity $(1+x)^{1/2} \leq 1 + 0.5x$ for $x \geq -1$ to complete the argument. $\square$

A p p e n d i x   D

DEFERRED PROOFS

This section contains deferred proofs of results from the main text.

D.1   Proof of Theorem 3.12

We use a more algebraic version of the proof of [DRVW06, Prop. 3.3]. Fix a sufficiently small parameter $\delta > 0$ and generate the matrix

$$A = (k+1)\mathbf{I}_{k+1} - (1-\delta)\mathbf{1}_{k+1}\mathbf{1}_{k+1}^* \in \mathbb{R}^{(k+1)\times(k+1)}.$$

This matrix has eigenvalue $(k+1)$ with multiplicity k and eigenvalue $(k+1)\delta$ with multiplicity one. Therefore,

$$\operatorname{tr}(A - [\![A]\!]_k) = (k+1)\delta.$$

Since the matrix A is the same under any conjugation by a permutation matrix, we can assume without loss of generality that $\mathsf{S} = \{1, \ldots, k\}$. We compute

$$\operatorname{tr}(A - A\langle \mathsf{S}\rangle) = a_{(k+1)(k+1)} - a_{k+1}(1:k)^* A(1:k, 1:k)^{-1} a_{k+1}(1:k).$$

Applying the Sherman–Morrison formula gives

$$A(1:k, 1:k)^{-1} = \frac{1}{k+1}\cdot\mathbf{I}_k + \frac{1-\delta}{(k+1)(1+k\delta)}\cdot\mathbf{1}_k\mathbf{1}_k^*.$$

Thus,

$$\operatorname{tr}(A - A\langle \mathsf{S}\rangle) = (k+\delta) - \frac{(1-\delta)^2 k}{k+1} - \frac{(1-\delta)^3 k^2}{(k+1)(1+k\delta)} = (k+1)^2\delta + O(\delta^2).$$

Taking δ sufficiently small proves the claim. $\square$

D.2   Proof of Theorem 10.8

The existence result for weighted CUR decompositions follows from Fact 10.11. We focus on establishing the nonexistence result for unweighted CUR approximations. Our argument is based on a construction of Dereziński, Warmuth, and Hsu [DH18, Thm. 1]. First consider the case $k = 1$.

Consider the matrix

$$\boldsymbol{B} = \begin{bmatrix} 1 & 1 \\ p^{-1/2} & 0 \\ p^{-1/2} & 0 \\ p^{-1/2} & 0 \\ \vdots & \vdots \\ p^{-1/2} & 0 \end{bmatrix} \in \mathbb{R}^{(p+1)\times 2},$$

and choose column subset $\mathsf{S} := \{1\}$. The column projection approximation is

$$\widehat{\boldsymbol{B}}_{\mathrm{CP}} = \boldsymbol{B}(:,\mathsf{S})\boldsymbol{B}(:,\mathsf{S})^\dagger \boldsymbol{B} = \begin{bmatrix} \boldsymbol{b}_1 & \tfrac{1}{2}\boldsymbol{b}_1 \end{bmatrix},$$

with error

$$\|\boldsymbol{B} - \widehat{\boldsymbol{B}}_{\mathrm{CP}}\|_{\mathrm{F}}^2 = \frac{1}{2}.$$

Let $\mathsf{T} \subseteq \{1,\dots,p+1\}$ denote any row subset of ℓ elements of $\boldsymbol{B}$. If $1 \notin \mathsf{T}$, then the unweighted CUR cross approximation is zero and thus $\|\boldsymbol{B} - \widehat{\boldsymbol{B}}\|_{\mathrm{F}}^2 = 3$. Thus, we may assume $1 \in \mathsf{T}$. Then the unweighted CUR cross approximation is

$$\widehat{\boldsymbol{B}} = \boldsymbol{B}(:,\mathsf{S})\boldsymbol{B}(\mathsf{T},\mathsf{S})^\dagger \boldsymbol{B}(\mathsf{T},:) = \begin{bmatrix} \boldsymbol{b}_1 & \frac{\boldsymbol{b}_1(\mathsf{T})^*\boldsymbol{b}_2(\mathsf{T})}{\boldsymbol{b}_1(\mathsf{T})^*\boldsymbol{b}_1(\mathsf{T})}\boldsymbol{b}_1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{b}_1 & \frac{\boldsymbol{b}_1}{1+(\ell-1)/p} \end{bmatrix}$$

with error

$$\|\boldsymbol{B} - \widehat{\boldsymbol{B}}\|_{\mathrm{F}}^2 = 1 - \frac{2}{1 + (\ell-1)/p} + \frac{2}{(1 + (\ell-1)/p)^2} =: f(\ell). \tag{D.1}$$

The function f is strictly decreasing on the interval $[1, p+1]$ and it achieves the value $3/4$ at $\ell = 1 + (3 - 2\sqrt{2})p$. The result is proven for $k = 1$.

To prove the result for $k > 1$, generate a block diagonal matrix with k copies of $\boldsymbol{B}/\sqrt{k}$, and set $\mathsf{S} = \{1, p+1, \dots, (k-1)p+1\}$. The optimal squared Frobenius column projection error is again $1/2$, and the CUR error is bounded by

$$\|\boldsymbol{B} - \widehat{\boldsymbol{B}}\|_{\mathrm{F}}^2 \geq \frac{1}{k}\left(f(\theta_1\ell) + \dots + f(\theta_2\ell) + \dots f(\theta_k\ell)\right), \tag{D.2}$$

where $\theta_j\ell$ denotes the number of indices assigned to block j. Here, to handle boundary cases, the values $f(t)$ are defined as (D.1) for $0 < t \leq p$, $f(0) = 3$, and $f(t) = 1/2$ for $t > p$. With this definition, f is convex on $\mathbb{R}_+$. Therefore, by Jensen's inequality, the right-hand side of (D.2) is minimized by the equal apportionment $\theta_1 = \dots = \theta_k = 1/k$:

$$\|\boldsymbol{B} - \widehat{\boldsymbol{B}}\|_{\mathrm{F}}^2 \geq \frac{1}{k}\left(f(\theta_1\ell) + \dots + f(\theta_2\ell) + \dots f(\theta_k\ell)\right) \geq f(\ell/k).$$

Ergo, the optimal unweighted CUR decomposition has a squared Frobenius norm error at least $1.5\times$ the squared Frobenius column projection error provided $\ell \leq k + (3 - 2\sqrt{2})pk$. $\qquad\square$

A p p e n d i x E

IMPLEMENTATION OF SPARSE RANDOM EMBEDDINGS

Program E.1: MEX (MATLAB-callable C function) implementation of sparse sign embeddings.

```c
#include <math.h>
#include "mex.h"

int log_base_d(long a, int d) {
    int output = 0;
    while (a > 0) {
        a /= d;
        output += 1;
    }
    return output-1;
}


void mexFunction(int nlhs, mxArray *plhs[],
                 int nrhs, const mxArray *prhs[]) {

    mwSize d = (mwSize) mxGetScalar(prhs[0]);
    mwSize m = (mwSize) mxGetScalar(prhs[1]);
    mwSize zeta = (mwSize) mxGetScalar(prhs[2]);
    mwSize nnz = m*zeta;

    int idx_per_rand = log_base_d((long) RAND_MAX + 1, d);
    int bit_per_rand = log_base_d((long) RAND_MAX + 1, 2);

    if (zeta > d) zeta = d;

    double lowval = -1/sqrt((double) zeta);
    double increment = -2*lowval;

    plhs[0] = mxCreateSparse(d,m,nnz,false);
    double *vals  = mxGetPr(plhs[0]);
    mwIndex *rows = mxGetIr(plhs[0]);
```

```
mwIndex *colstarts = mxGetJc(plhs[0]);

// Set values
int myrand = rand();
for (int i = 0; i+bit_per_rand < nnz; i += bit_per_rand) {
    for (int j = i; j < i+bit_per_rand; ++j) {
        vals[j] = (myrand % 2) * increment + lowval;
        myrand = myrand >> 1;
    }
    myrand = rand();
}
for (int i = bit_per_rand*(nnz/bit_per_rand); i < nnz; ++i) {
    vals[i] = (myrand % 2) * increment + lowval;
    myrand = myrand >> 1;
}

// Set column starts
for (int i = 1; i < m+1; ++i){
    colstarts[i] = i*zeta;
}

// Set row indices
myrand = rand();
int ir = 0;
for (int i = 0; i < m*zeta; i += zeta) {
    int idx = 0;
    while (idx < zeta) {
        rows[i+idx] = myrand % d;
        ir++;
        if (ir == idx_per_rand) {
            ir = 0;
            myrand = rand();
        } else {
            myrand /= d;
        }
        int j = 0;
        for (; j < idx; ++j) {
            if (rows[i+idx] == rows[i+j]) break;
```

```
            }
            idx += (int) (j == idx);
        }
    }

}
```

Appendix F

HELPFUL MATLAB SUBROUTINES

Program F.1 `cnormc.m`. Code to normalize the columns of a matrix.

```matlab
function M = cnormc(M)
% Input:  Matrix M
% Output: Matrix M rescaled to have unit norm columns

M = M ./ vecnorm(M,2,1);

end
```

Program F.2 `random_signs.m`. Generate matrix of random signs.

```matlab
function Om = random_signs(n,s)
% Input:  Dimensions n, s for random sign matrix
% Output: An n*s random sign matrix Om
    Om = -3 + 2*randi(2,n,s);
end
```

Program F.3 `sqrownorms.m`. Compute squared row norms of a matrix.

```matlab
function srn = sqrownorms(F)
% Input:  Matrix F
% Output: Vector srn containing the squared row norms of F

srn = vecnorm(F,2,2) .^ 2;

end
```

Program F.4 `sqcolnorms.m`. Compute squared column norms of a matrix.

```matlab
function scn = sqcolnorms(F)
% Input:  Matrix F
% Output: Vector scn containing the squared column norms of F

scn = vecnorm(F,2,1)' .^ 2;

end
```

Program F.5 `mypcg.m`. Preconditioned conjugate gradient for solving positive definite linear systems.

```
function X = mypcg(A,pre,b,iter)
% Input:  Function A() computing matrix products A(z) = A*z, function
%         pre applying the preconditioner, right-hand side b, and
%         number of iterations iter
% Output: Approximate solutions stacked columnwise as X

X = zeros(size(b,1),iter+1);        % Initialize solutions
z = pre(b); p = z;                   % Initialize search direction
for i = 1:iter
    v = A(p);                        % Apply A
    zb = z'*b; eta = zb / (v'*p);
    X(:,i+1) = X(:,i) + eta*p;       % Update solution
    b = b - eta*v;                   % Update residual
    z = pre(b);                      % Apply preconditioner
    gamma = z'*b/zb;
    p = z + gamma*p;                 % Update search direction
end

end
```

Program F.6 `rand_with_evals.m`. Generate a Hermitian matrix with Haar-random eigenvectors and the specified eigenvalues.

```
function A = rand_with_evals(evals)
% Input:  Eigenvalues evals
% Output: Hermitian matrix A with specified eigenvalues

n = length(evals);
[Q,~] = qr(randn(n));
A = Q * diag(evals) * Q';
A = (A+A')/2;

end
```

Program F.7 `haarorth.m`. Generate a Haar-random matrix with orthonormal columns.

```matlab
function Q = haarorth(m,n)
% Input:  Dimensions m and n
% Output: Haar-random matrix Q with orthonormal columns

    [Q,R] = qr(randn(m,n),"econ"); % Orthonomalize a Gaussian matrix
    Q = Q*diag(sign(diag(R)));     % Rescale as appropriately

end
```

BIBLIOGRAPHY

[ABBB+99]   Edward Anderson, Zhaojun Bai, Christian Bischof, L. Susan Blackford, James Demmel, Jack Dongarra, Jeremy Du Croz, Anne Greenbaum, Sven Hammarling, Alan McKenney, and Danny Sorensen. *LAPACK Users' Guide*. 3rd ed. Society for Industrial and Applied Mathematics, 1999. DOI: 10.1137/1.9780898719604 (Cited on pp. 41, 149, 314, 357).

[ABKS75]   Ralph Alben, Marshall Blume, Henry Krakauer, and Lawrence Schwartz. "Exact Results for a Three-Dimensional Alloy with Site Diagonal Disorder: Comparison with the Coherent Potential Approximation". In: *Physical Review B* 12.10 (Nov. 1975), pp. 4090–4094. DOI: 10.1103/PhysRevB.12.4090 (Cited on p. 194).

[AC06]   Nir Ailon and Bernard Chazelle. "Approximate Nearest Neighbors and the Fast Johnson-Lindenstrauss Transform". In: *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing*. May 2006, pp. 557–563. DOI: 10.1145/1132516.1132597 (Cited on p. 374).

[AD18]   Mohammad Al Hasan and Vachik S. Dave. "Triangle Counting in Large Networks: A Review". In: *WIREs Data Mining and Knowledge Discovery* 8.2 (2018), e1226. DOI: 10.1002/widm.1226 (Cited on pp. 191, 192).

[ADMM+15]   David Anderson, Simon Du, Michael Mahoney, Christopher Melgaard, Kunming Wu, and Ming Gu. "Spectral Gap Error Bounds for Improving CUR Matrix Decomposition and the Nyström Method". In: *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*. Feb. 2015, pp. 19–27. URL: https://proceedings.mlr.press/v38/anderson15.html (Cited on p. 169).

[AKGH+14]   Charu C. Aggarwal, Xiangnan Kong, Quanquan Gu, Jiawei Han, and Philip S. Yu. "Active Learning: A Survey". In: *Data Classification*. Chapman and Hall/CRC, 2014, pp. 599–634. DOI: 10.1201/b17320 (Cited on p. 44).

[ALV22]   Nima Anari, Yang P. Liu, and Thuy-Duong Vuong. "Optimal Sublinear Sampling of Spanning Trees and Determinantal Point Processes via Average-Case Entropic Independence". In: *Proceedings of Sixty-Third Annual Symposium on Foundations of Computer Science*. Oct. 2022, pp. 123–134. DOI: 10.1109/FOCS54457.2022.00019 (Cited on p. 56).

[AM15]   Ahmed Alaoui and Michael W Mahoney. "Fast Randomized Kernel Ridge Regression with Statistical Guarantees". In: *Advances in Neural Information Processing Systems*. Vol. 28. 2015. URL: https://dl.acm.org/doi/10.5555/2969239.2969326 (Cited on p. 51).

[AMT10]   Haim Avron, Petar Maymounkov, and Sivan Toledo. "Blendenpik: Supercharging LAPACK's Least-Squares Solver". In: *SIAM Journal on Scientific Computing* 32.3 (Jan. 2010), pp. 1217–1236. DOI: 10.1137/090767911 (Cited on pp. 1, 4, 305, 320, 322, 324, 377).

[And05]   Tsuyoshi Ando. "Schur Complements and Matrix Inequalities: Operator-Theoretic Approach". In: *The Schur Complement and Its Applications*. Ed. by Fuzhen Zhang. Numerical Methods and Algorithms. Springer, 2005, pp. 137–162. DOI: 10.1007/0-387-24273-2_6 (Cited on p. 22).

[AOR16]   Nima Anari, Shayan Oveis Gharan, and Alireza Rezaei. "Monte Carlo Markov Chain Algorithms for Sampling Strongly Rayleigh Distributions and Determinantal Point Processes". In: *Conference on Learning Theory*. June 2016, pp. 103–115. URL: https://proceedings.mlr.press/v49/anari16.html (Cited on p. 56).

[Aro50]   Nachman Aronszajn. "Theory of Reproducing Kernels". In: *Transactions of the American Mathematical Society* 68.3 (1950), pp. 337–404. DOI: 10.2307/1990404 (Cited on p. 77).

[AS17]   Guillaume Aubrun and Stanisław J. Szarek. *Alice and Bob Meet Banach: The Interface of Asymptotic Geometric Analysis and Quantum Information Theory*. Vol. 223. Mathematical Surveys and Monographs. American Mathematical Society, 2017. DOI: 10.1090/surv/223 (Cited on p. 372).

[ASE14]   Erlend Aune, Daniel P. Simpson, and Jo Eidsvik. "Parameter Estimation in High Dimensional Gaussian Distributions". In: *Statistics and Computing* 24 (2014), pp. 247–263. DOI: 10.1007/s11222-012-9368-y (Cited on p. 252).

[Bac17]   Francis Bach. "On the Equivalence between Kernel Quadrature Rules and Random Feature Expansions". In: *The Journal of Machine Learning Research* 18.1 (2017), pp. 714–751. DOI: https://dl.acm.org/doi/10.5555/3122009.3122030 (Cited on p. 106).

[BBC19]   Ayoub Belhadji, Rémi Bardenet, and Pierre Chainais. "Kernel Quadrature with DPPs". In: *Advances in Neural Information Processing Systems*. Vol. 32. 2019. URL: https://dl.acm.org/doi/10.5555/3454287.3455445 (Cited on p. 106).

[BBC20]   Ayoub Belhadji, Rémi Bardenet, and Pierre Chainais. "Kernel Interpolation with Continuous Volume Sampling". In: *Proceedings of the Thirty-Seventh International Conference on Machine Learning*. Nov. 2020, pp. 725–735. URL: https://dl.acm.org/doi/10.5555/3524938.3525006 (Cited on p. 124).

[BBCF+12]   Ronald Babich, Richard C. Brower, Michael A. Clark, George T. Fleming, James C. Osborn, Claudio Rebbi, and David Schaich. "Exploring Strange Nucleon Form Factors on the Lattice". In: *Physical Review D* 85.5 (Mar. 2012), p. 054510. DOI: 10.1103/PhysRevD.85.054510 (Cited on p. 252).

[BCW20]   Ainesh Bakshi, Nadiia Chepurko, and David P. Woodruff. "Robust and Sample Optimal Algorithms for PSD Low Rank Approximation". In: *Proceedings of the Sixty-First Annual IEEE Symposium on Foundations of Computer Science*. Nov. 2020, pp. 506–516. DOI: 10.1109/FOCS46700.2020.00054 (Cited on p. 184).

[BDHO24]   Pau Batlle, Matthieu Darcy, Bamdad Hosseini, and Houman Owhadi. "Kernel Methods Are Competitive for Operator Learning". In: *Journal of Computational Physics* 496 (Jan. 2024), p. 112549. DOI: 10.1016/j.jcp.2023.112549 (Cited on p. 2).

[BEHW87]   Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. "Occam's Razor". In: *Information processing letters* 24.6 (1987), pp. 377–380 (Cited on p. 73).

[Bel21]   Mikhail Belkin. "Fit without Fear: Remarkable Mathematical Phenomena of Deep Learning through the Prism of Interpolation". In: *Acta Numerica* 30 (May 2021), pp. 203–248. DOI: 10.1017/S0962492921000039 (Cited on p. 87).

[BG65]   Peter Businger and Gene H. Golub. "Linear Least Squares Solutions by Householder Transformations". In: *Numerische Mathematik* 7.3 (June 1965), pp. 269–276. DOI: 10.1007/BF01436084 (Cited on p. 314).

[Bis06]   Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. New York: Springer, 2006. URL: https://dl.acm.org/doi/10.5555/1162264 (Cited on p. 87).

[BJT23]   Francis Bloch, Matthew O. Jackson, and Pietro Tebaldi. "Centrality Measures in Networks". In: *Social Choice and Welfare* 61.2 (Aug. 2023), pp. 413–453. DOI: 10.1007/s00355-023-01456-4 (Cited on p. 192).

[BKM22]   Vladimir Braverman, Aditya Krishnan, and Christopher Musco. "Sublinear Time Spectral Density Estimation". In: *Proceedings of the Fifty-Fourth Annual ACM SIGACT Symposium on Theory of*

Computing. June 2022, pp. 1144–1157. DOI: `10.1145/3519935` `.3520009` (Cited on p. 202).

[BKS07]   Costas Bekas, Effrosyni Kokiopoulou, and Yousef Saad. "An Estimator for the Diagonal of a Matrix". In: *Applied Numerical Mathematics*. Numerical Algorithms, Parallelism and Applications (2) 57.11 (Nov. 2007), pp. 1214–1229. DOI: `10.1016/j.apnum.200` `7.01.003` (Cited on pp. 192, 197, 254).

[BKW25]   Bernhard Beckermann, Daniel Kressner, and Heather Wilber. *Compression Properties for Large Toeplitz-like Matrices*. Feb. 2025. arXiv: `2502.09823v1` (Cited on p. 178).

[BLLT20]   Peter L. Bartlett, Philip M. Long, Gábor Lugosi, and Alexander Tsigler. "Benign Overfitting in Linear Regression". In: *Proceedings of the National Academy of Sciences* 117.48 (Dec. 2020), pp. 30063–30070. DOI: `10.1073/pnas.1907378117` (Cited on p. 87).

[BLM13]   Stéphane Boucheron, Gábor Lugosi, and Pascal Massart. *Concentration Inequalities: A Nonasymptotic Theory of Independence*. Oxford University Press, 2013. DOI: `10.1093/acprof:oso/978` `0199535255.001.0001` (Cited on p. 291).

[BMR22]   Claude Brezinski, Gérard Meurant, and Michela Redivo-Zaglia. *A Journey through the History of Numerical Linear Algebra*. Society for Industrial and Applied Mathematics, 2022. DOI: `10.1137/1.9` `781611977233` (Cited on pp. 305–307).

[BN22]   Robert A. Baston and Yuji Nakatsukasa. *Stochastic Diagonal Estimation: Probabilistic Bounds and an Improved Algorithm*. Jan. 2022. arXiv: `2201.10684v1` (Cited on pp. 254, 255).

[BP20]   Burak Bartan and Mert Pilanci. *Distributed Sketching Methods for Privacy Preserving Regression*. June 2020. arXiv: `2002.06538v2` (Cited on p. 393).

[BR03]   Mario Bebendorf and Sergej Rjasanow. "Adaptive Low-Rank Approximation of Collocation Matrices". In: *Computing* 70.1 (Feb. 2003), pp. 1–24. DOI: `10.1007/s00607-002-1469-6` (Cited on pp. 166, 177).

[BR24]   Mathieu Blondel and Vincent Roulet. *The Elements of Differentiable Programming*. July 2024. arXiv: `2403.14606v2` (Cited on p. 188).

[BSW14]   Pierre Baldi, Peter Sadowski, and Daniel Whiteson. "Searching for Exotic Particles in High-Energy Physics with Deep Learning". In: *Nature Communications* 5.1 (July 2014), p. 4308. DOI: `10.1038` `/ncomms5308` (Cited on p. 103).

[BT19]   Bernhard Beckermann and Alex Townsend. "Bounds on the Singular Values of Matrices with Displacement Structure". In: *SIAM Review* 61.2 (Jan. 2019), pp. 319–344. DOI: `10.1137/19M124443 3` (Cited on p. 12).

[Bv24]   Tatiana Brailovskaya and Ramon van Handel. "Universality and Sharp Matrix Concentration Inequalities". In: *Geometric and Functional Analysis* 34.6 (Dec. 2024), pp. 1734–1838. DOI: `10.1007 /s00039-024-00692-9` (Cited on p. 372).

[BW09a]   Mohamed-Ali Belabbas and Patrick J. Wolfe. "On Landmark Selection and Sampling in High-Dimensional Data Analysis". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 367.1906 (Nov. 2009), pp. 4295–4312. DOI: `10.1098/rsta.2009.0161` (Cited on pp. 27, 29).

[BW09b]   Mohamed-Ali Belabbas and Patrick J. Wolfe. "Spectral Methods in Machine Learning and New Strategies for Very Large Datasets". In: *Proceedings of the National Academy of Sciences* 106.2 (Jan. 2009), pp. 369–374. DOI: `10.1073/pnas.0810600105` (Cited on p. 55).

[BW17]   Christos Boutsidis and David P. Woodruff. "Optimal CUR Matrix Decompositions". In: *SIAM Journal on Computing* 46.2 (Jan. 2017), pp. 543–589. DOI: `10.1137/140977898` (Cited on p. 177).

[BY93]   Z. D. Bai and Y. Q. Yin. "Limit of the Smallest Eigenvalue of a Large Dimensional Sample Covariance Matrix". In: *The Annals of Probability* 21.3 (July 1993), pp. 1275–1294. DOI: `10.1214/aop /1176989118` (Cited on p. 370).

[BZCX+03] Dongbo Bu, Yi Zhao, Lun Cai, Hong Xue, Xiaopeng Zhu, Hongchao Lu, Jingfen Zhang, Shiwei Sun, Lunjiang Ling, Nan Zhang, Guojie Li, and Runsheng Chen. "Topological Structure Analysis of the Protein–Protein Interaction Network in Budding Yeast". In: *Nucleic Acids Research* 31.9 (May 2003), pp. 2443–2450. DOI: `10.1093/nar/gkg340` (Cited on pp. 235, 262).

[Car10]   Eric Carlen. "Trace Inequalities and Quantum Entropy: An Introductory Course". In: *Contemporary Mathematics*. Ed. by Robert Sims and Daniel Ueltschi. Vol. 529. American Mathematical Society, 2010, pp. 73–140. DOI: `10.1090/conm/529/10428` (Cited on p. 65).

[CCBG22]   Yuan Cao, Zixiang Chen, Misha Belkin, and Quanquan Gu. "Benign Overfitting in Two-layer Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems* 35 (Dec.

2022), pp. 25237–25250. URL: https://dl.acm.org/doi/10
.5555/3600270.3602100 (Cited on p. 87).

[CCF04]   Moses Charikar, Kevin Chen, and Martin Farach-Colton. "Finding
          Frequent Items in Data Streams". In: *Theoretical Computer Sci-
          ence*. Automata, Languages and Programming 312.1 (Jan. 2004),
          pp. 3–15. DOI: 10.1016/S0304-3975(03)00400-6 (Cited on
          p. 383).

[CD13]   Jiawei Chiu and Laurent Demanet. "Sublinear Randomized Algo-
          rithms for Skeleton Decompositions". In: *SIAM Journal on Matrix
          Analysis and Applications* 34.3 (Jan. 2013), pp. 1361–1383. DOI:
          10.1137/110852310 (Cited on p. 35).

[CDD25a] Shabarish Chenakkod, Michał Dereziński, and Xiaoyu Dong. *Op-
          timal Oblivious Subspace Embeddings with Near-optimal Sparsity*.
          To appear in ICALP 2025. Apr. 2025. arXiv: 2411.08773v2 (Cited
          on pp. 319, 372, 382, 388).

[CDD25b] Shabarish Chenakkod, Michał Dereziński, and Xiaoyu Dong. *Opti-
          mal Subspace Embeddings: Resolving Nelson-Nguyen Conjecture
          Up to Sub-Polylogarithmic Factors*. Aug. 2025. DOI: 10.48550/a
          rXiv.2508.14234. eprint: 2508.14234 (Cited on p. 391).

[CDDR24] Shabarish Chenakkod, Michał Dereziński, Xiaoyu Dong, and Mark
          Rudelson. "Optimal Embedding Dimension for Sparse Subspace
          Embeddings". In: *Proceedings of the Fifty-Sixth Annual ACM Sym-
          posium on Theory of Computing*. June 2024, pp. 1106–1117. DOI:
          10.1145/3618260.3649762 (Cited on pp. 319, 367, 372, 388).

[CDV20]   Daniele Calandriello, Michal Derezinski, and Michal Valko. "Sam-
          pling from a k-DPP without Looking at All Items". In: *Advances in
          Neural Information Processing Systems*. Vol. 33. 2020, pp. 6889–
          6899. URL: https://dl.acm.org/doi/10.5555/3495724.34
          96302 (Cited on p. 56).

[CEMT25] Chris Camaño, Ethan N. Epperly, Raphael A. Meyer, and Joel A.
          Tropp. *Faster Linear Algebra Algorithms with Structured Ran-
          dom Matrices*. Aug. 2025. DOI: 10.48550/arXiv.2508.21189.
          eprint: 2508.21189 (cs) (Cited on pp. 385, 391).

[CET25]   Chris Camaño, Ethan N. Epperly, and Joel A. Tropp. *Successive
          Randomized Compression: A Randomized Algorithm for the Com-
          pressed MPO–MPS Product*. Apr. 2025. arXiv: 2504.06475v1
          (Cited on p. 7).

[CETW25] Yifan Chen, Ethan N. Epperly, Joel A. Tropp, and Robert J. Webber.
          "Randomly Pivoted Cholesky: Practical Approximation of a Kernel
          Matrix with Few Entry Evaluations". In: *Communications on Pure
          and Applied Mathematics* 78.5 (2025), pp. 995–1041. DOI: 10.10

`02/cpa.22234` (Cited on pp. 31, 43, 47, 49, 53, 57, 60, 61, 68, 69, 93, 101, 111, 125, 139, 143, 147, 158, 161, 181, 182).

[CF79]   Randall E. Cline and Robert E. Funderlic. "The Rank of a Difference of Matrices and Associated Generalized Inverses". In: *Linear Algebra and its Applications* 24 (Apr. 1979), pp. 185–215. DOI: `10.1016/0024-3795(79)90158-7` (Cited on p. 166).

[CFG95]   Moody T. Chu, Robert E. Funderlic, and Gene H. Golub. "A Rank–One Reduction Formula and Its Applications to Matrix Factorizations". In: *SIAM Review* 37.4 (Dec. 1995), pp. 512–530. DOI: `10.1137/1037124` (Cited on p. 166).

[CFS21]   Coralia Cartis, Jan Fiala, and Zhen Shao. *Hashing Embeddings of Optimal Dimension, with Applications to Linear Least Squares*. May 2021. arXiv: `2105.11815v1` (Cited on pp. 320, 383).

[CGSX+08] Shivkumar Chandrasekaran, Ming Gu, X. Sun, Jianlin Xia, and Jiang Zhu. "A Superfast Algorithm for Toeplitz Systems of Linear Equations". In: *SIAM Journal on Matrix Analysis and Applications* 29.4 (2008), pp. 1247–1266. DOI: `10.1137/040617200` (Cited on pp. 32, 178).

[CH23]   Tyler Chen and Eric Hallman. "Krylov-Aware Stochastic Trace Estimation". In: *SIAM Journal on Matrix Analysis and Applications* 44.3 (Sept. 2023), pp. 1218–1244. DOI: `10.1137/22M1494257` (Cited on pp. 190, 251).

[Che22]   Tyler Chen. "Lanczos-Based Methods for Matrix Functions". PhD thesis. University of Washington, 2022. URL: `https://research.chen.pw/thesis.pdf` (Cited on pp. 194, 196).

[Che24]   Tyler Chen. *The Lanczos Algorithm for Matrix Functions: A Handbook for Scientists*. Oct. 2024. arXiv: `2410.11090v1` (Cited on pp. 188, 235, 249, 250, 262, 271, 278, 333, 335).

[Che78]   Simone Chevet. "Séries de variables aléatoires gaussiennes à valeurs dans $E \widehat{\otimes}_\varepsilon F$. Application aux produits d'espaces de Wiener abstraits". In: *Séminaire Maurey-Schwartz* (1977–1978). talk:19, pp. 1–15. URL: `https://www.numdam.org/item/SAF_1977-1978____A14_0/` (Cited on p. 372).

[CK06]   Harr Chen and David R. Karger. "Less Is More: Probabilistic Models for Retrieving Fewer Relevant Documents". In: *Proceedings of the Twenty-Ninth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. Aug. 2006, pp. 429–436. DOI: `10.1145/1148170.1148245` (Cited on p. 44).

[CK24]   Alice Cortinovis and Daniel Kressner. *Adaptive Randomized Pivoting for Column Subset Selection, DEIM, and Low-Rank Approximation*. Dec. 2024. arXiv: 2412.13992v1 (Cited on pp. 29, 155, 161, 164, 175, 177, 179).

[CLV17]   Daniele Calandriello, Alessandro Lazaric, and Michal Valko. "Distributed Adaptive Sampling for Kernel Matrix Approximation". In: *Proceedings of the Twentieth International Conference on Artificial Intelligence and Statistics*. Apr. 2017, pp. 1421–1429. URL: https://proceedings.mlr.press/v54/calandriello17a.html (Cited on p. 52).

[ÇM09]   Ali Çivril and Malik Magdon-Ismail. "On Selecting a Maximum Volume Sub-Matrix of a Matrix and Related Problems". In: *Theoretical Computer Science* 410.47-49 (Nov. 2009), pp. 4801–4811. DOI: 10.1016/j.tcs.2009.06.018 (Cited on p. 54).

[CNRS+25]   Tyler Chen, Pradeep Niroula, Archan Ray, Pragna Subrahmanya, Marco Pistoia, and Niraj Kumar. *GPU-Parallelizable Randomized Sketch-and-Precondition for Linear Regression Using Sparse Sign Sketches*. June 2025. DOI: 10.48550/arXiv.2506.03070. eprint: 2506.03070 (Cited on p. 385).

[Coh16]   Michael B. Cohen. "Nearly Tight Oblivious Subspace Embeddings by Trace Inequalities". In: *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, Jan. 2016, pp. 278–287. DOI: 10.1137/1.9781611974331.ch21 (Cited on pp. 318, 387, 388).

[CP19]   Xue Chen and Eric Price. "Active Regression via Linear-Sample Sparsification". In: *Proceedings of the Thirty-Second Conference on Learning Theory*. June 2019, pp. 663–695. URL: https://proceedings.mlr.press/v99/chen19a.html (Cited on pp. 103, 168, 175, 269).

[CRBD18]   Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. "Neural Ordinary Differential Equations". In: *Advances in Neural Information Processing Systems*. Vol. 31. 2018. URL: https://dl.acm.org/doi/10.5555/3327757.3327764 (Cited on p. 191).

[CTU25]   Tyler Chen, Thomas Trogdon, and Shashanka Ubaru. "Randomized Matrix-Free Quadrature: Unified and Uniform Bounds for Stochastic Lanczos Quadrature and the Kernel Polynomial Method". In: *SIAM Journal on Scientific Computing* 47.3 (June 2025), A1733–A1757. DOI: 10.1137/23M1600414 (Cited on p. 250).

[CW09]   Kenneth L. Clarkson and David P. Woodruff. "Numerical Linear Algebra in the Streaming Model". In: *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*. May 2009, pp. 205–214. DOI: 10.1145/1536414.1536445 (Cited on p. 166).

[CW17]   Kenneth L. Clarkson and David P. Woodruff. "Low-Rank Approximation and Regression in Input Sparsity Time". In: *Journal of the ACM* 63.6 (Feb. 2017), pp. 1–45. DOI: 10.1145/3019134 (Cited on p. 175).

[CY25]   Alice Cortinovis and Lexing Ying. "A Sublinear-Time Randomized Algorithm for Column and Row Subset Selection Based on Strong Rank-Revealing QR Factorizations". In: *SIAM Journal on Matrix Analysis and Applications* (Mar. 2025), pp. 22–44. DOI: 10.1137/24M164063X (Cited on p. 35).

[DCMP24]   Yijun Dong, Chao Chen, Per-Gunnar Martinsson, and Katherine Pearce. *Robust Blockwise Random Pivoting: Fast and Accurate Adaptive Interpolative Decomposition*. Dec. 2024. arXiv: 2309.16002v5 (Cited on pp. 125, 131, 150, 152, 154, 157, 177).

[DCV19]   Michal Derezinski, Daniele Calandriello, and Michal Valko. "Exact Sampling of Determinantal Point Processes with Sublinear Time Preprocessing". In: *Advances in Neural Information Processing Systems*. Vol. 32. 2019. URL: https://dl.acm.org/doi/10.5555/3454287.3455323 (Cited on p. 56).

[Dd89]   Hans De Raedt and Pedro de Vries. "Simulation of Two and Three-Dimensional Disordered Systems: Lifshitz Tails and Localization Properties". In: *Zeitschrift für Physik B Condensed Matter* 77.2 (June 1989), pp. 243–251. DOI: 10.1007/BF01313668 (Cited on p. 194).

[DEFT+24] Mateo Díaz, Ethan N. Epperly, Zachary Frangella, Joel A. Tropp, and Robert J. Webber. *Robust, Randomized Preconditioning for Kernel Ridge Regression*. July 2024. arXiv: 2304.12465v5 (Cited on pp. 93, 96, 97, 99).

[DELZ24]   Zhiyan Ding, Ethan N. Epperly, Lin Lin, and Ruizhe Zhang. "The ESPRIT Algorithm under High Noise: Optimal Error Scaling and Noisy Super-Resolution". In: *Sixty-Fifth IEEE Annual Symposium on Foundations of Computer Science*. 2024, pp. 2344–2366. DOI: 10.1109/FOCS61266.2024.00137 (Cited on p. 7).

[Dem97]   James W. Demmel. *Applied Numerical Linear Algebra*. Society for Industrial and Applied Mathematics, 1997. DOI: 10.1137/1.9781611971446 (Cited on pp. 297, 335).

[Der18]   Manfred K. Dereziński Michałand Warmuth. "Reverse Iterative
             Volume Sampling for Linear Regression". In: *The Journal of Ma-
             chine Learning Research* 19.1 (2018), pp. 853–891. URL: https:
             //dl.acm.org/doi/10.5555/3291125.3291148 (Cited on
             p. 175).

[Der19]   Michał Dereziński. "Fast Determinantal Point Processes via
             Distortion-Free Intermediate Sampling". In: *Conference on Learn-
             ing Theory*. June 2019, pp. 1029–1049. URL: https://proceedi
             ngs.mlr.press/v99/derezinski19a.html (Cited on p. 56).

[DFFG+03]   Jack Dongarra, Ian Foster, Geoffrey Fox, William Gropp, Ken
             Kennedy, Linda Torczon, and Andy White, eds. *Sourcebook of Par-
             allel Computing*. Morgan Kaufmann Publishers Inc., 2003. URL:
             https://dl.acm.org/doi/10.5555/941480 (Cited on
             p. 126).

[DGK98]   Vladamir Druskin, Anne Greenbaum, and Leonid Knizhnerman.
             "Using Nonorthogonal Lanczos Vectors in the Computation of
             Matrix Functions". In: *SIAM Journal on Scientific Computing*
             19.1 (Jan. 1998), pp. 38–54. DOI: 10.1137/S1064827596303661
             (Cited on pp. 335, 349, 355).

[DH18]   Manfred K. Dereziński Michałand Warmuth and Daniel J. Hsu.
             "Leveraged Volume Sampling for Linear Regression". In: *Thirty-
             Second Conference on Neural Information Processing Systems*.
             2018, p. 10. URL: https://dl.acm.org/doi/10.5555/33271
             44.3327176 (Cited on pp. 175, 395).

[DK91]   Vladamir L. Druskin and Leonid A. Knizhnerman. "Error Bounds
             in the Simple Lanczos Procedure for Computing Functions of Sym-
             metric Matrices and Eigenvalues". In: *Computational Mathematics
             and Mathematical Physics* 31.7 (1991), pp. 20–30. URL: https:
             //dl.acm.org/doi/abs/10.5555/151905.151908 (Cited on
             pp. 335, 349).

[DKM06]   Petros Drineas, Ravi Kannan, and Michael W. Mahoney. "Fast
             Monte Carlo Algorithms for Matrices III: Computing a Com-
             pressed Approximate Matrix Decomposition". In: *SIAM Journal
             on Computing* 36.1 (Jan. 2006), pp. 184–206. DOI: 10.1137/S00
             97539704442702 (Cited on p. 177).

[DKM20]   Michał Dereziński, Rajiv Khanna, and Michael W Mahoney. "Im-
             proved Guarantees and a Multiple-Descent Curve for Column Sub-
             set Selection and the Nyström Method". In: *Advances in Neural
             Information Processing Systems* 33 (2020). URL: https://dl.ac
             m.org/doi/10.5555/3495724.3496140 (Cited on p. 29).

[DL19]   Edgar Dobriban and Sifan Liu. "Asymptotics for Sketching in Least Squares Regression". In: *Advances in Neural Information Processing Systems*. Vol. 32. 2019. URL: https://dl.acm.org/doi/10.5555/3454287.3454617 (Cited on p. 372).

[DM05]   Petros Drineas and Michael W. Mahoney. "On the Nyström Method for Approximating a Gram Matrix for Improved Kernel-Based Learning". In: *Journal of Machine Learning Research* 6.72 (2005), pp. 2153–2175. URL: https://jmlr.org/papers/v6/drineas05a.html (Cited on pp. 47, 48).

[DM16]   Petros Drineas and Michael W. Mahoney. "RandNLA: Randomized Numerical Linear Algebra". In: *Communications of the ACM* 59.6 (May 2016), pp. 80–90. DOI: 10.1145/2842602 (Cited on p. 2).

[DM21]   Michał Dereziński and Michael W. Mahoney. "Determinantal Point Processes in Randomized Numerical Linear Algebra". In: *Notices of the American Mathematical Society* 68.01 (Jan. 2021), p. 1. DOI: 10.1090/noti2202 (Cited on p. 55).

[DM22]   Raaz Dwivedi and Lester Mackey. "Generalized Kernel Thinning". In: *International Conference on Learning Representations*. Available at https://arxiv.org/abs/2110.01593v8. Jan. 2022 (Cited on pp. 123, 124).

[DM23a]   Prathamesh Dharangutte and Christopher Musco. "A Tight Analysis of Hutchinson's Diagonal Estimator". In: *2023 Symposium on Simplicity in Algorithms*. Society for Industrial and Applied Mathematics, Jan. 2023, pp. 353–364. DOI: 10.1137/1.9781611977585.ch32 (Cited on pp. 154, 159, 164, 254).

[DM23b]   Yijun Dong and Per-Gunnar Martinsson. "Simpler Is Better: A Comparative Study of Randomized Pivoting Algorithms for CUR and Interpolative Decompositions". In: *Advances in Computational Mathematics* 49.4 (Aug. 2023), p. 66. DOI: 10.1007/s10444-023-10061-z (Cited on pp. 152, 177, 365, 385).

[DMBS79]   Jack J. Dongarra, Cleve B. Moler, James R. Bunch, and G. W. Stewart. *LINPACK Users' Guide*. Society for Industrial and Applied Mathematics, Jan. 1979. DOI: 10.1137/1.9781611971811 (Cited on p. 41).

[DMM06]   Petros Drineas, Michael W. Mahoney, and S. Muthukrishnan. "Sampling Algorithms for ℓ_2 Regression and Applications". In: *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*. Jan. 2006, pp. 1127–1136. URL: https://dl.acm.org/doi/10.5555/1109557.1109682 (Cited on pp. 316, 319).

[DMM08]   Petros Drineas, Michael W. Mahoney, and S. Muthukrishnan. "Relative-Error *CUR* Matrix Decompositions". In: *SIAM Journal on Matrix Analysis and Applications* 30.2 (Jan. 2008), pp. 844–881. DOI: `10.1137/07070471X` (Cited on p. 176).

[DMMS11]   Petros Drineas, Michael W. Mahoney, S. Muthukrishnan, and Tamás Sarlós. "Faster Least Squares Approximation". In: *Numerische Mathematik* 117.2 (Feb. 2011), pp. 219–249. DOI: `10.1007/s00211-010-0331-6` (Cited on p. 393).

[DMMW12]   Petros Drineas, Malik Magdon-Ismail, Michael W. Mahoney, and David P. Woodruff. "Fast Approximation of Matrix Coherence and Statistical Leverage". In: *The Journal of Machine Learning Research* 13.1 (2012), pp. 3475–3506. URL: `https://dl.acm.org/doi/10.5555/2503308.2503352` (Cited on pp. 193, 269, 270).

[DPPL24]   Yijun Dong, Xiang Pan, Hoang Phan, and Qi Lei. "Randomly Pivoted V-optimal Design: Fast Data Selection under Low Intrinsic Dimension". In: *Workshop on Machine Learning and Compression, NeurIPS 2024*. 2024. URL: `https://openreview.net/forum?id=WPvQVQrbch` (Cited on p. 68).

[DRVW06]   Amit Deshpande, Luis Rademacher, Santosh Vempala, and Grant Wang. "Matrix Approximation and Projective Clustering via Volume Sampling". In: *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*. 2006, pp. 1117–1126. DOI: `10.1145/1109557.1109681` (Cited on pp. 43, 55, 131, 142, 147, 158, 159, 175, 395).

[DS01]   Kenneth R. Davidson and Stanislaw J. Szarek. "Chapter 8 - Local Operator Theory, Random Matrices and Banach Spaces". In: *Handbook of the Geometry of Banach Spaces*. Ed. by W. B. Johnson and J. Lindenstrauss. Vol. 1. Handbook of the Geometry of Banach Spaces. Elsevier Science B.V., Jan. 2001, pp. 317–366. DOI: `10.1016/S1874-5849(01)80010-3` (Cited on p. 372).

[DT12]   Alex Druinsky and Sivan Toledo. *How Accurate is `inv(A)*b`?* Jan. 2012. arXiv: `1201.6035v1` (Cited on p. 170).

[DV06]   Amit Deshpande and Santosh Vempala. "Adaptive Sampling and Fast Low-Rank Matrix Approximation". In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Lecture Notes in Computer Science. 2006, pp. 292–303. DOI: `10.1007/11830924_28` (Cited on pp. 142, 147, 158–160).

[DW17]   Michał Dereziński and Manfred K. Warmuth. "Unbiased Estimates for Linear Regression via Volume Sampling". In: *Advances in Neural Information Processing Systems* 30 (2017). URL: https://dl.acm.org/doi/10.5555/3294996.3295068 (Cited on p. 175).

[Eck87]   Roger Eckhardt. "Stan Ulam, John von Neumann, and the Monte Carlo Method". In: *Los Alamos Science, Special Issue* 15 (1987), pp. 131–137. URL: https://mcnp.lanl.gov/pdf_files/Article_1987_LAS_Eckhardt_131--141.pdf (Cited on p. 1).

[Efr82]   Bradley Efron. *The Jackknife, the Bootstrap and Other Resampling Plans*. Society for Industrial and Applied Mathematics, 1982. DOI: 10.1137/1.9781611970319 (Cited on p. 287).

[EGN25]   Ethan N. Epperly, Anne Greenbaum, and Yuji Nakatsukasa. *Stable Algorithms for General Linear Systems by Preconditioning the Normal Equations*. Mar. 2025. arXiv: 2502.17767v2 (Cited on p. 356).

[EGW25]   Ethan N. Epperly, Gil Goldshlager, and Robert J. Webber. *Randomized Kaczmarz with Tail Averaging*. Apr. 2025. arXiv: 2411.19877v3 (Cited on p. 7).

[ELN22]   Ethan N. Epperly, Lin Lin, and Yuji Nakatsukasa. "A Theory of Quantum Subspace Diagonalization". In: *SIAM Journal on Matrix Analysis and Applications* 43.3 (2022), pp. 1263–1290. DOI: 10.1137/21M145954X (Cited on p. 5).

[EM23]   Ethan N. Epperly and Elvira Moreno. "Kernel Quadrature with Randomly Pivoted Cholesky". In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 65850–65868. URL: https://dl.acm.org/doi/10.5555/3666122.3668997 (Cited on pp. 105, 106, 111, 115, 116, 124).

[EMN25]   Ethan N. Epperly, Maike Meier, and Yuji Nakatsukasa. "Fast Randomized Least-Squares Solvers Can Be Just as Accurate and Stable as Classical Direct Solvers". In: *Communications on Pure and Applied Mathematics, accepted* (2025). Preprint available at https://arxiv.org/abs/2406.03468v2 (Cited on pp. 311, 327, 336, 339, 340, 343–345, 349, 352, 355, 365, 385).

[Epp22a]   Ethan N. Epperly. *Low-Rank Approximation Toolbox: Nyström Approximation*. Oct. 2022. URL: https://www.ethanepperly.com/index.php/2022/10/11/low-rank-approximation-toolbox-nystrom-approximation/ (Cited on p. 22).

[Epp22b]   Ethan N. Epperly. *Note to Self: Hanson–Wright Inequality*. Oct. 2022. URL: https://www.ethanepperly.com/index.php/2

`022/10/04/note-to-self-hanson-wright-inequality/` (Cited on p. 208).

[Epp23a]   Ethan N. Epperly. *Five Interpretations of Kernel Quadrature*. Dec. 2023. URL: `https://www.ethanepperly.com/index.php/2023/12/01/five-interpretations-of-kernel-quadrature/` (Cited on p. 119).

[Epp23b]   Ethan N. Epperly. *Stochastic Trace Estimation*. Jan. 2023. URL: `https://www.ethanepperly.com/index.php/2023/01/26/stochastic-trace-estimation/` (Cited on pp. 200, 201).

[Epp23c]   Ethan N. Epperly. *Which Sketch Should I Use?* Nov. 2023. URL: `https://www.ethanepperly.com/index.php/2023/11/27/which-sketch-should-i-use/` (Cited on pp. 365, 385).

[Epp24a]   Ethan N. Epperly. *Don't Use Gaussians in Stochastic Trace Estimation*. Jan. 2024. URL: `https://www.ethanepperly.com/index.php/2024/01/28/dont-use-gaussians-in-stochastic-trace-estimation/` (Cited on p. 201).

[Epp24b]   Ethan N. Epperly. "Fast and Forward Stable Randomized Algorithms for Linear Least-Squares Problems". In: *SIAM Journal on Matrix Analysis and Applications* (Dec. 2024), pp. 1782–1804. DOI: `10.1137/23M1616790` (Cited on pp. 311, 327, 331–333, 335, 365).

[Epp24c]   Ethan N. Epperly. *Low-Rank Approximation Toolbox: The Gram Correspondence*. Dec. 2024. URL: `https://www.ethanepperly.com/index.php/2024/12/07/low-rank-approximation-toolbox-the-gram-correspondence/` (Cited on pp. 11, 29).

[Epp24d]   Ethan N. Epperly. *Note to Self: Sketch-and-Solve with a Gaussian Embedding*. Nov. 2024. URL: `https://www.ethanepperly.com/index.php/2024/11/19/note-to-self-sketch-and-solve-with-a-gaussian-embedding/` (Cited on p. 393).

[Epp25]   Ethan N. Epperly. *Note to Self: How Accurate Is Sketch and Solve?* Feb. 2025. URL: `https://www.ethanepperly.com/index.php/2025/02/12/note-to-self-how-accurate-is-sketch-and-solve/` (Cited on p. 393).

[ES81]   Bradley Efron and Charles Stein. "The Jackknife Estimate of Variance". In: *The Annals of Statistics* 9.3 (1981), pp. 586–596. DOI: `10.1214/aos/1176345462` (Cited on p. 290).

[Est22]   Ernesto Estrada. "The Many Facets of the Estrada Indices of Graphs and Networks". In: *SeMA Journal* 79.1 (Mar. 2022), pp. 57–125. DOI: `10.1007/s40324-021-00275-w` (Cited on pp. 191, 192, 234, 262).

[ET24]   Ethan N. Epperly and Joel A. Tropp. "Efficient Error and Variance Estimation for Randomized Matrix Computations". In: *SIAM Journal on Scientific Computing* 46.1 (Feb. 2024), A508–A528. DOI: 10.1137/23M1558537 (Cited on pp. 216, 280, 283, 288, 292, 294, 297, 298).

[ETW24]   Ethan N. Epperly, Joel A. Tropp, and Robert J. Webber. "XTrace: Making the Most of Every Sample in Stochastic Trace Estimation". In: *SIAM Journal on Matrix Analysis and Applications* (Mar. 2024), pp. 1–23. DOI: 10.1137/23M1548323 (Cited on pp. 24, 194, 203, 207, 212, 213, 216, 226, 230, 233, 237, 238, 240, 246–248, 253, 257, 266).

[ETW25]   Ethan N. Epperly, Joel A. Tropp, and Robert J. Webber. *Embrace Rejection: Kernel Matrix Approximation by Accelerated Randomly Pivoted Cholesky*. Apr. 2025. arXiv: 2410.03969v3 (Cited on pp. 125, 127, 130–132, 138–140, 143, 161).

[FGN06]   Reinhard Furrer, Marc G. Genton, and Douglas Nychka. "Covariance Tapering for Interpolation of Large Spatial Datasets". In: *Journal of Computational and Graphical Statistics* 15.3 (2006), pp. 502–523. JSTOR: 27594195 (Cited on p. 99).

[FKV98]   Alan Frieze, Ravi Kannan, and Santosh Vempala. "Fast Monte-Carlo Algorithms for Finding Low-Rank Approximations". In: *Proceedings of the Thirty-Ninth Annual Symposium on Foundations of Computer Science*. Nov. 1998, pp. 370–370. DOI: 10.1109/SFCS.1998.743487 (Cited on pp. 1, 34, 48, 49, 157, 316).

[FL24]   Mark Fornace and Michael Lindsey. *Column and Row Subset Selection Using Nuclear Scores: Algorithms and Theory for Nyström Approximation, CUR Decomposition, and Graph Laplacian Reduction*. Aug. 2024. arXiv: 2407.01698v2 (Cited on pp. 192, 193, 269, 271).

[FLPR99]   Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran. "Cache-Oblivious Algorithms". In: *Proceedings of the Fortieth Annual Symposium on Foundations of Computer Science*. Oct. 1999, pp. 285–297. DOI: 10.1109/SFFCS.1999.814600 (Cited on p. 127).

[FRS25]   Andreas Frommer, Michele Rinelli, and Marcel Schweitzer. "Analysis of Stochastic Probing Methods for Estimating the Trace of Functions of Sparse Symmetric Matrices". In: *Mathematics of Computation* 94.352 (2025), pp. 801–823. DOI: 10.1090/mcom/3984 (Cited on p. 252).

[FS01]   Shai Fine and Katya Scheinberg. "Efficient SVM Training Using
             Low-Rank Kernel Representations". In: *Journal of Machine Learn-
             ing Research* 2 (Dec. 2001), pp. 243–264. URL: `https://www.jm`
             `lr.org/papers/v2/fine01a.html` (Cited on pp. 39, 41, 47).

[FS11]   David Chin-Lung Fong and Michael Saunders. "LSMR: An It-
             erative Algorithm for Sparse Least-Squares Problems". In: *SIAM
             Journal on Scientific Computing* 33.5 (Jan. 2011), pp. 2950–2971.
             DOI: `10.1137/10079687X` (Cited on p. 314).

[GCBS+19]   Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, Ilya
             Sutskever, and David Duvenaud. "FFJORD: Free-form Contin-
             uous Dynamics for Scalable Reversible Generative Models". In:
             International Conference on Learning Representations. Available
             at `https://arxiv.org/abs/1810.01367v3`. 2019 (Cited on
             p. 191).

[GEL22]   Nithin Govindarajan, Ethan N. Epperly, and Lieven De Lathauwer.
             "$(L_r, L_r, 1)$-Decompositions, Sparse Component Analysis, and the
             Blind Separation of Sums of Exponentials". In: *SIAM Journal on
             Matrix Analysis and Applications* 43.2 (June 2022), pp. 912–938.
             DOI: `10.1137/21M1426444` (Cited on p. 7).

[Gil14]   Jennifer Ann Gillenwater. "Approximate Inference for Determinan-
             tal Point Processes". PhD thesis. University of Pennsylvania, 2014,
             p. 163. URL: `https://jgillenw.com/thesis.pdf` (Cited on
             p. 72).

[Gir87]   Didier A. Girard. *Un Algorithme Simple et Rapide Pour la Vali-
             dation Croisée Généralisée sur des Problèmes de Grande Taille*.
             Rapport de recherche IMAG-RR-665-M. Institut d'informatique
             et de mathématiques appliquées de Grenoble (IMAG), 1987. URL:
             `http://www-ljk.imag.fr/membres/Didier.Girard/TR-6`
             `65-M-IMAG.pdf` (Cited on pp. 2, 194, 196).

[Gir89]   Didier A. Girard. "A Fast "Monte-Carlo Cross-Validation" Pro-
             cedure for Large Least Squares Problems with Noisy Data". In:
             Numerische Mathematik 56.1 (Jan. 1989), pp. 1–23. DOI: `10.100`
             `7/BF01395775` (Cited on p. 196).

[Git11]   Alex Gittens. "The Spectral Norm Error of the Naïve Nyström
             Extension". In: (Oct. 2011). arXiv: `1110.5305v1` (Cited on pp. 27,
             29).

[Giv54]   Wallace Givens. *Numerical Computation of the Characteristic Val-
             ues of a Real Symmetric Matrix*. Tech. rep. Oak Ridge National Lab,
             1954. URL: `https://www.osti.gov/servlets/purl/441217`
             `5` (Cited on p. 310).

[GJK18]   Damien Garreau, Wittawat Jitkrittum, and Motonobu Kanagawa. *Large Sample Analysis of the Median Heuristic*. Oct. 2018. eprint: 1707.07269v3 (Cited on p. 78).

[GJT12]   Serge Gratton, Pavel Jiránek, and David Titley-Peloquin. "On the Accuracy of the Karlson–Waldén Estimate of the Backward Error for Linear Least Squares Problems". In: *SIAM Journal on Matrix Analysis and Applications* 33.3 (Jan. 2012), pp. 822–836. DOI: 10.1137/110825467 (Cited on p. 342).

[GLRE05]   Luc Giraud, Julien Langou, Miroslav Rozložník, and Jasper Van Den Eshof. "Rounding Error Analysis of the Classical Gram-Schmidt Orthogonalization Process". In: *Numerische Mathematik* 101.1 (July 2005), pp. 87–100. DOI: 10.1007/s00211-005-061 5-4 (Cited on p. 151).

[GM10]   Gene H. Golub and Gérard A. Meurant. *Matrices, Moments, and Quadrature with Applications*. Princeton Series in Applied Mathematics. Princeton University Press, 2010. JSTOR: j.ctt7tbvs (Cited on p. 250).

[GM13]   Alex Gittens and Michael Mahoney. "Revisiting the Nyström Method for Improved Large-Scale Machine Learning". In: *International Conference on Machine Learning*. May 2013, pp. 567–575. URL: https://proceedings.mlr.press/v28/gittens13.html (Cited on p. 29).

[GN86]   Alan George and Esmond Ng. "Orthogonal Reduction of Sparse Matrices to Upper Triangular Form Using Householder Transformations". In: *SIAM Journal on Scientific and Statistical Computing* 7.2 (Apr. 1986), pp. 460–472. DOI: 10.1137/0907031 (Cited on p. 314).

[Gol65]   Gene Golub. "Numerical Methods for Solving Linear Least Squares Problems". In: *Numerische Mathematik* 7.3 (June 1965), pp. 206–216. DOI: 10.1007/BF01436075 (Cited on pp. 307, 309, 310).

[Gor85]   Yehoram Gordon. "Some Inequalities for Gaussian Processes and Applications". In: *Israel Journal of Mathematics* 50.4 (1985), pp. 265–289. DOI: 10.1007/BF02759761 (Cited on p. 372).

[GPBV19]   Guillaume Gautier, Guillermo Polito, Remi Bardenet, and Michal Valko. "DPPy: DPP Sampling with Python". In: *Journal of Machine Learning Research* (2019), p. 7. URL: https://jmlr.org/papers/v20/19-179.html (Cited on p. 56).

[Grc03]   Joseph F. Grcar. "Optimal Sensitivity Analysis of Linear Least Squares". In: *Lawrence Berkeley National Laboratory, Report LBNL-52434* 99 (2003). URL: https://crd.lbl.gov/asse

ts/pubs_presos/MCS/CCSE/LBNL-52434.pdf (Cited on p. 342).

[Gre89]   Anne Greenbaum. "Behavior of Slightly Perturbed Lanczos and Conjugate-Gradient Recurrences". In: *Linear Algebra and its Applications* 113 (1989), pp. 7–63. DOI: 10.1016/0024-3795(89)90285-1 (Cited on p. 354).

[Gre97a]   Anne Greenbaum. "Estimating the Attainable Accuracy of Recursively Computed Residual Methods". In: *SIAM Journal on Matrix Analysis and Applications* 18.3 (July 1997), pp. 535–551. DOI: 10.1137/S0895479895284944 (Cited on p. 354).

[Gre97b]   Anne Greenbaum. *Iterative Methods for Solving Linear Systems*. Frontiers in Applied Mathematics 17. Society for Industrial and Applied Mathematics, 1997. DOI: 10.1137/1.9781611970937 (Cited on p. 368).

[GS12]   Venkatesan Guruswami and Ali Kemal Sinop. "Optimal Column-Based Low-Rank Matrix Reconstruction". In: *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*. Jan. 2012, pp. 1207–1214. DOI: 10.1137/1.9781611973099.95 (Cited on pp. 29, 43, 55).

[GS92]   Anne Greenbaum and Zdeněk Strakoš. "Predicting the Behavior of Finite Precision Lanczos and Conjugate Gradient Computations". In: *SIAM Journal on Matrix Analysis and Applications* 13.1 (Jan. 1992), pp. 121–137. DOI: 10.1137/0613011 (Cited on p. 354).

[GSO17]   Arjun Singh Gambhir, Andreas Stathopoulos, and Kostas Orginos. "Deflation as a Method of Variance Reduction for Estimating the Trace of a Matrix Inverse". In: *SIAM Journal on Scientific Computing* 39.2 (Jan. 2017), A532–A558. DOI: 10.1137/16M1066361 (Cited on pp. 206, 252).

[GSS07]   Joseph F. Grcar, Michael A. Saunders, and Zheng Su. *Estimates of Optimal Backward Perturbations for Linear Least Squares Problems*. Tech. rep. Lawrence Berkeley National Lab, 2007. URL: https://web.stanford.edu/group/SOL/reports/SOL-2007-1.pdf (Cited on p. 342).

[GSTV07]   Anna C. Gilbert, Martin J. Strauss, Joel A. Tropp, and Roman Vershynin. "One Sketch for All: Fast Algorithms for Compressed Sensing". In: *Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing*. June 2007, pp. 237–246. DOI: 10.1145/1250790.1250824 (Cited on p. 207).

[GT01]   Sergei Goreinov and Evgeny Tyrtyshnikov. "The Maximal-Volume Concept in Approximation by Low-Rank Matrices". In: *Contemporary Mathematics*. Ed. by Vadim Olshevsky. Vol. 280. American

Mathematical Society, 2001, pp. 47–51. DOI: `10.1090/conm/28` `0/4620` (Cited on p. 166).

[GTZ97] Sergei A. Goreinov, Evgeny E. Tyrtyshnikov, and Nikolai L. Zamarashkin. "A Theory of Pseudoskeleton Approximations". In: *Linear Algebra and its Applications* 261.1 (Aug. 1997), pp. 1–21. DOI: `10.1016/S0024-3795(96)00301-1` (Cited on p. 166).

[Gu15] Ming Gu. "Subspace Iteration Randomization and Singular Value Problems". In: *SIAM Journal on Scientific Computing* 37.3 (Jan. 2015), A1139–A1173. DOI: `10.1137/130938700` (Cited on pp. 19, 20, 23).

[GU77] Gene H. Golub and Richard Underwood. "The Block Lanczos Method for Computing Eigenvalues". In: *Mathematical Software*. Ed. by John R. Rice. Academic Press, Jan. 1977, pp. 361–377. DOI: `10.1016/B978-0-12-587260-7.50018-2` (Cited on p. 251).

[Gu98] Ming Gu. "Backward Perturbation Bounds for Linear Least Squares Problems". In: *SIAM Journal on Matrix Analysis and Applications* 20.2 (Jan. 1998), pp. 363–372. DOI: `10.1137/S08954` `79895296446` (Cited on p. 342).

[GV13] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Fourth. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins Press, 2013. URL: `https://epubs.siam.org/doi/b` `ook/10.1137/1.9781421407944` (Cited on pp. 144, 152, 188, 308, 314).

[GW66] Gene H. Golub and James H. Wilkinson. "Note on the Iterative Refinement of Least Squares Solution". In: *Numerische Mathematik* 9.2 (Dec. 1966), pp. 139–148. DOI: `10.1007/BF02166032` (Cited on pp. 310, 311, 337, 338).

[GZT97] Sergei A. Goreinov, Nikolai L. Zamarashkin, and Evgeny E. Tyrtyshnikov. "Pseudo-Skeleton Approximations by Matrices of Maximal Volume". In: *Mathematical Notes* 62.4 (Oct. 1997), pp. 515–519. DOI: `10.1007/BF02358985` (Cited on p. 166).

[Hal46] Paul R. Halmos. "The theory of unbiased estimation". In: *The Annals of Mathematical Statistics* 17.1 (Mar. 1946), pp. 34–43. DOI: `10.1214/aoms/1177731020` (Cited on p. 211).

[HB04] Matthias Hein and Olivier Bousquet. *Kernels, Associated Structures and Generalizations*. Technical Report 127. Max Planck Institute for Biological Cybernetics, July 2004. URL: `https://is.m` `pg.de/publications/2816` (Cited on pp. 106, 107).

[HG18]   Eric Hallman and Ming Gu. "LSMB: Minimizing the Backward
          Error for Least-Squares Problems". In: *SIAM Journal on Matrix
          Analysis and Applications* 39.3 (Jan. 2018), pp. 1295–1317. DOI:
          `10.1137/17M1157106` (Cited on p. 314).

[HH20]   Keaton Hamm and Longxiu Huang. "Perspectives on CUR De-
          compositions". In: *Applied and Computational Harmonic Analysis*
          48.3 (May 2020), pp. 1088–1099. DOI: `10.1016/j.acha.2019`
          `.08.006` (Cited on p. 164).

[Hig02]   Nicholas J. Higham. *Accuracy and Stability of Numerical Algo-
          rithms*. Society for Industrial and Applied Mathematics, 2002.
          DOI: `10.1137/1.9780898718027` (Cited on pp. 31, 144, 152,
          308, 309, 311, 313, 328, 336, 341, 346–348, 359).

[Hig08]   Nicholas J. Higham. *Functions of Matrices: Theory and Compu-
          tation*. Society for Industrial and Applied Mathematics, 2008. DOI:
          `10.1137/1.9780898717778` (Cited on p. 188).

[Hig22]   Nicholas J. Higham. *Seven Sins of Numerical Linear Algebra*. Oct.
          2022. URL: `https://nhigham.com/2022/10/11/seven-sins`
          `-of-numerical-linear-algebra/` (Cited on p. 233).

[Hig90]   Nicholas J. Higham. "Analysis of the Cholesky Decomposition of
          a Semi-Definite Matrix". In: *Reliable Numerical Commputation*.
          Ed. by M. G. Cox and S. Hammarling. Oxford University Press,
          Sept. 1990. DOI: `10.1093/oso/9780198535645.003.0010`
          (Cited on pp. 29, 39, 41).

[HJ12]   Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cam-
          bridge University Press, Oct. 2012. DOI: `10.1017/CBO97805118`
          `10817` (Cited on p. 34).

[HKPV06] J. Ben Hough, Manjunath Krishnapur, Yuval Peres, and Bálint
          Virág. "Determinantal Processes and Independence". In: *Proba-
          bility Surveys* 3 (Jan. 2006), pp. 206–229. DOI: `10.1214/154957`
          `806000000078` (Cited on p. 72).

[HMT11]   Nathan Halko, Per-Gunnar Martinsson, and Joel A. Tropp. "Find-
          ing Structure with Randomness: Probabilistic Algorithms for Con-
          structing Approximate Matrix Decompositions". In: *SIAM Review*
          53.2 (Jan. 2011), pp. 217–288. DOI: `10.1137/090771806` (Cited
          on pp. 1, 2, 17, 19, 20, 23, 208).

[HOL22]   Satoshi Hayakawa, Harald Oberhauser, and Terry Lyons. "Pos-
          itively Weighted Kernel Quadrature via Subsampling". In: *Ad-
          vances in Neural Information Processing Systems* 35 (Dec. 2022),
          pp. 6886–6900. DOI: `https://dl.acm.org/doi/10.5555/360`
          `0270.3600769` (Cited on pp. 123, 124).

[Hou58]   Alston S. Householder. "Unitary Triangularization of a Nonsymmetric Matrix". In: *Journal of the ACM* 5.4 (Oct. 1958), pp. 339–342. DOI: 10.1145/320941.320947 (Cited on pp. 307, 310).

[HS52]   Magnus R. Hestenes and Eduard Stiefel. "Methods of Conjugate Gradients for Solving Linear Systems". In: *Journal of Research of the National Bureau of Standards* 49.6 (1952), pp. 409–436. DOI: 10.6028/jres.049.044 (Cited on p. 314).

[Hut89]   M. F. Hutchinson. "A Stochastic Estimator of the Trace of the Influence Matrix for Laplacian Smoothing Splines". In: *Communications in Statistics - Simulation and Computation* 18.3 (May 1989), pp. 1059–1076. DOI: 10.1080/03610918908812806 (Cited on pp. 194, 196).

[Jac04]   A. Jackson. "As If Summoned from the Void, the Life of Alexandre Grothendieck". In: *Notices of the AMS* 51 (2004), pp. 1038–1056. URL: https://www.ams.org/notices/200409/fea-grothendieck-part1.pdf (Cited on p. 105).

[JL84]   William B. Johnson and Joram Lindenstrauss. "Extensions of Lipschitz Mappings into a Hilbert Space". In: *Contemporary mathematics* 26 (1984), pp. 189–206. DOI: 10.1090/conm/026/737400 (Cited on pp. 198, 269, 369).

[JWHT21]   Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: With Applications in R*. Springer Texts in Statistics. Springer US, 2021. DOI: 10.1007/978-1-0716-1418-1 (Cited on p. 51).

[Kel89]   William Thomson Baron Kelvin. *Popular Lectures and Addresses*. Macmillan and Company, 1889. DOI: 10.1017/CBO9780511997242 (Cited on p. 266).

[KHSS18]   Motonobu Kanagawa, Philipp Hennig, Dino Sejdinovic, and Bharath K. Sriperumbudur. *Gaussian Processes and Kernel Methods: A Review on Connections and Equivalences*. July 2018. arXiv: 1807.02582v1 (Cited on p. 73).

[Kle13]   A. Klenke. *Probability Theory: A Comprehensive Course*. Universitext. Springer London, 2013. DOI: 10.1007/978-3-030-56402-5 (Cited on p. 85).

[KMT12]   Sanjiv Kumar, Mehryar Mohri, and Ameet Talwalkar. "Sampling Methods for the Nyström Method". In: (2012), p. 26. URL: https://jmlr.org/papers/v13/kumar12a.html (Cited on p. 159).

[KN12]   Daniel M. Kane and Jelani Nelson. "Sparser Johnson-Lindenstrauss Transforms". In: *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*. 2012,

pp. 1195–1206. DOI: `10.1137/1.9781611973099.94` (Cited on pp. 383, 388).

[KT11]   Alex Kulesza and Ben Taskar. "*k*-DPPs: Fixed-Size Determinantal Point Processes". In: *International Conference on Machine Learning*. Jan. 2011. URL: `https://dl.acm.org/doi/10.5555/3104482.3104632` (Cited on pp. 55, 56).

[KT12]   Alex Kulesza and Ben Taskar. "Determinantal Point Processes for Machine Learning". In: *Foundations and Trends in Machine Learning* 5.2–3 (Dec. 2012), pp. 123–286. DOI: `10.1561/2200000044` (Cited on p. 55).

[KT24]   Anastasia Kireeva and Joel A. Tropp. *Randomized Matrix Computations: Themes and Variations*. Feb. 2024. DOI: `10.7907/7yade-5k351` (Cited on pp. 2, 339, 367, 392).

[KV17]   Ravindran Kannan and Santosh Vempala. "Randomized Algorithms in Numerical Linear Algebra". In: *Acta Numerica* 26 (2017), pp. 95–135. DOI: `10.1017/S0962492917000058` (Cited on p. 2).

[KW92]   Jacek Kuczyński and Henryk Woźniakowski. "Estimating the Largest Eigenvalue by the Power and Lanczos Algorithms with a Random Start". In: *SIAM Journal on Matrix Analysis and Applications* 13.4 (1992), pp. 1094–1122. DOI: `10.1137/0613066` (Cited on p. 2).

[KW97]   Rune Karlson and Bertil Waldén. "Estimation of Optimal Backward Perturbation Bounds for the Linear Least Squares Problem". In: *BIT Numerical Mathematics* 37.4 (Dec. 1997), pp. 862–869. DOI: `10.1007/BF02510356` (Cited on p. 342).

[Lan50]   Cornelius Lanczos. "An Iteration Method for the Solution of the Eigenvalue Problem of Linear Differential and Integral Operators". In: *Journal of Research of the National Bureau of Standards* (1950). DOI: `10.6028/jres.045.026` (Cited on pp. 2, 249).

[Lan98]   Bruno Lang. "Using Level 3 BLAS in Rotation-Based Algorithms". In: *SIAM Journal on Scientific Computing* 19.2 (Jan. 1998), pp. 626–634. DOI: `10.1137/S1064827595280211` (Cited on p. 314).

[Läu61]   Peter Läuchli. "Jordan-Elimination und Ausgleichung nach kleinsten Quadraten". In: *Numerische Mathematik* 3.1 (Dec. 1961), pp. 226–240. DOI: `10.1007/BF01386022` (Cited on p. 307).

[LH74]   Charles L. Lawson and Richard J. Hanson. *Solving Least Squares Problems*. Prentice-Hall Series in Automatic Computation. Prentice-Hall, 1974. DOI: `10.1137/1.9781611971217` (Cited on p. 40).

[Lin17]   Lin Lin. "Randomized Estimation of Spectral Densities of Large Matrices Made Accurate". In: *Numerische Mathematik* 136.1 (May 2017), pp. 183–213. DOI: 10.1007/s00211-016-0837-7 (Cited on p. 206).

[Lin23]   Michael Lindsey. *Fast Randomized Entropically Regularized Semidefinite Programming*. Mar. 2023. arXiv: 2303.12133v1 (Cited on pp. 192, 193, 269, 271).

[Liu04]   Jun S. Liu. *Monte Carlo Strategies in Scientific Computing*. Springer Series in Statistics. New York, NY: Springer New York, 2004. DOI: 10.1007/978-0-387-76371-2 (Cited on pp. 112, 113, 195, 206).

[LK22]   Brett W. Larsen and Tamara G. Kolda. *Sketching Matrix Least Squares via Leverage Scores Estimates*. Jan. 2022. arXiv: 2201.10638v1 (Cited on p. 393).

[LLJD+20]   Yong Liu, Shizhong Liao, Shali Jiang, Lizhong Ding, Hailun Lin, and Weiping Wang. "Fast Cross-Validation for Kernel-Based Algorithms". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42.5 (May 2020), pp. 1083–1096. DOI: 10.1109/TPAMI.2019.2892371 (Cited on p. 78).

[LLSS+17]   Huamin Li, George C. Linderman, Arthur Szlam, Kelly P. Stanton, Yuval Kluger, and Mark Tygert. "Algorithm 971: An Implementation of a Randomized Algorithm for Principal Component Analysis". In: *ACM Transactions on Mathematical Software* 43.3 (Jan. 2017). DOI: 10.1145/3004053 (Cited on pp. 11, 24).

[LLYC+09]   Lin Lin, Jianfeng Lu, Lexing Ying, Roberto Car, and Weinan E. "Fast Algorithm for Extracting the Diagonal of the Inverse Matrix with Application to the Electronic Structure Analysis of Metallic Systems". In: *Communications in Mathematical Sciences* 7.3 (Sept. 2009), pp. 755–777. DOI: 10.4310/CMS.2009.V7.N3.A12 (Cited on p. 192).

[LMP13]   Mu Li, Gary L. Miller, and Richard Peng. "Iterative Row Sampling". In: *Fifty-Fourth IEEE Annual Symposium on Foundations of Computer Science*. 2013, pp. 127–136. DOI: 10.1109/FOCS.2013.22 (Cited on p. 269).

[LP23]   Jonathan Lacotte and Mert Pilanci. *Faster Least Squares Optimization*. Apr. 2023. arXiv: 1911.02675v3 (Cited on pp. 330, 332).

[LR20]   Tengyuan Liang and Alexander Rakhlin. "Just Interpolate: Kernel "Ridgeless" Regression Can Generalize". In: *The Annals of Statistics* 48.3 (June 2020). DOI: 10.1214/19-AOS1849 (Cited on p. 87).

[Mac75]   Odile Macchi. "The Coincidence Approach to Stochastic Point Processes". In: *Advances in Applied Probability* 7.1 (Mar. 1975), pp. 83–122. DOI: `10.2307/1425855` (Cited on p. 55).

[Mar11]   P. G. Martinsson. "A Fast Randomized Algorithm for Computing a Hierarchically Semiseparable Representation of a Matrix". In: *SIAM Journal on Matrix Analysis and Applications* 32.4 (Oct. 2011), pp. 1251–1274. DOI: `10.1137/100786617` (Cited on pp. 44, 146).

[MBFJ+11]   C. Morningstar, J. Bulava, J. Foley, K. J. Juge, D. Lenkner, M. Peardon, and C. H. Wong. "Improved Stochastic Estimation of Quark Propagation with Laplacian Heaviside Smearing in Lattice QCD". In: *Physical Review D* 83.11 (June 2011), p. 114505. DOI: `10.1103/PhysRevD.83.114505` (Cited on p. 252).

[MBMD+25]   Maksim Melnichenko, Oleg Balabanov, Riley Murray, James Demmel, Michael W. Mahoney, and Piotr Luszczek. *CholeskyQR with Randomization and Pivoting for Tall Matrices (CQRRPT)*. Mar. 2025. arXiv: `2311.08316v3` (Cited on pp. 149, 365, 385).

[MC23]   Kevin Miller and Jeff Calder. "Poisson Reweighted Laplacian Uncertainty Sampling for Graph-Based Active Learning". In: *SIAM Journal on Mathematics of Data Science* 5.4 (Dec. 2023), pp. 1160–1190. DOI: `10.1137/22M1531981` (Cited on p. 103).

[MCRR20]   Giacomo Meanti, Luigi Carratino, Lorenzo Rosasco, and Alessandro Rudi. "Kernel Methods Through the Roof: Handling Billions of Points Efficiently". In: *Advances in Neural Information Processing Systems*. Vol. 33. 2020, pp. 14410–14422. URL: `https://dl.acm.org/doi/abs/10.5555/3495724.3496932` (Cited on p. 2).

[MD09]   Michael W. Mahoney and Petros Drineas. "CUR Matrix Decompositions for Improved Data Analysis". In: *Proceedings of the National Academy of Sciences* 106.3 (Jan. 2009), pp. 697–702. DOI: `10.1073/pnas.0803205106` (Cited on pp. 44, 162, 164, 176).

[MDME+23]   Riley Murray, James Demmel, Michael W. Mahoney, N. Benjamin Erichson, Maksim Melnichenko, Osman Asif Malik, Laura Grigori, Piotr Luszczek, Michał Dereziński, Miles E. Lopes, Tianyu Liang, Hengrui Luo, and Jack Dongarra. *Randomized Numerical Linear Algebra : A Perspective on the Field With an Eye to Software*. Apr. 2023. arXiv: `2302.11474v2` (Cited on pp. 2, 59, 305, 318, 383, 386, 392).

[Mei24]   Maike Meier. "Randomized Algorithms and Theory for Rank Estimation and Least Squares". PhD thesis. University of Oxford, Dec.

2024. URL: https://ora.ox.ac.uk/objects/uuid:70eb0fd
6-7245-4a36-8b06-c5b8ba4b19c0 (Cited on p. 323).

[Mey21]   Raphael A. Meyer. *Hutch++ for Undergrads*. Feb. 2021. URL:
           https://ram900.com/hutchplusplus/%5C#hutch%5C_for
           %5C_undergrads (Cited on p. 207).

[Mey23]   Raphael A. Meyer. *RandNLA Proof Wiki: Active L_2 Regression via
           Leverage Score Sampling*. 2023. URL: https://randnla.githu
           b.io/leverage-score-regression/ (Cited on p. 393).

[Mey24]   Raphael Arkady Meyer. "Towards Optimal Matrix-Vector Com-
           plexity in Numerical Linear Algebra". PhD thesis. New York
           University Tandom School of Engineering, May 2024 (Cited on
           pp. 208, 242).

[MM15]   Cameron Musco and Christopher Musco. "Randomized Block
           Krylov Methods for Stronger and Faster Approximate Singular
           Value Decomposition". In: *Proceedings of the Twenty-Eigth Inter-
           national Conference on Neural Information Processing Systems*.
           Dec. 2015, pp. 1396–1404. URL: https://dl.acm.org/doi/10
           .5555/2969239.2969395 (Cited on pp. 1, 21, 23).

[MM17]   Cameron Musco and Christopher Musco. "Recursive Sampling
           for the Nyström Method". In: *Proceedings of the Thirtieth Inter-
           national Confernece on Neural Information Processing Systems*.
           2017, pp. 3833–3845. URL: https://dl.acm.org/doi/10.555
           5/3294996.3295140 (Cited on pp. 51, 52).

[MM20]   Cameron Musco and Christopher Musco. *Projection-Cost-
           Preserving Sketches: Proof Strategies and Constructions*. 2020.
           arXiv: 2004.08434v1 (Cited on p. 208).

[MMB21]   Anant Mathur, Sarat Moka, and Zdravko Botev. "Variance Reduc-
           tion for Matrix Computations with Applications to Gaussian Pro-
           cesses". In: ed. by Qianchuan Zhao and Li Xia. Vol. 404. Springer,
           2021, pp. 243–261. DOI: 10.1007/978-3-030-92511-6_16
           (Cited on pp. 191, 193, 269, 271).

[MMMW+23] Raphael A. Meyer, Cameron Musco, Christopher Musco, David P.
           Woodruff, and Samson Zhou. "Near-Linear Sample Complexity for
           L_p Polynomial Regression". In: *Proceedings of the Thirty-Fourth
           Annual ACM-SIAM Symposium on Discrete Algorithms*. Jan. 2023.
           DOI: 10.1137/1.9781611977554 (Cited on p. 269).

[MMMW21a] Raphael A. Meyer, Cameron Musco, Christoper Musco, and David
           P. Woodruff. "Hutch++: Optimal Stochastic Trace Estimation". In:
           Symposium on Simplicity in Algorithms. Jan. 2021. DOI: 10.1137
           /1.9781611976496 (Cited on pp. 204, 206–208).

[MMMW21b]   Raphael A. Meyer, Cameron Musco, Christopher Musco, and David P. Woodruff. *Hutch++: Optimal Stochastic Trace Estimation*. June 2021. arXiv: `2010.09649v5` (Cited on pp. 194, 207, 242).

[MMS18]   Cameron Musco, Christopher Musco, and Aaron Sidford. "Stability of the Lanczos Method for Matrix Function Approximation". In: *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, Jan. 2018, pp. 1605–1624. DOI: `10.1137/1.97816 11975031.105` (Cited on pp. 335, 349, 354).

[MMWY22]   Cameron Musco, Christopher Musco, David P. Woodruff, and Taisuke Yasuda. "Active Linear Regression for ℓ_p Norms and Beyond". In: *Sixty-Third Annual IEEE Symposium on Foundations of Computer Science*. Oct. 2022, pp. 744–753. DOI: `10.1109/FOCS5 4457.2022.00076` (Cited on p. 103).

[MNTW24]   Maike Meier, Yuji Nakatsukasa, Alex Townsend, and Marcus Webb. "Are Sketch-and-Precondition Least Squares Solvers Numerically Stable?" In: *SIAM Journal on Matrix Analysis and Applications* (June 2024), pp. 905–929. DOI: `10.1137/23M1551973` (Cited on pp. 5, 306, 324, 327, 329, 335, 339).

[MP67]   Volodymyr A. Marčenko and Leonid A. Pastur. "Distribution of Eigenvalues for Some Sets of Random Matrices". In: *Mathematics of the USSR-Sbornik* 1.4 (Apr. 1967), p. 457. DOI: `10.1070/SM19 67v001n04ABEH001994` (Cited on p. 370).

[MR95]   Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. 1st edition. Cambridge University Press, Aug. 1995. DOI: `10.1017/CBO9780511814075` (Cited on pp. 53, 377).

[MS06]   Gérard Meurant and Zdeněk Strakoš. "The Lanczos and Conjugate Gradient Algorithms in Finite Precision Arithmetic". In: *Acta Numerica* 15 (May 2006), pp. 471–542. DOI: `10.1017/S096249290 626001X` (Cited on p. 354).

[MSM14]   Xiangrui Meng, Michael A. Saunders, and Michael W. Mahoney. "LSRN: A Parallel Iterative Solver for Strongly Over- or Underdetermined Systems". In: *SIAM Journal on Scientific Computing* 36.2 (Jan. 2014), pp. C95–C118. DOI: `10.1137/120866580` (Cited on pp. 320, 331).

[MT20]   Per-Gunnar Martinsson and Joel A. Tropp. "Randomized Numerical Linear Algebra: Foundations and Algorithms". In: *Acta Numerica* 29 (May 2020), pp. 403–572. DOI: `10.1017/S09624929 20000021` (Cited on pp. 2, 92, 145, 168, 280, 281, 305, 318, 372, 383, 392).

[MW17]   Cameron Musco and David P. Woodruff. "Sublinear Time Low-Rank Approximation of Positive Semidefinite Matrices". In: *Fifty-Eighth Annual IEEE Symposium on Foundations of Computer Science*. Oct. 2017, pp. 672–683. DOI: 10.1109/FOCS.2017.68 (Cited on pp. 29, 160, 184).

[Nak20]   Yuji Nakatsukasa. *Fast and Stable Randomized Low-Rank Matrix Approximation*. Sept. 2020. arXiv: 2009.11392v1 (Cited on pp. 1, 166).

[NN13]   Jelani Nelson and Huy L. Nguyên. "OSNAP: Faster Numerical Linear Algebra Algorithms via Sparser Subspace Embeddings". In: *Fifty-Fourth IEEE Annual Symposium on Foundations of Computer Science*. 2013, pp. 117–126. DOI: 10.1109/FOCS.2013.21 (Cited on p. 383).

[NN14]   Jelani Nelson and Huy L. Nguyên. "Lower Bounds for Oblivious Subspace Embeddings". In: *Automata, Languages, and Programming*. Vol. 8572. 2014, pp. 883–894. DOI: 10.1007/978-3-662-43948-7_73 (Cited on pp. 388–391).

[Now22]   Sebastian Nowozin. *Thoughts on Trace Estimation in Deep Learning*. Aug. 2022. URL: https://www.nowozin.net/sebastian/blog/thoughts-on-trace-estimation-in-deep-learning.html (Cited on p. 191).

[NP23]   Yuji Nakatsukasa and Taejun Park. "Randomized Low-Rank Approximation for Symmetric Indefinite Matrices". In: *SIAM Journal on Matrix Analysis and Applications* 44.3 (Sept. 2023), pp. 1370–1392. DOI: 10.1137/22M1538648 (Cited on p. 23).

[NT24]   Yuji Nakatsukasa and Joel A. Tropp. "Fast and Accurate Randomized Algorithms for Linear Systems and Eigenvalue Problems". In: *SIAM Journal on Matrix Analysis and Applications* 45.2 (June 2024), pp. 1183–1214. DOI: 10.1137/23M1565413 (Cited on p. 383).

[OPA19]   Ibrahim Kurban Ozaslan, Mert Pilanci, and Orhan Arikan. "Iterative Hessian Sketch with Momentum". In: *IEEE International Conference on Acoustics, Speech and Signal Processing*. 2019, pp. 7470–7474. DOI: 10.1109/ICASSP.2019.8682720 (Cited on pp. 330, 332).

[OT10]   Ivan Oseledets and Eugene Tyrtyshnikov. "TT-cross Approximation for Multidimensional Arrays". In: *Linear Algebra and its Applications* 432.1 (2010), pp. 70–88. DOI: 10.1016/j.laa.2009.07.024 (Cited on pp. 44, 146).

[OT18]   Samet Oymak and Joel A Tropp. "Universality Laws for Random-ized Dimension Reduction, with Applications". In: *Information and Inference: A Journal of the IMA* 7.3 (Sept. 2018), pp. 337–446. DOI: 10.1093/imaiai/iax011 (Cited on pp. 372, 373).

[Ove01]   Michael L. Overton. *Numerical Computing with IEEE Floating Point Arithmetic*. Society for Industrial and Applied Mathematics, Jan. 2001. DOI: 10.1137/1.9780898718072 (Cited on p. 308).

[Par98]   Beresford N. Parlett. *The Symmetric Eigenvalue Problem*. Classics in Applied Mathematics. Society for Industrial and Applied Mathematics, Jan. 1998. DOI: 10.1137/1.9781611971163 (Cited on pp. 20, 344).

[PBK25]   David Persson, Nicolas Boullé, and Daniel Kressner. "Randomized Nyström Approximation of Non-negative Self-Adjoint Operators". In: *SIAM Journal on Mathematics of Data Science* (June 2025), pp. 670–698. DOI: 10.1137/24M165082X (Cited on p. 29).

[PCK22]   David Persson, Alice Cortinovis, and Daniel Kressner. "Improved Variants of the Hutch++ Algorithm for Trace Estimation". In: *SIAM Journal on Matrix Analysis and Applications* (Sept. 2022), pp. 1162–1185. DOI: 10.1137/21M1447623 (Cited on pp. 210, 216, 223, 230, 248).

[PCM25]   David Persson, Tyler Chen, and Christopher Musco. *Randomized Block-Krylov Subspace Methods for Low-Rank Approximation of Matrix Functions*. Feb. 2025. arXiv: 2502.01888v1 (Cited on pp. 190, 251).

[Pie24]   Matt Piekenbrock. *primate: Implicit matrix function and trace estimator*. Dec. 2024. URL: https://github.com/peekxc/primate (Cited on p. 248).

[PK23]   David Persson and Daniel Kressner. "Randomized Low-Rank Approximation of Monotone Matrix Functions". In: *SIAM Journal on Matrix Analysis and Applications* 44.2 (June 2023), pp. 894–918. DOI: 10.1137/22M1523923 (Cited on p. 295).

[PMM25]   David Persson, Raphael A. Meyer, and Christopher Musco. "Algorithm-Agnostic Low-Rank Approximation of Operator Monotone Matrix Functions". In: *SIAM Journal on Matrix Analysis and Applications* 46.1 (Mar. 2025), pp. 1–21. DOI: 10.1137/23M1619435 (Cited on p. 295).

[PN25]   Taejun Park and Yuji Nakatsukasa. "Accuracy and Stability of CUR Decompositions with Oversampling". In: *SIAM Journal on Matrix Analysis and Applications* 46.1 (Mar. 2025), pp. 780–810. DOI: 10.1137/24M1660346 (Cited on pp. 162, 164, 166, 171, 175, 301).

[Pop23]   Niclas Joshua Popp. "Randomized Diagonal Estimation". MA thesis. KTH Royal Insitute of Technology, 2023. URL: https://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-339542 (Cited on pp. 187, 192).

[Pou20]   Jack Poulson. "High-Performance Sampling of Generic Determinantal Point Processes". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 378.2166 (Mar. 2020), p. 20190059. DOI: 10.1098/rsta.2019.0059 (Cited on pp. 72, 161).

[PS05]   Simo Puntanen and George P. H. Styan. "Historical Introduction: Issai Schur and the Early Development of the Schur Complement". In: *The Schur Complement and Its Applications*. Ed. by Fuzhen Zhang. Numerical Methods and Algorithms. Springer, 2005, pp. 1–16. DOI: 10.1007/0-387-24273-2_1 (Cited on p. 224).

[PS82]   Christopher C. Paige and Michael A. Saunders. "LSQR: An Algorithm for Sparse Linear Equations and Sparse Least Squares". In: *ACM Transactions on Mathematical Software* 8.1 (Mar. 1982), pp. 43–71. DOI: 10.1145/355984.355989 (Cited on p. 314).

[PS98]   George Pólya and Gabor Szegő. *Problems and Theorems in Analysis I*. Springer, 1998. DOI: 10.1007/978-3-642-61983-0 (Cited on p. 212).

[PTRV98]   Christos H. Papadimitriou, Hisao Tamaki, Prabhakar Raghavan, and Santosh Vempala. "Latent Semantic Indexing: A Probabilistic Analysis". In: *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. May 1998, pp. 159–168. DOI: 10.1145/275487.275505 (Cited on pp. 1, 157, 316).

[Puk06]   Friedrich Pukelsheim. *Optimal Design of Experiments*. Society for Industrial and Applied Mathematics, Jan. 2006. DOI: 10.1137/1.9780898719109 (Cited on p. 44).

[PW16]   Mert Pilanci and Martin J. Wainwright. "Iterative Hessian Sketch: Fast and Accurate Solution Approximation for Constrained Least-Squares". In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 1842–1879. URL: https://dl.acm.org/doi/10.5555/2946645.2946698 (Cited on pp. 329, 330, 332).

[PZBC+07] Peristera Paschou, Elad Ziv, Esteban G. Burchard, Shweta Choudhry, William Rodriguez-Cintron, Michael W. Mahoney, and Petros Drineas. "PCA-correlated SNPs for Structure Identification in Worldwide Human Populations". In: *PLoS genetics* 3.9 (2007), e160. DOI: 10.1371/journal.pgen.0030160 (Cited on p. 44).

[Que49]   Maurice H. Quenouille. "Approximate Tests of Correlation in Time-Series". In: *Journal of the Royal Statistical Society. Series B (Methodological)* 11.1 (1949), pp. 68–84. JSTOR: 2983696 (Cited on p. 292).

[RCCR18]   Alessandro Rudi, Daniele Calandriello, Luigi Carratino, and Lorenzo Rosasco. "On Fast Leverage Score Sampling and Optimal Learning". In: *Proceedings of the Thirty-Second International Conference on Neural Information Processing Systems*. Dec. 2018, pp. 5677–5687. URL: https://dl.acm.org/doi/10.5555/3327345.3327470 (Cited on p. 52).

[RCR15]   Alessandro Rudi, Raffaello Camoriano, and Lorenzo Rosasco. "Less Is More: Nyström Computational Regularization". In: *Advances in Neural Information Processing Systems*. Vol. 28. 2015. URL: https://dl.acm.org/doi/10.5555/2969239.2969424 (Cited on p. 99).

[RCV14]   Juri Ranieri, Amina Chebira, and Martin Vetterli. "Near-Optimal Sensor Placement for Linear Inverse Problems". In: *IEEE Transactions on Signal Processing* 62.5 (2014), pp. 1135–1146. DOI: 10.1109/TSP.2014.2299518 (Cited on p. 44).

[RDRv14]   Raghunathan Ramakrishnan, Pavlo O. Dral, Matthias Rupp, and O. Anatole von Lilienfeld. "Quantum Chemistry Structures and Properties of 134 Kilo Molecules". In: *Scientific Data* 1.1 (Dec. 2014), p. 140022. DOI: 10.1038/sdata.2014.22 (Cited on p. 97).

[RO19]   Alireza Rezaei and Shayan Oveis Gharan. "A Polynomial Time MCMC Method for Sampling from Continuous Determinantal Point Processes". In: *Proceedings of the Thirty-Sixth International Conference on Machine Learning*. May 2019, pp. 5438–5447. URL: https://proceedings.mlr.press/v97/rezaei19a.html (Cited on pp. 56, 71).

[RST10]   Vladimir Rokhlin, Arthur Szlam, and Mark Tygert. "A Randomized Algorithm for Principal Component Analysis". In: *SIAM Journal on Matrix Analysis and Applications* 31.3 (Jan. 2010), pp. 1100–1124. DOI: 10.1137/080736417 (Cited on pp. 19, 21, 23).

[RT08]   Vladimir Rokhlin and Mark Tygert. "A Fast Randomized Algorithm for Overdetermined Linear Least-Squares Regression". In: *Proceedings of the National Academy of Sciences* 105.36 (Sept. 2008), pp. 13212–13217. DOI: 10.1073/pnas.0804869105 (Cited on pp. 1, 4, 320, 377).

[RV13] Mark Rudelson and Roman Vershynin. "Hanson-Wright Inequality and Sub-Gaussian Concentration". In: *Electronic Communications in Probability* 18.none (Jan. 2013), pp. 1–9. DOI: `10.1214/ECP.v18-2865` (Cited on p. 208).

[RvBR12] Lars Ruddigkeit, Ruud van Deursen, Lorenz C. Blum, and Jean-Louis Reymond. "Enumeration of 166 Billion Organic Small Molecules in the Chemical Universe Database GDB-17". In: *Journal of Chemical Information and Modeling* 52.11 (Nov. 2012), pp. 2864–2875. DOI: `10.1021/ci300415d` (Cited on p. 97).

[RW05] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning Series. MIT Press, 2005. DOI: `10.7551/mitpress/3206.001.0001` (Cited on pp. 2, 73).

[Saa03] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. 2nd ed. Vol. 82. Society for Industrial and Applied Mathematics, 2003. DOI: `10.1137/1.9780898718003` (Cited on pp. 93, 188, 315).

[Saa11] Yousef Saad. *Numerical Methods for Large Eigenvalue Problems*. Classics in Applied Mathematics. Society for Industrial and Applied Mathematics, Jan. 2011. DOI: `10.1137/1.9781611970739` (Cited on p. 188).

[SAI17] Arvind K. Saibaba, Alen Alexanderian, and Ilse C. F. Ipsen. "Randomized Matrix-Free Trace and Log-Determinant Estimators". In: *Numerische Mathematik* 137.2 (Oct. 2017), pp. 353–395. DOI: `10.1007/s00211-017-0880-z` (Cited on p. 203).

[Sar06] Tamás Sarlós. "Improved Approximation Algorithms for Large Matrices via Random Projections". In: *Forty-Seventh Annual IEEE Symposium on Foundations of Computer Science*. Oct. 2006, pp. 143–152. DOI: `10.1109/FOCS.2006.37` (Cited on pp. 316, 317, 319).

[SB00] Alex J. Smola and Peter L. Bartlett. "Sparse Greedy Gaussian Process Regression". In: *Advances in Neural Information Processing Systems*. Vol. 13. 2000. URL: `https://dl.acm.org/doi/10.5555/3008751.3008838` (Cited on pp. 99, 101).

[Sch07] R Schaback. *Kernel-Based Meshless Methods*. Available online at `http://num.math.uni-goettingen.de/~schaback/teaching/Appverf_II.pdf`. July 2007 (Cited on p. 77).

[SDM22] Abhishek Shetty, Raaz Dwivedi, and Lester Mackey. "Distribution Compression in Near-Linear Time". In: *International Conference on Learning Representations*. Avilable at `https://arxiv.org/abs/2111.07941v6`. Apr. 2022 (Cited on pp. 123, 124).

[SE16]   D. C. Sorensen and Mark Embree. "A DEIM Induced CUR Factorization". In: *SIAM Journal on Scientific Computing* 38.3 (Jan. 2016), A1454–A1482. DOI: 10.1137/140978430 (Cited on pp. 164, 175).

[SL22]   Aleksandros Sobczyk and Mathieu Luisier. "Approximate Euclidean Lengths and Distances beyond Johnson-Lindenstrauss". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 19357–19369. URL: https://dl.acm.org/doi/10.5555/3600270.3601677 (Cited on p. 271).

[SM22]   Gilbert Strang and Cleve Moler. "LU and CR Elimination". In: *SIAM Review* 64.1 (Feb. 2022), pp. 181–190. DOI: 10.1137/20M1358694 (Cited on p. 142).

[Sob24]   Aleksandros Sobczyk. "Algorithms for Hermitian Eigenproblems and Applications in Quantum Chemistry and Machine Learning". PhD thesis. ETH Zurich, 2024. DOI: 10.3929/ethz-b-000734741 (Cited on pp. 193, 271).

[SS02]   B. Schölkopf and A.J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Adaptive Computation and Machine Learning. MIT Press, 2002. DOI: 10.7551/mitpress/4175.001.0001 (Cited on pp. 2, 73, 79).

[SS08]   Daniel A. Spielman and Nikhil Srivastava. "Graph Sparsification by Effective Resistances". In: *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*. Victoria British Columbia Canada: ACM, May 2008, pp. 563–568. DOI: 10.1145/1374376.1374456 (Cited on pp. 193, 269).

[SS11]   Stefan A. Sauter and Christoph Schwab. *Boundary Element Methods*. Vol. 39. Springer Series in Computational Mathematics. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. DOI: 10.1007/978-3-540-68093-2 (Cited on p. 32).

[SS12]   Ingo Steinwart and Clint Scovel. "Mercer's Theorem on General Domains: On the Interaction between Measures, Kernels, and RKHSs". In: *Constructive Approximation* 35.3 (June 2012), pp. 363–417. DOI: 10.1007/s00365-012-9153-3 (Cited on pp. 108, 109).

[SS90]   G. W. Stewart and Ji-guang Sun. *Matrix Perturbation Theory*. 1st Edition. Computer Science and Scientific Computing. Academic Press, 1990 (Cited on p. 13).

[SSKK+21] Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. "Score-Based Generative Modeling through Stochastic Differential Equations". In: *Ninth International Conference on Learning Representations*. Available

at `https://arxiv.org/abs/2011.13456v2`. 2021 (Cited on p. 191).

[ST01]   Daniel Spielman and Shang-Hua Teng. "Smoothed Analysis of Algorithms: Why the Simplex Algorithm Usually Takes Polynomial Time". In: *Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing*. Hersonissos Greece: ACM, July 2001, pp. 296–305. DOI: `10.1145/380752.380813` (Cited on p. 339).

[Ste24]   Stefan Steinerberger. *Randomly Pivoted Partial Cholesky: Random How?* Apr. 2024. arXiv: `2404.11487v1` (Cited on pp. 68, 69).

[Ste86]   J. Michael Steele. "An Efron-Stein Inequality for Nonsymmetric Statistics". In: *Annals of Statistics* 14.2 (June 1986), pp. 753–758. DOI: `10.1214/aos/1176349952` (Cited on p. 290).

[Ste98]   G. W. Stewart. *Afternotes Goes to Graduate School*. Society for Industrial and Applied Mathematics, Jan. 1998. DOI: `10.1137/1.9781611971422` (Cited on p. 81).

[Ste99]   Gilbert W. Stewart. "Four Algorithms for the the Efficient Computation of Truncated Pivoted QR Approximations to a Sparse Matrix". In: *Numerische Mathematik* 83 (1999), pp. 313–323. DOI: `10.1007/s002110050451` (Cited on p. 163).

[TB22]   Lloyd N. Trefethen and David Bau. *Numerical Linear Algebra*. Vol. 181. Society for Industrial and Applied Mathematics, 2022. DOI: `10.1137/1.9781611977165` (Cited on p. 187).

[Tit09]   Michalis Titsias. "Variational Learning of Inducing Variables in Sparse Gaussian Processes". In: *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics*. Apr. 2009, pp. 567–574. URL: `https://proceedings.mlr.press/v5/titsias09a.html` (Cited on p. 101).

[Tre05]   Nick Trefethen. *Ten Digit Algorithms*. Tech. rep. 05/13. University of Oxford, 2005. URL: `https://people.maths.ox.ac.uk/trefethen/papers.html` (Cited on p. 7).

[Tre22]   Lloyd N. Trefethen. *An Applied Mathematician's Apology*. Society for Industrial and Applied Mathematics, Jan. 2022. DOI: `10.1137/1.9781611977196` (Cited on p. 327).

[Tro11]   Joel A. Tropp. "Improved Analysis of the Subsampled Randomized Hadamard Transform". In: *Advances in Adaptive Data Analysis* 03.01n02 (Apr. 2011), pp. 115–126. DOI: `10.1142/S1793536911000787` (Cited on pp. 375, 377).

[Tro20]   Joel A Tropp. *Randomized Algorithms for Matrix Computations*. 2020. DOI: `10.7907/nwsv-df59` (Cited on pp. 201, 250).

[Tro21]   Joel A. Tropp. *Probability in High Dimensions*. 2021. DOI: `10.79 07/mxr0-c422` (Cited on pp. 233, 291, 372).

[Tro22]   Joel A. Tropp. *Matrix Analysis*. 2022 (Cited on p. 65).

[Tro23]   Joel A. Tropp. *Probability Theory & Computational Mathematics*. Dec. 2023. DOI: `10.7907/Q75SZ-E1E79` (Cited on pp. 46, 86).

[Tro25]   Joel A. Tropp. *Comparison Theorems for the Minimum Eigenvalue of a Random Positive-Semidefinite Matrix*. Jan. 2025. arXiv: `2501 .16578v1` (Cited on pp. 319, 367, 372, 380, 387, 388).

[TS12]   Jok M. Tang and Yousef Saad. "A Probing Method for Computing the Diagonal of a Matrix Inverse". In: *Numerical Linear Algebra with Applications* 19.3 (2012), pp. 485–501. DOI: `10.1002/nla .779` (Cited on pp. 252, 253).

[TSL24]   Joseph Tindall, Miles Stoudenmire, and Ryan Levy. *Compressing Multivariate Functions with Tree Tensor Networks*. Oct. 2024. arXiv: `2410.03572v1` (Cited on pp. 44, 146).

[TT15]   Alex Townsend and Lloyd N. Trefethen. "Continuous Analogues of Matrix Factorizations". In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 471.2173 (Jan. 2015), p. 20140585. DOI: `10.1098/rspa.2014.0585` (Cited on p. 81).

[Tuk58]   John Tukey. "Bias and Confidence in Not Quite Large Samples". In: *Annals of Mathematical Statistics* 29 (1958), p. 614 (Cited on p. 290).

[Tur48]   Alan M. Turing. "Rounding-off Errors in Matrix Processes". In: *The Quarterly Journal of Mechanics and Applied Mathematics* 1.1 (1948), pp. 287–308. DOI: `10.1093/qjmam/1.1.287` (Cited on p. 307).

[TW23]   Joel A. Tropp and Robert J. Webber. *Randomized Algorithms for Low-Rank Matrix Approximation: Design, Analysis, and Applications*. Sept. 2023. arXiv: `2306.12418v3` (Cited on pp. 1, 2, 17, 19–21, 23, 26, 29, 208).

[TYUC17a]   Joel A. Tropp, Alp Yurtsever, Madeleine Udell, and Volkan Cevher. "Fixed-Rank Approximation of a Positive-Semidefinite Matrix from Streaming Data". In: *Advances in Neural Information Processing Systems*. Vol. 30. 2017, pp. 1225–1234. URL: `https://dl .acm.org/doi/10.5555/3294771.3294888` (Cited on pp. 24, 29, 377).

[TYUC17b]   Joel A. Tropp, Alp Yurtsever, Madeleine Udell, and Volkan Cevher. "Practical Sketching Algorithms for Low-Rank Matrix Approximation". In: *SIAM Journal on Matrix Analysis and Applications* 38.4 (Jan. 2017), pp. 1454–1485. DOI: `10.1137/17M1111590` (Cited on p. 166).

[TYUC19]   Joel A. Tropp, Alp Yurtsever, Madeleine Udell, and Volkan Cevher. "Streaming Low-Rank Matrix Approximation with an Application to Scientific Simulation". In: *SIAM Journal on Scientific Computing* 41.4 (Jan. 2019), A2430–A2463. DOI: `10.1137/18M1201068` (Cited on p. 385).

[US17]   Shashanka Ubaru and Yousef Saad. "Applications of Trace Estimation Techniques". In: *High Performance Computing in Science and Engineering*. Lecture Notes in Computer Science. Springer, 2017, pp. 19–33. DOI: `10.1007/978-3-319-97136-0_2` (Cited on pp. 187, 191).

[UT19]   Madeleine Udell and Alex Townsend. "Why Are Big Data Matrices Approximately Low Rank?" In: *SIAM Journal on Mathematics of Data Science* 1.1 (Jan. 2019), pp. 144–160. DOI: `10.1137/18M1183480` (Cited on p. 12).

[van14]   Ramon van Handel. *Probability in High Dimension*. June 2014. DOI: `10.21236/ADA623999` (Cited on pp. 291, 372).

[Van19]   Axel Vanraes. *Python Implementation of the RLS-Nyström Method*. 2019. URL: `https://github.com/axelv/recursive-nystrom` (Cited on p. 52).

[van69]   Abraham van der Sluis. "Condition Numbers and Equilibration of Matrices". In: *Numerische Mathematik* 14.1 (1969), pp. 14–23. DOI: `10.1007/BF02165096` (Cited on p. 314).

[van74]   A. van der Sluis. "Stability of the Solutions of Linear Least Squares Problems". In: *Numerische Mathematik* 23.3 (June 1974), pp. 241–254. DOI: `10.1007/BF01400307` (Cited on p. 307).

[Van92]   Charles Van Loan. *Computational Frameworks for the Fast Fourier Transform*. Society for Industrial and Applied Mathematics, 1992. DOI: `10.1137/1.9781611970999` (Cited on p. 375).

[Ver18]   Roman Vershynin. *High-Dimensional Probability: An Introduction with Applications in Data Science*. Vol. 47. Cambridge University Press, 2018. DOI: `10.1017/9781108231596` (Cited on p. 372).

[vG47]   John von Neumann and Herman Heine Goldstine. "Numerical Inverting of Matrices of High Order". In: *Bulletin of the Americal Mathematical Society* (1947). DOI: `10.1090/S0002-9904-1947-08909-6` (Cited on p. 307).

[VM17]   Sergey Voronin and Per-Gunnar Martinsson. "Efficient Algorithms for CUR and Interpolative Matrix Decompositions". In: *Advances in Computational Mathematics* 43.3 (June 2017), pp. 495–516. DOI: `10.1007/s10444-016-9494-8` (Cited on p. 152).

[von51]   John von Neumann. "13. Various Techniques Used in Connection with Random Digits". In: *Applied Mathe Series* 12.36-38 (1951), p. 5. URL: `https://mcnp.lanl.gov/pdf_files/InBook_Computing_1961_Neumann_JohnVonNeumannCollectedWorks_VariousTechniquesUsedinConnectionwithRandomDigits.pdf` (Cited on pp. 112, 113).

[Wah90]   Grace Wahba. *Spline Models for Observational Data*. Society for Industrial and Applied Mathematics, 1990. DOI: `10.1137/1.9781611970128` (Cited on p. 99).

[WCV12]   Zhuang Wang, Koby Crammer, and Slobodan Vucetic. "Breaking the Curse of Kernelization: Budgeted Stochastic Gradient Descent for Large-Scale SVM Training". In: *Journal of Machine Learning Research* 13.100 (2012), pp. 3103–3131. URL: `https://jmlr.org/papers/v13/wang12b.html` (Cited on p. 92).

[WDT22]   Heather Wilber, Anil Damle, and Alex Townsend. "Data-Driven Algorithms for Signal Processing with Trigonometric Rational Functions". In: *SIAM Journal on Scientific Computing* 44.3 (June 2022), pp. C185–C209. DOI: `10.1137/21M1420277` (Cited on p. 44).

[WEB25]   Heather Wilber, Ethan N. Epperly, and Alex H. Barnett. "Superfast Direct Inversion of the Nonuniform Discrete Fourier Transform via Hierarchically Semiseparable Least Squares". In: *SIAM Journal on Scientific Computing* (June 2025), A1702–A1732. DOI: `10.1137/24M1656694` (Cited on pp. 7, 178).

[Wed34]   J. Wedderburn. *Lectures on Matrices*. Vol. 17. Colloquium Publications. Providence, Rhode Island: American Mathematical Society, Dec. 1934. DOI: `10.1090/coll/017` (Cited on p. 166).

[Wed73]   Per-Åke Wedin. "Perturbation Theory for Pseudo-Inverses". In: *BIT Numerical Mathematics* 13.2 (June 1973), pp. 217–232. DOI: `10.1007/BF01933494` (Cited on p. 311).

[Wen04]   Holger Wendland. *Scattered Data Approximation*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2004. DOI: `10.1017/CBO9780511617539` (Cited on p. 99).

[Wil21]   Heather Denise Wilber. "Computing Numerically With Rational Functions". PhD thesis. Cornell University, 2021. URL: `https:`

`//heatherw3521.github.io/phd_thesis.pdf` (Cited on pp. 32, 44, 146).

[Wil55]   James Hardy Wilkinson. "The Use of Iterative Methods for Finding the Latent Roots and Vectors of Matrices". In: *Mathematical Tables and Other Aids to Computation* 9.52 (1955), pp. 184–191. DOI: `10.2307/2002055` (Cited on p. 125).

[Wil85]   J. H. Wilkinson. "The State of the Art in Error Analysis". In: *NAG Newsletter* 2/85 (1985). Invited lecture for the NAG 1984 Annual General Meeting, pp. 5–28 (Cited on p. 344).

[WKS95]   Bertil Waldén, Rune Karlson, and Ji-Guang Sun. "Optimal Backward Perturbation Bounds for the Linear Least Squares Problem". In: *Numerical Linear Algebra with Applications* 2.3 (May 1995), pp. 271–286. DOI: `10.1002/nla.1680020308` (Cited on p. 342).

[WLRT08]   Franco Woolfe, Edo Liberty, Vladimir Rokhlin, and Mark Tygert. "A Fast Randomized Algorithm for the Approximation of Matrices". In: *Applied and Computational Harmonic Analysis* 25.3 (Nov. 2008), pp. 335–366. DOI: `10.1016/j.acha.2007.12.002` (Cited on pp. 166, 374, 375).

[WMG19]   Romain Warlop, Jérémie Mary, and Mike Gartrell. "Tensorized Determinantal Point Processes for Recommendation". In: *Proceedings of the Twenty-Fifth ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. July 2019, pp. 1605–1615. DOI: `10.1145/3292500.3330952` (Cited on p. 44).

[Woo14]   David P. Woodruff. "Sketching as a Tool for Numerical Linear Algebra". In: *Foundations and Trends in Theoretical Computer Science* 10.1–2 (Oct. 2014), pp. 1–157. DOI: `10.1561/0400000060` (Cited on pp. 2, 284, 305, 369, 375, 385).

[WPHC+22]   Jonathan Wenger, Geoff Pleiss, Philipp Hennig, John Cunningham, and Jacob Gardner. "Preconditioning for Scalable Gaussian Process Hyperparameter Optimization". In: *Proceedings of the Thirty-Ninth International Conference on Machine Learning*. June 2022, pp. 23751–23780. URL: `https://proceedings.mlr.press/v162/wenger22a.html` (Cited on p. 191).

[WS00]   Christopher K. I. Williams and Matthias Seeger. "Using the Nyström Method to Speed up Kernel Machines". In: *Proceedings of the Thirteenth International Conference on Neural Information Processing Systems*. NIPS'00. Cambridge, MA, USA: MIT Press, Jan. 2000, pp. 661–667 (Cited on p. 47).

[WW76]   Denis Weaire and A. R. Williams. "New Numerical Approach to the Anderson Localization Problem". In: *Journal of Physics C:*

Solid State Physics 9.17 (1976), p. L461. DOI: `10.1088/0022-3719/9/17/004` (Cited on p. 194).

[WW77] Denis Weaire and A. R. Williams. "The Anderson Localization Problem. I. A New Numerical Approach". In: *Journal of Physics C: Solid State Physics* 10.8 (1977), p. 1239. DOI: `10.1088/0022-3719/10/8/025` (Cited on p. 194).

[WWZ14] Karl Wimmer, Yi Wu, and Peng Zhang. "Optimal Query Complexity for Estimating the Trace of a Matrix". In: *Automata, Languages, and Programming*. Vol. 8572. Springer, 2014, pp. 1051–1062. DOI: `10.1007/978-3-662-43948-7_87` (Cited on p. 237).

[WZ13] Shusen Wang and Zhihua Zhang. "Improving CUR Matrix Decomposition and the Nyström Approximation via Adaptive Sampling". In: *The Journal of Machine Learning Research* 14.1 (2013), pp. 2729–2769. DOI: `https://dl.acm.org/doi/10.5555/2567709.2567748` (Cited on p. 159).

[XXCB14] Yuanzhe Xi, Jianlin Xia, Stephen Cauley, and Venkataramanan Balakrishnan. "Superfast and Stable Structured Solvers for Toeplitz Least Squares via Randomized Sampling". In: *SIAM Journal on Matrix Analysis and Applications* 35.1 (2014), pp. 44–72. DOI: `10.1137/120895755` (Cited on p. 178).

[XXG12] Jianlin Xia, Yuanzhe Xi, and Ming Gu. "A Superfast Structured Solver for Toeplitz Linear Systems via Randomized Sampling". In: *SIAM Journal on Matrix Analysis and Applications* 33.3 (Jan. 2012), pp. 837–858. DOI: `10.1137/110831982` (Cited on p. 178).

[YGSM+21] Zhewei Yao, Amir Gholami, Sheng Shen, Mustafa Mustafa, Kurt Keutzer, and Michael Mahoney. "ADAHESSIAN: An Adaptive Second Order Optimizer for Machine Learning". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 35.12 (May 2021), pp. 10665–10673. DOI: `10.1609/aaai.v35i12.17275` (Cited on p. 192).

[YXY20] Xin Ye, Jianlin Xia, and Lexing Ying. "Analytical Low-Rank Compression via Proxy Point Selection". In: *SIAM Journal on Matrix Analysis and Applications* 41.3 (Jan. 2020), pp. 1059–1085. DOI: `10.1137/19M1247838` (Cited on p. 155).