Randomized Numerical Linear Algebra

A Perspective on the Field With an Eye to Software

April 12, 2023

Preface

Randomized numerical linear algebra – RandNLA, for short – concerns the use of randomization as a resource to develop improved algorithms for large-scale linear algebra computations. The origins of contemporary RandNLA lay in theoretical computer science, where it blossomed from a simple idea: randomization provides an avenue for computing *approximate* solutions to linear algebra problems more efficiently than deterministic algorithms. This idea proved fruitful in and was largely driven by the development of scalable algorithms for machine learning and statistical data analysis applications. However, the true potential of RandNLA only came into focus once it began to integrate with the fields of numerical analysis and "classical" numerical linear algebra. Through the efforts of many individuals, randomized algorithms have been developed that provide full control over the accuracy of their solutions and that can be every bit as reliable as algorithms that might be found in libraries such as LAPACK.

The spectrum of possibilities offered by RandNLA has created a virtuous cycle of contributions by numerical analysts, statisticians, theoretical computer scientists, and the machine learning community. Recent years have even seen the incorporation of certain RandNLA methods into MATLAB, the NAG Library, NVIDIA's cuSOLVER, and SciKit-Learn. In view of these developments, we believe the time is right to accelerate the adoption of RandNLA in the scientific community. In particular, we believe the community stands to benefit significantly from a suitably defined "RandBLAS" and "RandLAPACK," to serve as standard libraries for RandNLA, in much the same way that BLAS and LAPACK serve as standards for deterministic linear algebra.

This monograph surveys the field of RandNLA as a step toward building meaningful RandBLAS and RandLAPACK libraries. Section 1 primes the reader for a dive into the field and summarizes this monograph's content at multiple levels of detail. Section 2 focuses on RandBLAS, which is to be responsible for *sketching*. Details of functionality suitable for RandLAPACK are covered in the five sections that follow. Specifically, Sections 3 to 5 cover least squares and optimization, low-rank approximation, and other select problems that are well-understood in how they benefit from randomized algorithms. The remaining sections – on statistical leverage scores (Section 6) and tensor computations (Section 7) – read more like traditional surveys. The different flavor of these latter sections reflects how, in our assessment, the literature on these topics is still maturing.

We provide a substantial amount of pseudo-code and supplementary material over the course of five appendices. Much of the pseudo-code has been tested via publicly available Matlab and Python implementations.

Authors

Riley Murray, ICSI, LBNL, and University of California, Berkeley
rjmurray@berkeley.edu

James Demmel, University of California, Berkeley
demmel@berkeley.edu

Michael W. Mahoney, ICSI, LBNL, and University of California, Berkeley
mmahoney@stat.berkeley.edu

N. Benjamin Erichson, ICSI and Lawrence Berkeley National Laboratory
erichson@icsi.berkeley.edu

Maksim Melnichenko, University of Tennessee, Knoxville
mmelnic1@vols.utk.edu

Osman Asif Malik, Lawrence Berkeley National Laboratory
oamalik@lbl.gov

Laura Grigori, INRIA Paris and J.L. Lions Laboratory, Sorbonne University
laura.grigori@inria.fr

Piotr Luszczek, University of Tennessee, Knoxville
luszczek@icl.utk.edu

Michał Dereziński, University of Michigan
derezin@umich.edu

Miles E. Lopes, University of California, Davis
melopes@ucdavis.edu

Tianyu Liang, University of California, Berkeley
tianyul@berkeley.edu

Hengrui Luo, Lawrence Berkeley National Laboratory
hrluo@lbl.gov

Jack Dongarra, University of Tennessee, Knoxville
dongarra@icl.utk.edu

Acknowledgements

Many individuals from the community gave detailed feedback on earlier versions of this monograph that were not circulated publicly. These individuals include Mark Tygert, Cameron Musco, Joel Tropp, Per-Gunnar Martinsson, Alex Townsend, Daniel Kressner, Alice Cortinovis, Ilse Ipsen, Sergey Voronin, Vivak Patel, Daniel Maldonado, Tammy Kolda, Florian Schaefer, Ramki Kannan, and Piyush Sao – each of them has our sincere gratitude for their assistance.

In addition, we thank the following people for providing input on the earliest stages of this project: Vivek Bharadwaj, Younghyun Cho, Jelani Nelson, Mark Gates, Weslley da Silva Pereira, Julie Langou, and Julien Langou.

This work was partially funded by an NSF Collaborative Research Framework: Basic ALgebra LIbraries for Sustainable Technology with Interdisciplinary Collaboration (BALLISTIC), a project of the International Computer Science Institute, the University of Tennessee's ICL, the University of California at Berkeley, and the University of Colorado at Denver (NSF Grant Nos. 2004235, 2004541, 2004763, 2004850, respectively) [DDL+20]. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. MWM would also like to thank the Office of Naval Research, which provided partial funding via a Basic Research Challenge on Randomized Numerical Linear Algebra.

Release History

- 11/13/2022. The first publicly-available draft of this monograph was circulated as a technical report at SC22.
- 02/22/2023: arXiv V1. Improvements were made to Section 3.2.2 based on comments from Joel Tropp. Helpful feedback from Robert Webber led to improvements throughout Sections 4 and 5.2. Clarifications and greater detail were added to Appendix B.2 following comments from Ilse Ipsen.
- 04/12/2023: arXiv V2. Section 5.1 was slightly revised based on valuable comments from Oleg Balabanov. Section 5.3 was rewritten and expanded following helpful discussions with Tyler Chen. Sections 4.1.1 and 4.5 now mention an important piece of software that Mark Tygert brought to our attention. Revisions were made to Section 5.2.2 to more clearly and accurately characterize methods from the literature.

Contents

Section 1

Introduction

This introductory section has three principal goals: to motivate our subject and clarify common misconceptions that surround it (§1.1); to explain this monograph's scope and overarching structure (§1.2); and to help direct the reader's attention through section-by-section summaries (§1.3). Many readers may benefit from our "survey of surveys" (§1.4), and all should at least briefly consult the section on notation and definitions (§1.5).

1.1 Our world

Numerical linear algebra (NLA) concerns algorithms for computations on matrices with numerical entries. Originally driven by applications in the physical sciences, it now provides the foundation for vast swaths of applied and computational mathematics. The cultural norms in this field developed many years ago, in large part from recurring themes in problem formulations and algorithmically-useful structures in matrices. However, more recently, NLA has also been motivated by developments in machine learning and data science. Applications in these fields also have their own themes of problem formulations and structures in data, often of a very different nature than those in more classical applications.

A dire situation. While communities that rely on NLA now vary widely, they share one essential property: a ravenous appetite for solving larger and larger problems. For decades, this hunger was satiated by complementary innovations in hardware and software. However, this progress should not be taken for granted. In particular, there are two factors that increasingly present obstacles to scaling linear algebra computations to the next level.

- *Space and power constraints in hardware.* Chips today have billions of transistors, and these transistors are packed into a very small amount of space. It takes power to run these transistors at gigahertz frequencies, and power generates heat. It is hard for one hot thing to dissipate heat when surrounded by millions of other hot things. Too much heat can fry a chip.

  These constraints are known to industry and research community alike, and often referred to as the breakdown of the Dennard's Law [DGR+74; Boh07] and the sunsetting of Moore's Law [Moo65]. The former represents the infamous *power wall* due to the inability of dissipating the heat produced by the processors leading to flattened curve of clock frequency increases. The latter introduced the post-Moore era of heterogeneous computing [VBG+18] leading to plethora specialized hardware targeting individual application spaces.

  The end result of all this? A situation where "more powerful processors" are just scaled-out versions of "less powerful processors," for both commodity and server-tier hardware. Any algorithm that does not parallelize well is fundamentally limited in its ability to leverage these advances. If one's pockets are deep enough, then one can try to get around this with purpose-built accelerators. But even then, there remains the matter of programming those accelerators, and high-performance implementations of classical NLA algorithms are anything but simple.

- *NLA's maturity as a field.* Software can only improve so much without algorithmic innovations. At the same time, linear algebra is a very well-studied topic, and most algorithmic breakthroughs in recent years have required carefully exploiting structures present in specific problems. Identifying new and useful problem structures has been increasingly difficult, often requiring deep knowledge of NLA alongside substantial domain expertise.

If we are to continue scaling our capabilities in matrix computations, then it is essential that we leverage all technologies that are on the table.

An underutilized technology. This monograph concerns *randomized numerical linear algebra*, or *RandNLA*, for short. Algorithms in this realm offer compelling advantages in a wide variety of settings. Some provide an unrivaled combination of efficiency and reliability in computing approximate solutions for massive problems. Others provide fine-grained control when balancing accuracy and computational cost, as is essential for practitioners who are operating at the limits of what their machines can handle. In many cases, the practicality of these algorithms can be seen even with elementary MATLAB or Python implementations, which increases their suitability for adapting to new hardware by leveraging similarly powerful abstraction layers. Finally, although truly high-performance implementations are more complicated, they remain relatively easy to implement when given the right building blocks.

But we are getting ahead of ourselves. What do we mean by "randomized algorithms," as the term is used within RandNLA? First and foremost, these are algorithms that are probabilistic in nature. They use randomness as part of their internal logic to make decisions or compute estimates, which they can go on to use in any number of ways. These algorithms do not presume a distribution over possible inputs, nor do they assume the inputs somehow possess intrinsic uncertainty. Rather, they use randomness as a tool, to find and exploit structures in problem data that would seem "hidden" from the perspective of classical NLA.

What's this about "finding hidden structures?" Consider the problem of highly overdetermined least squares, i.e., the problem of solving

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} \|\mathbf{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2, \tag{1.1}$$

where $\mathbf{A}$ has $m \gg n$ rows. It is well-known that if the columns of $\mathbf{A}$ are orthonormal then (1.1) can be solved in $O(mn)$ time by setting $\boldsymbol{x} = \mathbf{A}^*\boldsymbol{b}$, where $\mathbf{A}^*$ is the transpose of $\mathbf{A}$. The trouble, of course, is that the columns of $\mathbf{A}$ are very unlikely to be orthogonal in any interesting application, and the standard algorithms for solving this problem take $O(mn^2)$ time.

However, what if, by some miracle, we could easily find an $n \times n$ matrix $\mathbf{C}$ for which $\mathbf{A}\mathbf{C}^{-1}$ was column-orthonormal? In this case, we could compute the exact solution $\boldsymbol{x} = (\mathbf{C}^{-1})(\mathbf{C}^{-1})^*\mathbf{A}^*\boldsymbol{b}$ in time

$$O\left(mn + n^3\right)$$

by suitably factoring $\mathbf{C}$. Now, randomized algorithms do not work miracles, but at times they can *approach* the miraculous. In the specific case of (1.1), randomization can be used to quickly identify a basis in which $\mathbf{A}$ is *nearly* column-orthonormal. Any such basis can be incorporated into a standard iterative method from classical NLA. With such an approach, one can reliably solve (1.1) to ϵ-error in time

$$O\left(mn \log\left(\tfrac{1}{\epsilon}\right) + n^3\right)$$

(where we ask forgiveness for being vague about the meaning of "ϵ").

Back to the big picture. The approach to least squares described above has been known for well over ten years now. Since then, an entire suite of compelling results on RandNLA has been established. What's more, the literature also documents the existence of high-performance proof-of-concept implementations that testify to the practicality of these methods. Indeed, as we explain below, randomized algorithms have been developed to address the same basic challenges as classical methods. Randomized algorithms are also very well-suited to address many upcoming challenges with which classical algorithms struggle. RandNLA as a field is slowly achieving a certain level of maturity.

Despite this, substantial interdisciplinary gaps have impeded technology transfer from those doing research in RandNLA to those who might benefit from it. This stems partly from the absence of work that organizes the RandNLA literature in a way that supports the development of high-quality software.

This monograph is our attempt at addressing that absence. With it, we aim to provide a principled and practical foundation for developing high-quality software to address future needs of large-scale linear algebra computations, for scientific computing, large-scale data analysis and machine learning, and other related

applications. Our particular approach is informed by plans to develop such high-quality libraries – a "RandBLAS" and "RandLAPACK," if you will. Towards this end, we have implemented and tested many of the algorithms described herein in both MATLAB[1] and Python.[2] We provide more context on our approach and scope in Section 1.2. But first, we elaborate on the value propositions of RandNLA and the role of randomness in these algorithms.

1.1.1 Four value propositions of randomization

Our goal here is to introduce (and only introduce!) some value propositions for RandNLA. We do this for as broad an audience as possible and we have attempted to keep our introductions short. While these descriptions are unlikely to convince a skeptic, they should at least set the agenda for a debate.

Background: time complexity and FLOP counts. In the sequential RAM model of computing, an algorithm's *time complexity* is its worst-case total number of reads, writes, and elementary arithmetic operations, as a function of input size. Precise expressions for time complexity can be hard to come by and difficult to parse. Therefore it is standard to describe complexity asymptotically, with big-O notation.

In NLA, we also care about how the size of an algorithm's input affects the number of arithmetic operations that it requires. Arithmetic operations are presumed to be floating point operations ("flops") by default, and it is common to refer to an algorithm's *flop count* as a function of input size. Flop counts almost always agree asymptotically with time complexity. But, in contrast with time complexity, flop counts are often given with explicit constant factors. In determining the constant factors accurately one must consider subtleties such as whether a fused multiply-add instruction counts as one or two "flops." We do not consider such subtleties in this monograph.

Fighting the scourge of superlinear complexity. The dimensions of matrices arising in applications typically have semantic meanings. They might represent the number of points in a dataset, or they might be affected by the "fidelity" of a linear model for some nonlinear phenomenon. A scientist who relies on matrix computations will inevitably want to increase the size of their dataset or the fidelity of their model. This is often difficult because the complexity of classical algorithms for high-level linear algebra problems rarely scale linearly with the semantic notion of problem size. This brings us to the first value proposition of RandNLA.

> *For many important linear algebra problems, randomization offers entirely new avenues of computing approximate solutions with linear or nearly-linear complexity.*

To get a sense of why this matters, suppose that one needs to compute a Cholesky decomposition of a dense matrix of order n. The standard algorithm for this takes $n^3/3$ flops. At time of writing, a higher-end laptop can do this calculation for $n = 10{,}000$ in about one second. However, if n is the semantic notion of problem size, and if one wants to solve a problem ten times as large, then the calculation with $n = 100{,}000$ takes over 15 minutes.

[1] https://github.com/BallisticLA/MARLA
[2] https://github.com/BallisticLA/PARLA

There are two lessons in that simple example. The first is that superlinear complexity can be crippling when it comes to solving larger linear algebra problems. The second is that an informed user does well to think of their problem size in a more realistic way. In the case of this example, one should go into the problem thinking in terms of the number of free parameters in an $n \times n$ positive definite matrix: around 50 million when $n = 10{,}000$ and around 5 billion when $n = 100{,}000$.

Remark 1.1.1. One of RandNLA's success stories is a fast algorithm for computing sparse approximate Cholesky decompositions of so-called *graph Laplacians*. In order to keep the length of this monograph under control, we have opted *not* to include algorithms that only apply to sparse matrices. However, we do provide algorithms for computing approximate eigendecompositions of regularized positive semidefinite matrices, and these algorithms can be used to solve linear systems faster than Cholesky in certain applications.

Resisting the siren call of galactic algorithms. The problem of multiplying two $n \times n$ matrices is one of the most fundamental in all of NLA. If we only consider asymptotics, then the fastest algorithms for this task run in less than $O(n^{2.38})$ time. However, the fastest method that is practical (Strassen's algorithm), runs in time $O(n^{\log_2 7})$.

The trouble with these "fast" algorithms is that they have massive constants hidden in their big-O complexity. Such algorithms are called *galactic*, owing to common comparisons between the size of their hidden constants and the number of stars or atoms in the galaxy. And with this, we arrive at the second value proposition of RandNLA.

> *For a handful of important linear algebra problems, the asymptotically fastest (non-galactic) algorithms for computing accurate solutions are, in fact, randomized.*

Highly overdetermined least squares (see page 3) is one such problem.

Striking the Achilles' heel of the RAM model. The RAM model of computing, although useful, is not high-fidelity. Indeed, even in the setting of a shared-memory multi-core machine, it fails to account for the fact that moving data from main memory, through different levels of cache, and onward to processor registers is *much* more expensive than elementary arithmetic on the same data. This fact has been appreciated even in the earliest days of LAPACK's development, over 30 years ago. Its principal consequence is that even if the time complexities of two algorithms match up to *and including* constant factors, their performance by wallclock time can differ by orders of magnitude. This is the third value proposition of RandNLA.

> *Randomization creates a wealth of opportunities to reduce and redirect data movement. Randomized algorithms based on this principle are significantly faster than the best-available deterministic methods by wallclock time.*

Randomized algorithms for computing full QR decompositions with column pivoting fit this description.

Finite-precision arithmetic: once a curse, now a blessing. Finite-precision arithmetic and exact arithmetic are different beasts, and this has real consequences for NLA. For one thing, this limitation introduces many technicalities in understanding accuracy guarantees, even for seemingly straightforward problems like LU decomposition. It is tempting to view it as a curse. However, if we accept it as given, then it can be used to our advantage. Certain computations can be performed with lower precision without compromising the accuracy of a final result.

This perspective brings us to our final value proposition, stated in terms of the concept of sketching, defined momentarily.

> *In RandNLA, it is natural to perform computations on sketches of matrices in lower-precision arithmetic. Depending on how the sketch is constructed, one can be (nearly) certain of avoiding degenerate situations that are known to cause common deterministic algorithms to fail.*

1.1.2 What is, and isn't, subject to randomness

Sampling sketching operators from sketching distributions. We are concerned with algorithms that use random linear dimension reduction maps called *sketching operators*. The sketching operators used in RandNLA come in a wide variety of forms. They can be as simple as operators for selecting rows or columns from a matrix, and they can be even more complicated than algorithms for computing Fast Fourier Transforms. We refer to a distribution over sketching operators as a *sketching distribution*. Given this terminology, we can highlight the following essential fact.

> *For the vast majority of RandNLA algorithms, randomization is only used when sampling from the sketching distribution.*

From an implementation standpoint, one should know that while sketching distributions can be quite complicated, the sampling process always builds on some kind of basic random number generator. Upon specifying a seed for the random number generator involved in sampling, RandNLA algorithms become every bit as deterministic as classical algorithms.

Forming and processing sketches. When a sketching operator is applied to a large data matrix, it produces a smaller matrix called a *sketch*. A wealth of different outcomes can be achieved through different methods for processing a sketch and using the processed representation downstream.

> *Some processing schemes inevitably yield rough approximations to the solution of a given problem. Other processing schemes can lead to high-accuracy approximations, if not exact solutions, under mild assumptions.*

Across these regimes, one of the most popular trends in algorithm analysis is to employ a two-part approach. In the first part, the task is to characterize algorithm output in terms of some simple property of the sketch. In the second part, one can employ results from random matrix theory to bound the probability that the sketch will possess the desired property.

Confidently managing uncertainty. The performance of a numerical algorithm is characterized by the accuracy of its solutions and the cost it incurs to produce those solutions. Naturally, one can expect some variation in algorithm performance when using randomized methods. Luckily, we have the following.

> *Most randomized algorithms "gamble" with only one of the two performance metrics, accuracy or cost. Through optional algorithm parameters, users retain fine-grained control over one of these two metrics.*

Furthermore, when cost is controllable, the algorithm parameters can be adjusted to influence accuracy; when accuracy is controllable, they can be adjusted to influence cost. The effects of these influences can sometimes be masked by variability in run-to-run performance. However, there is a general trend in RandNLA algorithms of becoming more predictable as they are applied to larger problems. At large enough scales, many randomized algorithms are nearly as predictable as deterministic ones.

1.2 This monograph, from an astronaut's-eye view

This monograph started as a development plan for two C++ libraries for RandNLA, primarily working within a shared-memory dense-matrix data model. We prepared a preliminary plan for these libraries in short order by leveraging existing surveys. However, after pausing our writing for some number of months to receive feedback from community members, we found ourselves with many unanswered questions that would affect our implementations. Before long we found ourselves in a cycle of diving ever-deeper into the RandNLA literature with an eye to implementation, each time coming up with more answers and more questions.

This monograph does not answer every question we came across in the foregoing months. Rather, it represents what we know at a time when the best way to answer our remaining questions is to focus on developing the libraries themselves. Therefore we provide the reader with this — a monograph that aggregates material from over 300 references on classical and randomized NLA — which functions partly as a survey and partly as original research. In it, we present new (unifying) taxonomies, candidate application areas, and even a handful of novel theoretical results and algorithms.

Although its scope has greatly increased, the original purpose of this monograph informs its structure. It also contains a number of clear statements about plans for our C++ libraries. Therefore, while we do not want to give the impression that this monograph's value depends on its connections to specific pieces of software, we provide the following remarks on our planned libraries up-front.

> The first library, RandBLAS, concerns basic sketching and is the subject of Section 2. Our hope is that RandBLAS will grow to become a community standard for RandNLA, in the sense that its API would see wider adoption than any single implementation thereof. In order to achieve this goal we think it is important to keep its scope narrowly focused.
>
> The second library, RandLAPACK, concerns algorithms for traditional linear algebra problems (Sections 3 to 5, on least-squares and optimization, low-rank approximation, and additional possibilities, respectively) and advanced sketching functionality (Sections 6 and 7). The design

> spaces of algorithms for these tasks are *large*, and we believe that powerful abstractions are needed for a library to leverage this fact. Consistent with this, we are developing RandLAPACK in an object-oriented programming style wherein *algorithms are objects*. Such a style is naturally instantiated with functors when working in C++.

We have written this monograph to be modular and accessible, without sacrificing depth. The modularity manifests in how there are almost no technical dependencies across Sections 3 to 7. For the sake of accessibility, each section gives background on its core subject. We use two strategies to provide accessibility without sacrificing depth. First, we make liberal use of appendices. In them, the reader can find proofs, background on special topics, low-level algorithm implementation notes, and high-level algorithm pseudocode. Second, our citations regularly indicate precisely where a given concept can be found in a manuscript. Therefore, if we give too brief a treatment on a topic of interest, the reader will know exactly where to look to learn more.

A word on "drivers" and "computational routines"

We designate most algorithms as either *drivers* or *computational routines*. These terms are borrowed from LAPACK's API. In general, drivers solve higher-level problems than computational routines, and their implementations tend to use a small number of computational routines. In our context,

> *drivers* are only for traditional linear algebra problems,

while

> *computational routines* address a mix of traditional linear algebra problems and specialized problems that are only of interest in RandNLA.

Sections 3 and 4 cover drivers *and* the computational routines behind them; they are the most comprehensive sections in this monograph. Section 5 also covers drivers, but at less depth than the two that precede it. In particular, it does not identify algorithmic building blocks that would be considered computational routines. Meanwhile, the advanced sketching functionality in Sections 6 and 7 would *only* be considered for computational routines.

One reason why we use the "driver" and "computational routine" taxonomy is to push much of the RandNLA design space into computational routines. This is essential to keeping drivers simple and few in number. However, it has a side effect: since choices made in the computational routines decisively affect the drivers, it is hard to state theoretical guarantees for the drivers without being prescriptive on the choice of computational routine. This is compounded by two factors. First, we prefer to *not* be prescriptive on choices of computational routines within drivers, since there is always a possibility that some problems benefit more from some approaches than others. Second, even if we recommended specific implementations, it would be very complicated to characterize their performance with consideration to the full range of possibilities for their tuning parameters.

As a result of all this, we make relatively few statements about performance guarantees or computational complexity of driver-level algorithms. While this is a limitation of our approach, we believe it is not severe. One can supplement this monograph with a variety of resources discussed in Section 1.4.

1.3 This monograph, from a bird's-eye view

Section-by-section summaries are provided below to help direct the reader's attention. While space limitations prevent them from being comprehensive, they are effective for what they are. They assume familiarity with standard linear algebra concepts, including least squares models, singular value decomposition, Hermitian matrices, eigendecomposition, and positive (semi)definiteness. We define all of these concepts in Section 1.5 for completeness. Finally, as one disclaimer, some problem formulations below have slight differences from those used in the sections themselves.

Essential notation and conventions. The adjoint of a linear operator $\mathbf{A}$ is denoted by $\mathbf{A}^*$. When $\mathbf{A}$ is a real matrix, the adjoint is simply the transpose. Vectors have column orientations by default, so the standard inner product of two vectors $\boldsymbol{u}, \boldsymbol{v}$ is $\boldsymbol{u}^*\boldsymbol{v}$.

We sometimes call a vector of length n an *n-vector*. If we refer to an $m \times n$ matrix as "tall" then the reader can be certain that $m \geq n$ and reasonably expect that $m > n$. If m is much larger than n and we want to emphasize this fact, then we write $m \gg n$ and would call an $m \times n$ matrix "very tall." We use analogous conventions for "wide" and "very wide" matrices.

Basic Sketching (Section 2)

This section documents our work toward developing a RandBLAS standard. It begins with remarks on the Basic Linear Algebra Subprograms (BLAS), which are to classical NLA as we hope the RandBLAS will be to RandNLA.

Section 2.1 addresses high-level design questions for a RandBLAS standard. By starting with a simple premise, we arrive at the conclusion that it should provide functionality for *data-oblivious sketching* (that is, sketching without consideration to the numerical properties of the data). We then offer our thoughts on how such a library should be organized and how it should handle random number generation.

Section 2.2 summarizes a variety of concepts in sketching. In it, we answer questions such as the following.

- What are the geometric interpretations of sketching?
- How does one measure the quality of a sketch?
- What are the "standard" properties for the first and second moments of sketching operator distributions? When and how are these properties important in RandNLA algorithms?

Detail-oriented readers should consider Section 2.2 alongside Appendix A.1, which presents a novel concept called *effective distortion* that is useful in characterizing the behavior of randomized algorithms for least squares and related problems.

Sections 2.3 to 2.5 review the three types of sketching operator distributions that the RandBLAS might support. These types of distributions consist of dense sketching operators (e.g., Gaussian matrices), sparse sketching operators, and sketching operators based on subsampled fast trigonometric transforms (such as discrete Fourier, discrete cosine, and Walsh-Hadamard transforms). As we explain in Section 2.4, we consider row-sampling and column-sampling as particular types of

sparse sketching. The interested reader is referred to Appendix A.2 for details on a class of sparse sketching operators that is distinct from row or column sampling. These details include notes on high-performance implementations that have not appeared in earlier literature.

Our chapter on basic sketching concludes with Section 2.6, which presents a handful of elementary sketching operations that are not naturally represented by a linear transformation that acts only on the columns or only on the rows of a matrix. These operations arise in the fastest randomized algorithms for low-rank approximation.

Least Squares and Optimization (Section 3)

This is one of three sections that cover driver-level functionality, and it is one of two that discuss drivers *and* computational routines. It is narrower in scope but greater in depth than the other sections that address drivers.

Problem classes. In Section 3.1 we consider a variety of least squares problems within a common framework. The framework describes all problems in terms of an $m \times n$ data matrix $\mathbf{A}$ where $m \geq n$. Given $\mathbf{A}$, any pair of vectors $(\boldsymbol{b}, \boldsymbol{c})$ of respective lengths (m, n) can be considered along with a parameter $\mu \geq 0$ to define "primal" and "dual" *saddle point problems*. The primal problem is always

$$\min_{\boldsymbol{x} \in \mathbb{R}^n} \left\{ \|\mathbf{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 + \mu \|\boldsymbol{x}\|_2^2 + 2\boldsymbol{c}^*\boldsymbol{x} \right\}. \tag{P_μ}$$

The dual problem takes one of two forms, depending on the value of μ:

$$\left.\begin{array}{ll} \min\limits_{\boldsymbol{y} \in \mathbb{R}^m} \left\{ \|\mathbf{A}^*\boldsymbol{y} - \boldsymbol{c}\|_2^2 + \mu \|\boldsymbol{y} - \boldsymbol{b}\|_2^2 \right\} & \text{if } \mu > 0 \\ \min\limits_{\boldsymbol{y} \in \mathbb{R}^m} \left\{ \|\boldsymbol{y} - \boldsymbol{b}\|_2^2 \ : \ \mathbf{A}^*\boldsymbol{y} = \boldsymbol{c} \right\} & \text{if } \mu = 0 \end{array}\right\}. \tag{D_μ}$$

Special cases of these problems include overdetermined and underdetermined least squares, as well as ridge regression with tall or wide matrices. Appendix B.2 gives background on accuracy metrics, sensitivity analysis, and error estimation methods that apply to the most prominent problems under this umbrella.

Section 3.1 considers one type of problem that does not fit nicely into the above framework. Specifically, for a positive semidefinite linear operator $\mathbf{G}$ and a positive parameter μ, it also considers the *regularized quadratic* problem

$$\min_{\boldsymbol{w}} \boldsymbol{w}^*(\mathbf{G} + \mu\mathbf{I})\boldsymbol{w} - 2\boldsymbol{h}^*\boldsymbol{w}. \tag{R_μ}$$

We note that (P_μ) and (D_μ) can be cast to this form when μ is positive. However, to make this reformulation would be to obfuscate the structure in a saddle point problem, rather than reveal it.

Drivers. We start in Section 3.2.1 by covering a low-accuracy method for overdetermined least squares known as *sketch-and-solve*. This method is remarkable for the simplicity of its description and its analysis. It is also the first place where our newly-proposed concept of effective distortion provides improved insight into algorithm behavior.

Sections 3.2.2 and 3.2.3 concern methods for solving problems (P_μ), (D_μ), and (R_μ) to high accuracy. These methods use randomization to find a *preconditioner*. The preconditioner is used to implicitly change the coordinate system that describes the optimization problem, in such a way that the preconditioned problem can easily be solved by iterative methods from classical NLA. These methods are intended for use with certain problem structures (e.g. $m \gg n$) that we clearly identify.

The broader idea of sketch-and-solve algorithms has been successfully used for kernel ridge regression (KRR – see Appendix B.4.1 for a primer). In Section 3.2.4, we reinterpret two algorithms for approximate KRR as sketch-and-solve algorithms for (R_μ). We further identify how the *sketched problems* amount to saddle point problems with $m \gg n$. Appendix B.4.2 details how the saddle point framework is useful in the more complicated of these two settings.

Computational routines. The computational routines that we cover in Section 3.3 only pertain to drivers based on random preconditioning. We kick off our discussion in Section 3.3.1 with background on saddle point problems. Then, Section 3.3.2 addresses preconditioner generation for saddle point problems when $m \gg n$. It opens with a theoretical result (Proposition 3.3.1) characterizing the spectrum of the preconditioned data matrix $\mathbf{A}$ (see also Appendix B.1) before providing a comprehensive overview of implementation considerations. Special attention is paid to how one can generate the preconditioner when $\mu > 0$ at no added cost compared to when $\mu = 0$. In Section 3.3.3, we extend recently proposed methods from the literature to define novel low-memory preconditioners for regularized saddle point problems. Finally, Section 3.3.4 reviews a suite of deterministic iterative algorithms from classical NLA that are needed for randomized preconditioning algorithms.

Low-rank Approximation (Section 4)

Low-rank approximation problems take the following form.

> Given as input an $m \times n$ target matrix $\mathbf{A}$, compute suitably structured factor matrices $\mathbf{E}$, $\mathbf{F}$, and $\mathbf{G}$ where
>
> $$\underset{m \times n}{\hat{\mathbf{A}}} := \underset{m \times k}{\mathbf{E}} \quad \underset{k \times k}{\mathbf{F}} \quad \underset{k \times n}{\mathbf{G}}$$
>
> approximates $\mathbf{A}$. The accuracy of the approximation $\hat{\mathbf{A}} \approx \mathbf{A}$ may vary from one application to another, but we require that $k \ll \min\{m, n\}$.

This section summarizes the massive design spaces of randomized algorithms for such problems, as documented in the existing literature. One of its core contributions is to clarify what parts of this design space are relevant in what situations.

Problem classes. Section 4.1 starts by explaining the significance of the SVD and eigendecomposition in relation to principal component analysis. From there, it introduces the reader to a handful of *submatrix-oriented decompositions* – CUR, one-sided interpolative decompositions (one-sided ID), and two-sided interpolative decompositions (two-sided ID) – along with their applications. Section 4.1 concludes with guidance on how one should and should-not quantify approximation error in low-rank approximation problems. We note that this background is *much* more detailed than that Section 3.1 provided on least squares and optimization problems. This extra background will be important for many readers.

Drivers. Section 4.2 gives concise yet comprehensive overviews for RandNLA algorithms for SVD and Hermitian eigendecomposition (§4.2.1 and 4.2.2) as well as CUR and two-sided interpolative decomposition (§4.2.3). In the process, we take care to prevent misunderstandings in what we mean by a *Nyström approximation* of a positive semidefinite matrix. Pseudocode is provided for at least one algorithm for each of these problems.

Computational routines. As is typical for surveys on this topic, we identify *QB decomposition* (§4.3.2) and *column subset selection (CSS) / one-sided ID* (§4.3.4) as the basic building blocks for most drivers. We also isolate power iteration (§4.3.1) and partial column-pivoted matrix decompositions (§4.3.3) as subproblems with nontrivial design spaces that are important to low-rank approximation.

Some of the building blocks covered cumulatively from Sections 4.3.1 to 4.3.4 can be used to compute low-rank approximations iteratively. If one seeks an approximation that is accurate to within some given tolerance, then these iterative algorithms require methods for estimating norms of linear operators; we cover such norm estimation methods briefly in Section 4.3.5. Appendix C.2 contains pseudocode for seven computational routines and details their dependency structure.

Further Possibilities for Drivers (Section 5)

This section covers a handful of independent topics.

Section 5.1 covers *multi-purpose matrix decompositions*. Section 5.1.1 explains a simple algorithm for computing an unpivoted QR decomposition of a tall-and-thin matrix of full column rank; the algorithm uses randomization to precondition Cholesky QR for numerical stability. Section 5.1.2 first describes an existing algorithm from the literature for Householder QRCP of matrices with any aspect ratio, and then presents an extension of preconditioned Cholesky QR that incorporates pivoting and allows for rank-deficient matrices. Section 5.1.3 summarizes methods for computing decompositions known by various names (UTV, URV, QLP) that all aim to serve as cheaper surrogates for the SVD.

Section 5.2 addresses randomized algorithms for the solution of unstructured linear systems. This includes direct methods based on accelerating (or safely bypassing) pivoting in matrix decompositions (§5.2.1) as well as iterative methods (§5.2.2). Some of these iterative methods were developed fairly recently and are a subject of considerable practical interest.

Section 5.3 considers the problem of estimating the trace of a linear operator. This problem is unique in the context of this monograph, since it makes no sense to consider in the shared-memory dense-matrix data model. We have opted to cover it anyway since randomized methods are *extremely* effective for it. Section 5.3.1 introduces the elementary Girard–Hutchinson estimator developed in the late 1980s. Section 5.3.2 covers methods that benefit from contemporary developments on randomized algorithms for low-rank approximation. Finally, Section 5.3.3 covers methods for computing the trace of $f(\mathbf{B})$ where $\mathbf{B}$ is a Hermitian matrix and f is a matrix function. We give a significant amount of background material to help the newcomer understand the methods in this last category.

Advanced Sketching: Leverage Score Sampling (Section 6)

Leverage scores constitute measures of importance for the rows or columns of a matrix. They can be used to define data-aware sketching operators that implement row or column sampling.

Section 6.1 introduces three types of leverage scores: standard leverage scores, subspace leverage scores, and ridge leverage scores. We explain how each type is suitable for sketching with different downstream tasks in mind. For example, a proposition in Section 6.1.1 bounds the probability that a row-sampling operator satisfies a subspace embedding property for the range of a matrix $\mathbf{A}$. The bound shows that if rows are sampled according to a distribution $\boldsymbol{q}$, then it becomes more likely that the subspace embedding property holds as $\boldsymbol{q}$ approaches $\mathbf{A}$'s standard leverage score distribution.

Section 6.2 covers randomized algorithms for approximating leverage scores. Such approximation methods are important since leverage scores are expensive to compute except when working with highly structured problem data. The structure of these algorithms bears similarities to those seen in earlier sections. For example, Section 6.2.2 explains how a longstanding algorithm for approximating subspace leverage scores can be extended with QB approaches from Section 4.3.2.

Advanced Sketching: Tensor Product Structures (Section 7)

Tensor computations are the domain of *multilinear algebra*. As such, it is reasonable to exclude them from the scope of a standard library from RandNLA. However, at the same time, it is reasonable for a RandNLA library to support the core subproblems in tensor computations that are linear algebraic in nature. Sketching implicit matrices with tensor product structure fits this description.

This section reviews efficient methods for sketching matrices with Kronecker product or Khatri–Rao product structures (see §7.1 for definitions). The material in 7.2.1–7.2.4 concerns data-oblivious sketching distributions that are similar to those from Section 2 but modified for the tensor product setting. Section 7.2.5, by contrast, concerns data-aware sketching methods based on leverage score sampling. Notably, there are methods to efficiently sample from the *exact* leverage score distributions of tall matrices with Kronecker and Khatri–Rao product structures without explicitly forming those matrices.

For completeness, Section 7.3 discusses motivating applications (specifically, tensor decomposition algorithms) that entail sketching matrices with these structures.

1.4 Recommended reading

This monograph is heavily influenced by a recent and sweeping survey by Martinsson and Tropp [MT20]. We draw detailed comparisons to that work in Section 1.4.5. But first, we give remarks on other resources of note for learning about RandNLA.

1.4.1 Tutorials, light on prerequisites

RandNLA: randomized numerical linear algebra, by Drineas and Mahoney [DM16].

Depending on one's background (and schedule!) this article can be read in one sitting. It requires no knowledge of NLA or probability. In fact, it does not even

presume that the reader already cares about matrix computations. It starts with basic ideas of matrix approximation by subsampling, explains the effect of sampling in different data-aware ways, and frames general data-oblivious sketching as "preprocessing followed by uniform subsampling." It summarizes, at a very high level, significant results of RandNLA in least squares, low-rank approximation, and the solution of structured linear systems known as *Laplacian systems*.

Lectures on randomized numerical linear algebra, by Drineas and Mahoney [DM18].

This book chapter is useful for those who want to see representative banner results in RandNLA *with proofs*. It covers algorithms for least squares and low-rank approximation. Its proofs emphasize decoupling deterministic and probabilistic aspects of analysis. Among resources that engage with the theory of RandNLA, it is notable for its brevity and its self-contained introductions to linear algebra and probability.

1.4.2 Broad and proof-heavy resources

Sketching as a tool for numerical linear algebra, by Woodruff [Woo14].

This monograph proceeds one problem at a time, starting with ℓ_2 regression, then on to ℓ_1 regression, then low-rank approximation, and finally graph sparsification. It develops the technical machinery needed for each of these settings, at various levels of detail. Among resources that address RandNLA theory, it is notable for its treatment of lower bounds (i.e., limitations of randomized algorithms).

An introduction to matrix concentration inequalities, by Tropp [Tro15].

This monograph gives an introduction to the theory of matrix concentration and its applications. It is not about RandNLA *per se*, but several of its applications do focus on RandNLA. The course notes [Tro19] build on this monograph, exploring theory and applications of matrix concentration developed after [Tro15] was written.

Lecture notes on randomized linear algebra, by Mahoney [Mah16].

These notes are fairly comprehensive in their coverage of results in RandNLA up to 2013. They address matrix concentration, approximate matrix multiplication, subspace embedding properties of sketching distributions, as well as various algorithms for least squares and low-rank approximation. These notes are distinct from [Woo14] in that they address theory and practice. (Of course, being course notes, they are not suitable as a formal reference.)

1.4.3 Perspectives on theory, light on proofs

Randomized algorithms for matrices and data, by Mahoney [Mah11].

This monograph heavily emphasizes concepts, interpretations, and qualitative proof strategies. It is a good resource for those who want to know what RandNLA can offer in terms of theory for the least squares and low-rank approximation. It is notable for the effort it expends to connect RandNLA theory to theoretical developments in other disciplines.

Determinantal point processes in randomized numerical linear algebra, by Dereziński and Mahoney [DM21a].

This article gives an overview of RandNLA theory from the perspective of determinantal point processes and statistical data analysis. Among the many resources for learning about RandNLA, it is notable for offering a distinctly *prospective* (rather than *retrospective*) viewpoint.

1.4.4 Deep investigations of specific topics

Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions, by Halko, Martinsson, and Tropp [HMT11].

As of late 2022, this article is the single most influential resource on RandNLA. Its introduction includes a history of how randomized algorithms have been used in numerical computing, as well as a brief summary of (then) active areas of research in RandNLA. Following the introduction, it focuses exclusively on low-rank approximation. It is extremely thorough in its treatment of both theory and practice.

This article is now somewhat out of date and is partially subsumed by [MT20]. However, it is still of distinct value for the fact that it proves all of its main results (and in certain cases, by novel methods). It also includes some algorithms that are not found in [MT20].

Randomized algorithms in numerical linear algebra, by Kannan and Vempala [KV17a].

This survey provides a detailed theory of row and column sampling methods. It also includes methods for tensor computations.

Randomized methods for matrix computations, by Martinsson [Mar18].

This book chapter focuses on practical aspects of randomized algorithms for low-rank approximation. In this regard, it is important to note that while [HMT11] provided thorough coverage of this topic *at the time*, the more recent [Mar18] reviews important practical advances developed after 2011. Among resources that provide an in-depth investigation into low-rank approximation, is notable for how it also includes algorithms for full-rank matrix decomposition.

1.4.5 Randomized numerical linear algebra: Foundations and Algorithms

Martinsson and Tropp's recent *Acta Numerica* survey, [MT20], covers a wide range of topics, each with substantial technical and historical depth. We have benefited from it tremendously in developing our plans for RandBLAS and RandLAPACK. Because we have found this resource so useful – and, at the same time, because we have gone through the trouble of writing a distinct monograph that is just as long – we think there is value in highlighting how it differs from our work.

Basic sketching. By comparison to [MT20], we focus more on implementation than on theory. The outcome of this is the broadest-yet review of the literature relevant to the implementation of sketching methods. In the appendices, we provide novel technical contributions to sketching theory and practice.

See Section 2 and Appendix A, [MT20, §7 – §9].

Least squares and optimization. Our coverage of these concepts is comprehensive, insofar as optimization can be reduced to linear algebra. It also includes a number of novel technical contributions and a review of relevant software. By comparison, [MT20] provides very limited coverage of this area, as acknowledged in [MT20, §1.6].

See Section 3 and Appendix B, [MT20, §10].

Low-rank approximation. Our approach here is very different than that of [MT20]. It provides effective scaffolding for a reader to get a handle on the vast literature on low-rank approximation. However, it comes at the price of creating fewer opportunities for mathematical explanations. Separately, our coverage here is distinguished by providing an overview of software that implements randomized algorithms for low-rank approximation.

See Section 4 and Appendix C, [MT20, §11 – §15].

Full-rank matrix decompositions. By comparison to [MT20], we emphasize a broader range of matrix decompositions and more algorithms for computing them. One of the algorithms we cover is novel and is accompanied by proofs that characterize its behavior. For the algorithms covered here *and* in [MT20], the latter provides more mathematical detail.

See Section 5.1 and Section 5.2.1, Appendix D, [MT20, §16].

Kernel methods. Randomized methods have proven very effective in processing machine learning models based on *positive definite kernels*. They are also effective in approximating matrices from scientific computing induced by *indefinite kernels*. Both of these topics are addressed in [MT20]. We only address the former topic, and we do so in a way that emphasizes the resulting linear algebra problems.

See Sections 3.2.2, 3.2.4, and 6.1.3, Appendix B.4.1, [MT20, §19, §20].

Linear system solvers. We cover slightly more material for solving unstructured linear systems than [MT20]. However, we do not cover methods that are specific to sparse problems. As a result, we do not cover a prominent method for approximate Cholesky decompositions of sparse graph Laplacians.

See Sections 3.2.3 and 5.2, [MT20, §17, §18].

Trace estimation. We have the luxury of being able to cover recently-developed algorithms that were not available when [MT20] was written. This includes two methods that provide the first major advances in trace estimation since the late 1980s. We provide less depth than [MT20] on average, with the notable exception of stochastic Lanczos quadrature.

See Section 5.3, [MT20, §4, §6].

Advanced sketching. Both this monograph and [MT20] cover leverage score sampling and sketching operators with tensor product structures. We cover these topics in substantially more detail, spending a full ten pages on each of them. We do this partly because these topics complement one another: implicit matrices with tensor product structures are among the best candidates for practical leverage score sampling, nearly on par with kernel matrices from machine learning.

See Sections 6 and 7, [MT20, §7.4, §9.4, §9.6, §19.2.3].

1.5 Notation and terminology

Our notation is summarized in Table 1.1; we also define some of this notation below as we explain basic concepts.

Matrices and vectors

Let $\mathbf{A}$ be an $m \times n$ matrix or linear operator. We use $\mathbf{A}^*$ to denote its adjoint (transpose, in the real case) and $\mathbf{A}^\dagger$ to denote its pseudo-inverse. It is called *Hermitian* if $\mathbf{A}^* = \mathbf{A}$ and *positive semidefinite* if it is Hermitian and all of its eigenvalues are nonnegative. We often abbreviate "positive semidefinite" with "psd."

We sometimes find it convenient to write $\mathbf{A} \in \mathbb{R}^{m \times n}$. However, it should be understood that the methods in this monograph generally apply to both real and complex matrices. We therefore tend to define a matrix by phrases like "$\mathbf{A}$ is m-by-n" or "an m-by-n matrix $\mathbf{A}$." We often call a vector of length n an *n-vector*; vectors are oriented as columns by default.

For $m \geq n$, a *QR decomposition* of $\mathbf{A}$ consists of an $m \times n$ column-orthonormal matrix $\mathbf{Q}$ and an upper-triangular matrix $\mathbf{R}$ for which $\mathbf{A} = \mathbf{QR}$. Those familiar with the NLA literature will note that this is typically called the *economic* QR decomposition. If $\mathbf{A}$ has rank $k < \min(m, n)$, then we also consider it valid for $\mathbf{Q}$ to be $m \times k$ and for $\mathbf{R}$ to be $k \times n$. We also consider *QR decomposition with column pivoting* (QRCP). To describe QRCP, we say that if $J = (j_1, \ldots, j_n)$ is a permutation of $[\![n]\!]$, then

$$\mathbf{A}[:, J] = [\boldsymbol{a}_{j_1}, \boldsymbol{a}_{j_2}, \ldots, \boldsymbol{a}_{j_n}]$$

where $\boldsymbol{a}_i$ is the i^{th} column of $\mathbf{A}$. In this notation, QRCP produces an index vector J and factors $(\mathbf{Q}, \mathbf{R})$ that provide a QR decomposition of $\mathbf{A}[:, J]$.

Now let $\mathbf{A}$ have rank k. Its *singular value decomposition* (SVD) takes the form $\mathbf{A} = \mathbf{U\Sigma V}^*$, where the matrices $(\mathbf{U}, \mathbf{V})$ have k orthonormal columns and $\mathbf{\Sigma} = \operatorname{diag}(\sigma_1, \ldots, \sigma_k)$ is a square matrix with sorted entries $\sigma_1 \geq \cdots \geq \sigma_k > 0$. The SVD can also be written as a sum of rank-one matrices: $\mathbf{A} = \sum_{i=1}^{k} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^*$, where $(\boldsymbol{u}_i, \boldsymbol{v}_i)$ are the i^{th} columns of $(\mathbf{U}, \mathbf{V})$ respectively. Those familiar with the NLA literature will note that this is typically called the *compact SVD*.

Probability and our usage of the term "random."

A *Rademacher* random variable uniformly takes values in $\{+1, -1\}$. The initialism "iid" expands to *independent and identically distributed*.

We often abuse terminology and say that a matrix "randomly" performs some operation. In reality, matrices only perform deterministic calculations, and randomness only comes into play when the matrix is first constructed. This convention

extends to "matrices" that are abstract linear operators, in which case randomness is only involved in constructing the data that defines the operator.

Unqualified use of the term "random" before performing an action with a finite set of outcomes (such as sampling components from a vector, applying a permutation, etc...) means the randomness is uniform over the space of possible actions.

Table 1.1: Notation

Arrays and indexing	
A_{ij} or $\mathbf{A}[i,j]$	$(i,j)^{\text{th}}$ entry of a matrix $\mathbf{A}$
$\boldsymbol{a}_i$ or $\mathbf{A}[:,i]$	i^{th} column of $\mathbf{A}$
v_i or $\boldsymbol{v}[i]$	i^{th} component of a vector $\boldsymbol{v}$
$[\![m]\!]$	index set of integers from 1 to m
I or J	partial permutation vector for indexing into an array
$\lvert I\rvert$	length of an index vector
$\mathbf{A}[I,:]$	submatrix consisting of (permuted) rows of $\mathbf{A}$
$\mathbf{A}[:,J]$	submatrix consisting of (permuted) columns of $\mathbf{A}$
$:k$	index into the leading k elements of an array, along an axis of length at least k
$k:$	index into the trailing $n-k+1$ elements of an array, along an axis of length $n \geq k$
Reserved symbols	
$\mathbf{S}$	sketching operator
$\mathbf{I}_k$	identity matrix of size $k \times k$
$\boldsymbol{\delta}_i$	i^{th} standard basis vector of implied dimension
$\mathbf{0}_n$	zero vector of length n
$\mathbf{0}_{m\times n}$	zero matrix of size $m \times n$
Linear algebra	
$\lVert\boldsymbol{x}\rVert_2$ or $\lVert\boldsymbol{x}\rVert$	Euclidean norm of a vector $\boldsymbol{x}$
$\lVert\mathbf{A}\rVert_2$	spectral norm of $\mathbf{A}$
$\lVert\mathbf{A}\rVert_{\text{F}}$	Frobenius norm of $\mathbf{A}$
$\operatorname{cond}(\mathbf{A})$	Euclidean condition number of $\mathbf{A}$
$\lambda_i(\mathbf{A})$	i^{th} largest eigenvalue of $\mathbf{A}$
$\sigma_i(\mathbf{A})$	i^{th} largest singular value of $\mathbf{A}$
$\mathbf{A}^*$	adjoint (transpose, in the real case) of $\mathbf{A}$
$\mathbf{A}^\dagger$	Moore–Penrose pseudoinverse of $\mathbf{A}$
$\mathbf{A}^{1/2}$	Hermitian matrix square root
$\mathbf{A} \preceq \mathbf{B}$	the matrix $\mathbf{B}-\mathbf{A}$ is positive semidefinite
Matrix decomposition conventions	
$\mathbf{A} = \mathbf{QR}$	QR decomposition (economic, by default)
$(\mathbf{Q},\mathbf{R},J) = \texttt{qrcp}(\mathbf{A})$	QR with column-pivoting; $\mathbf{A}[:,J] = \mathbf{QR}$.
$\mathbf{A} = \mathbf{U\Sigma V}^*$	singular value decomposition (compact, by default)
$\mathbf{R} = \texttt{chol}(\mathbf{G})$	upper triangular Cholesky factor of $\mathbf{G} = \mathbf{R}^*\mathbf{R}$
Probability	
$X \sim \mathcal{D}$	X is a random variable following a distribution $\mathcal{D}$
$\mathbb{E}[\mathbf{X}]$	expected value of a random matrix $\mathbf{X}$
$\operatorname{var}(X)$	variance of a random variable X
$\Pr\{E\}$	probability of the event E

Section 2

Basic Sketching

The BLAS (Basic Linear Algebra Subprograms) were originally a collection of Fortran routines for computations including vector scaling, vector addition, and applying Givens rotations [LHK+79]. They were later extended to operations such as matrix-vector multiplication and triangular solves [DDH+88] as well as matrix-matrix multiplication, block triangular solves, and symmetric rank-k updates [DDH+90]. These routines have subsequently been organized into three *levels* called BLAS 1, BLAS 2, and BLAS 3.

Over the years the BLAS have evolved into a *community standard*, with implementations targeting different machine architectures in many programming languages. This standardization has been instrumental in the development of linear algebra libraries – from the early days of LINPACK, through to LAPACK, and on to modern libraries such as PLASMA and SLATE [DMB+79; DDD+87; ABB+99; ADD+09; DGH+19; KWG+17; AAB+17]. It has also reduced the coupling between hardware and software design for NLA. Indeed, the spirit of the BLAS has been adapted to accommodate dramatic changes in prevailing architectures, such as those faced by ScaLAPACK and MAGMA [CDO+95; CDD+96; TDB10; NTD10].

This section summarizes our progress on the design of a "RandBLAS" library,

which is to be to RandNLA as BLAS is to classical NLA. Section 2.1 begins by speaking to high-level scope and design considerations. From there, Section 2.2 summarizes sketching concepts that remain important throughout this monograph; we encourage the reader to not dwell too long on this section and instead return to it as-needed later on. Sections 2.3 through 2.6 present our plans for sketching dense data matrices. In brief: our near-term plans are for the RandBLAS to support sketching operators which could naturally be represented by dense arrays or by sparse matrices with certain structures; we consider row sampling and column sampling as particular types of sparse sketching.

2.1 A high-level plan

We begin with a simple premise.

> The RandBLAS' defining purpose should be to facilitate implementation of high-level RandNLA algorithms.

This premise works to reduce the RandBLAS' scope, as there are "basic" operations in RandNLA which do not support this purpose.[1] Another way that we reduce the scope of the RandBLAS is to only consider sketching dense data matrices. It may be reasonable to lift this restriction in the future, and consider methods for producing dense sketches of sparse data matrices.

Our premise for the RandBLAS suggests that it should be concerned with *data-oblivious sketching* – that is, sketching without consideration to the numerical properties of a dataset. We identify three categories of operations on this topic:

- sampling a random sketching operator from a prescribed distribution,
- applying a sampled sketching operator to a data matrix, and
- sketching that is not naturally expressed as applying a single a linear operator to a data matrix.

These categories are somewhat analogous to BLAS 1, BLAS 2, and BLAS 3, insofar as their implementations admit more and more opportunities for machine-specific performance optimizations. At this time, however, we do not advocate for any formalization of "RandBLAS levels."

We note that data-oblivious sketching is not the only kind of sketching of value in RandNLA. Indeed, *data-aware* sketching operators such as those derived from power iteration are extremely important for low-rank approximation (see Section 4.3.1). Methods for row or column sampling based on leverage scores are also useful for kernel ridge regression and certain tensor computations; see Sections 6 and 7. Although important, most of the functionality for producing or applying these sketching operators should be addressed in higher-level libraries.

In the material under the next two headings, we address the questions of how to handle random number generation and reproducibility in the RandBLAS.

[1] For example, the problem of accepting two matrices and using randomization to approximate their product is certainly basic, and it is of conceptual value [DKM06a]. However, it is rarely used as an explicit building block in higher-level RandNLA algorithms.

2.1.1 Random number generation

For reproducibility's sake it is important that the RandBLAS include a specification for *random number generators* (RNGs).

We believe the RandBLAS should use *counter-based random number generators* (CBRNGs), which were first proposed in [SMD+11]. A CBRNG returns a random number upon being called with two integer parameters: the *counter* and the *key*. The time required for the CBRNG to return does not depend on either of these parameters. A serial application can set the key at the outset of the program and never change it. Parallel applications (particularly parallel simulations) can use different keys across different threads. Sequential calls to the CBRNG with a fixed key should use different values for the counter. For a fixed key, a CBRNG with a p-bit integer counter defines a stream of random numbers with period length 2^p.

In our context, CBRNGs are preferable to traditional state-based RNGs such as the Mersenne Twister. A key reason for this is that CBRNGs maximize flexibility in the order in which a sketching operator is generated. For example, given a user-provided counter offset c which acts as a random seed, the $(i,j)^{\text{th}}$ entry of a dense $d \times m$ sketching operator can be generated with counter $c + (i + dj)$. The fact that these computations are embarrassingly parallel will be important for vendors developing optimized RandBLAS implementations. We note that this flexibility also provides an advantage over widely-used linear congruential RNGs, which have separate shortcomings of performing very poorly on statistical tests [SMD+11, §2.2.1].

Particular examples of CBRNGs include Philox, ARS, and Threefish, each of which was defined in [SMD+11] and implemented in the Random123 library. These CBRNGs have periods of 2^{128}, can support 2^{64} different keys, and pass standard statistical tests for random number generators. Random123 provides the core of the sketching layer of the LibSkylark RandNLA library [KAI+15]. Implementations of Philox and ARS can also be found in MKL Vector Statistics [Int19, §6.5].

Shift-register RNGs

We have observed that the CBRNGs in Random123 are significantly more expensive than the state-based shift-register RNGs developed by Blackman and Vigna [BV21]. In fact, Blackman and Vigna's generators are so fast that we have been able to implement a method for applying a Gaussian sketching operator to a sparse matrix that beats Intel MKL's sparse-times-dense matrix multiplication methods. However, in the application where we observed that performance, processing the sketch downstream was more expensive than computing the sketch in the first place. Therefore, while CBRNGs were substantially more expensive in that application, their longer runtimes were inconsequential in that case. This longer runtime can be viewed as a price we pay for prioritizing reproducibility of sketching across compute environments with different levels of parallelism.

The overall situation is this:

> State-based RNGs may be preferable to CBRNGs *if* sketching is the bottleneck in a RandNLA algorithm *and* where the cost of random number generation decisively affects the cost of sketching. At this time we have no evidence that high-performance implementations of RandNLA algorithms run into such bottlenecks. Such evidence may arise in the future and warrant reconsideration to fast state-based RNGs for the Rand-

BLAS, particularly if major advances are made in hardware-accelerated sketching algorithms.

2.1.2 Portability, reproducibility and exception handling

We believe it is important that the RandBLAS lends itself to portability across programming languages. Therefore we plan for the RandBLAS to have a procedural API and make use of no special data structures beyond elementary structs. Higher-level libraries should take responsibility for exposing RandBLAS functionality with sophisticated abstractions. In particular, we plan for RandLAPACK to expose RandBLAS functionality through a suitable object-oriented linear operator interface. A key goal of this interface will be to make it possible to implement high-level RandNLA algorithms with minimal assumptions on the sketching operator's distribution. Such an interface will also reduce the coupling between determining RandBLAS's procedural API and prototyping RandLAPACK.

Debugging high-performance numerical code is notoriously difficult. Care must be taken in the design of the RandBLAS so as to not contribute to this difficulty. Indeed, it is essential that the RandBLAS be reproducible to the greatest extent possible. The actual extent of the reproducibility will depend on factors outside of our control. For example – the RandBLAS cannot offer bitwise reproducibility guarantees unless the BLAS does the same (see Remark 2.1.1). Therefore the main challenge for reproducibility for RandBLAS is in random number generation; this challenge can be resolved comprehensively through the aforementioned CBRNGs.

A key source of exceptions in NLA is the presence of `NaN`s or `Inf`s in problem data. Extremely sparse sketching matrices (such as those from Section 2.4.2) might not even read every entry of a data matrix, and so they might miss a `NaN` or `Inf`. Those routines will be clearly marked as carrying this risk. The majority of routines in the RandBLAS and RandLAPACK will *not* carry this risk: they will propagate `NaN`s and `Inf`s. (See [DDG+22] for a more detailed discussion of how the BLAS and LAPACK (should) deal with exceptions.) For any such routine the exact behavior will depend on how the random sketching operator interacts with the problem data. For example, if a data matrix containing multiple `Inf`s is sketched twice using different random seeds, then it is possible that an entry of the first sketch is an `Inf` while the corresponding entry of the second sketch is a `NaN`.

Remark 2.1.1. Making the BLAS bitwise reproducible is challenging because floating-point addition is not associative, and the order of summation can vary depending on the use of parallelism, vectorization, and other matters [RDA18]. Summation algorithms that guarantee bitwise reproducibility do exist [ADN20]. These algorithms may become practical on hardware that implements the latest IEEE 754 floating point standard, which includes a recommended instruction for bitwise-reproducible summation [IEE19]. However, we leave these matters to future work.

2.2 Helpful things to know about sketching

The purpose of sketching is to enact dimension-reduction so that computations of interest can be performed on a smaller matrix called a *sketch*. While precise computations performed on the sketch can vary dramatically, the simple statement of sketching's purpose lets us deduce the following facts.

- Sketching operators applied to the *left* of a data matrix must *must be wide* (i.e., they must have more columns than rows).
- Sketching operators applied to the *right* of a data matrix *must be tall* (i.e., they must have more rows than columns).

This is to say, in left-sketching we require that $\mathbf{SA}$ has fewer rows than $\mathbf{A}$, and in right-sketching we require that $\mathbf{AS}$ has fewer columns than $\mathbf{A}$. These facts are true regardless of the aspect ratio of the data matrix; see Figure 2.1 for an illustration. The facts are important because sketching operators in the literature are often defined under the assumption of left-sketching.

Before we proceed further, we reiterate some important advice.

> *We encourage the reader to not dwell too long on this section (Section 2.2) and instead return to it as needed later on.*

With that, Section 2.2.1 explains geometric interpretations of sketching from the left and right. It also introduces the concepts of "sketching in the embedding regime" and "sketching in the sampling regime." Section 2.2.2 covers concepts of subspace embedding distortion and the oblivious subspace embedding property – these are *central* to RandNLA theory, but they play a modest role in this monograph. Section 2.2.3 states properties of sketching distributions that should hold as part of a 'sanity check' for whether a proposed distribution is reasonable.

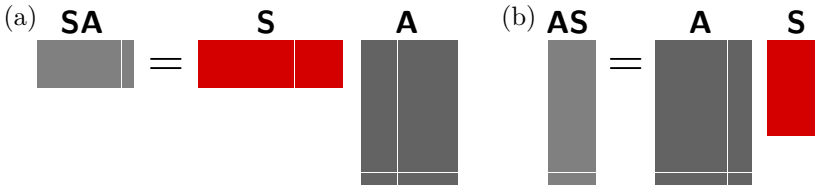

Figure 2.1: The left plot (a) shows that a sketching operator $\mathbf{S}$ applied to the left of a matrix $\mathbf{A}$ is wide, whereas the right plot (b) shows that a sketching operator $\mathbf{S}$ applied to the right of a matrix $\mathbf{A}$ is tall. These stated properties hold universally; there are no exceptions for any kind of sketching. Separately, we note that both cases in the figure illustrate *sketching in the sampling regime* in the sense of Section 2.2.1.

2.2.1 Geometric interpretations of sketching

Prototypical left-sketching and right-sketching

Sketching $\mathbf{A}$ from the left preserves its number of columns. Therefore, it is suitable for things such as estimating *right* singular vectors. We often interpret a left-sketch $\mathbf{SA}$ as a compression of the range of $\mathbf{A}$ to a space of lower ambient dimension. In the special case when $\text{rank}(\mathbf{SA}) = \text{rank}(\mathbf{A})$, there is a sense in which the quality of the compression is completely independent from the spectrum of $\mathbf{A}$.

Sketching $\mathbf{A}$ from the right preserves its number of rows, and is suitable for things such as estimating *left* singular vectors. Conceptually, the right-sketch $\mathbf{AS}$ can be interpreted as a sample from the range of $\mathbf{A}$ using a test matrix $\mathbf{S}$. In the special case when $\text{rank}(\mathbf{AS}) \ll \text{rank}(\mathbf{A})$ then it is appropriate to think of this as a "lossy sample" from a much larger "population," and it is natural to want this sample to capture as much information as possible for some sub-population of interest.

Equivalence of left-sketching and right-sketching

Left-sketching and right-sketching can be reduced to one another by replacing $\mathbf{A}$ and $\mathbf{S}$ by their adjoints. For example, a left-sketch $\mathbf{SA}$ can be viewed as a sample from the row space of $\mathbf{A}$, equivalent to the right-sketch $\mathbf{A}^*\mathbf{S}^*$. Conversely, a right-sketch $\mathbf{AS}$ can be viewed as a compression of the row space of $\mathbf{A}$, equivalent to the left-sketch $\mathbf{S}^*\mathbf{A}^*$. Therefore it is artificial to strongly distinguish sketching operators by whether they are first defined for left-sketching or right-sketching.

This leads us to an important point.

> If $\mathcal{D}_{d,m}$ is a distribution over wide $d \times m$ sketching operators, it is *canonically extended* to a distribution over tall $n \times d$ sketching operators by sampling $\mathbf{T}$ from $\mathcal{D}_{d,n}$ and then returning the adjoint $\mathbf{S} = \mathbf{T}^*$.

The notation in the statement above is carefully chosen: since our "data matrices" are typically $m \times n$, a typical left-sketching operator requires m columns, and a typical right-sketching operator requires n rows.

The embedding and sampling regimes

While it is artificial to associate a sketching distribution only with left-sketching or only with right-sketching, there are indeed families of sketching operators that are suited to qualitatively different situations. The following terms help with our discussion of such families.

> Sketching in the *embedding regime* is the use of a sketching operator that is *larger* than the data to be sketched. Sketching in the *sampling regime* is the use of a sketching operator that is *far smaller* than the data to be sketched.

In the above definitions one quantifies the size of an operator (or matrix) by the product of its number of rows and number of columns.

In Section 3, we will see that sketching in the embedding regime is nearly universal in randomized algorithms for least squares and related problems. In Section 4, we will see that sketching in the sampling regime is the foundation of randomized algorithms for low-rank approximation. Over these sections we tend to see sketching in the embedding regime happen from the left, and sketching in the sampling regime happen from the right. We stress that these tendencies are consequences of *exposition*; they do not always hold when developing or using RandNLA software.

2.2.2 Sketch quality

Let L be a subset of some high-dimensional Euclidean space $\mathbb{R}^m$, $\mathbf{S}$ be a sketching operator defined on $\mathbb{R}^m$, and consider the sketch $\mathbf{S}L$. Intuitively, $\mathbf{S}L$ should be useful if its geometry is somehow "effectively the same" as that of L. Here we discuss the preferred ways to quantify changes to geometry in RandNLA. We focus on methods suitable for when L is a linear subspace, but we also consider when L is a finite point set.

We acknowledge up-front that it only does so much good to measure the quality of an individual sketch. Indeed, in order to make predictive statements about the behavior of algorithms, it is necessary to understand how the distribution of a sketching operator $\mathbf{S} \sim \mathcal{D}$ induces a distribution over measures of sketch quality in

a given application. It is further necessary to analyze *families of distributions* $\mathcal{D}_{d,m}$ parameterized by an embedding dimension d, since the size of a sketch is often a key parameter that a user can control.

Subspace embeddings

Let $\mathbf{S}$ be a $d \times m$ sketching operator and L be a linear subspace of $\mathbb{R}^m$. We say that $\mathbf{S}$ *embeds* L *into* $\mathbb{R}^d$ and that it does so with *distortion* $\delta \in [0, 1]$ if $\boldsymbol{x} \in L$ implies

$$(1 - \delta)\|\boldsymbol{x}\|_2 \leq \|\mathbf{S}\boldsymbol{x}\|_2 \leq (1 + \delta)\|\boldsymbol{x}\|_2. \tag{2.1}$$

We often call such an operator an δ-*embedding*.

The concept of a subspace embedding was first used implicitly in RandNLA by [DMM06]; the sketching operators used in [DMM06] were based on a type of *data-aware sketching* called leverage score sampling (discussed in Section 6). The first explicit definition of subspace embeddings was given by [Sar06], who focused on *data-oblivious* sketching. We address data-oblivious subspace embeddings in detail momentarily.

The most transparent use of subspace embedding distortion arises when L is the range of a matrix $\mathbf{A}$. In this context, $\mathbf{S}$ is a δ-embedding for L if and only if the following two-sided linear matrix inequality holds:

$$(1 - \delta)^2\mathbf{A}^*\mathbf{A} \preceq (\mathbf{SA})^*(\mathbf{SA}) \preceq (1 + \delta)^2\mathbf{A}^*\mathbf{A}. \tag{2.2}$$

In other words, the distortion of $\mathbf{S}$ as an embedding for range($\mathbf{A}$) is a measurement of how well the Gram matrix of $\mathbf{SA}$ approximates that of $\mathbf{A}$.

Note that in order for $\mathbf{S}$ to be a subspace embedding for L it is necessary that $d \geq \dim(L)$. Therefore if L is the range of an $m \times n$ matrix of full-column-rank, the requirement that $d \geq \dim(L)$ means that subspace embeddings can only be achieved when "sketching in the embedding regime," in the sense of Section 2.2.1. Furthermore, substantial dimension reduction can only be achieved in this framework when $m \gg n$.

Effective distortion

Subspace embedding distortion is the most common measure of sketch quality, but it is not without its limitations. Its greatest limitation is that it is not invariant under scaling of $\mathbf{S}$ (i.e., it is not invariant under replacing $\mathbf{S} \leftarrow t\mathbf{S}$ for $t \neq 0$). This is a significant limitation since many RandNLA algorithms *are* invariant under scaling of $\mathbf{S}$; existing theoretical analyses of RandNLA algorithms simply do not take this into account.

In Appendix A.1 we explore a novel concept of *effective distortion* that resolves the scale-sensitivity problem. Formally, the effective distortion of a sketching operator $\mathbf{S}$ for a subspace L is

$$\begin{aligned}\mathscr{D}_e(\mathbf{S}; L) = \inf\{\, \delta \,:\, & 0 \leq \delta \leq 1,\ 0 < t \\ & t\mathbf{S} \text{ is a } \delta\text{-embedding for } L\}.\end{aligned} \tag{2.3}$$

In words, this is the minimum distortion that *any* sketching operator $t\mathbf{S}$ can achieve for L, optimizing over $t > 0$. We briefly reference this concept in our discussion of algorithms for least squares and optimization (§3.2). Appendix B.1 makes deeper connections between effective distortion and randomized preconditioning methods for least squares.

Oblivious subspace embeddings

Data-oblivious subspace embedding (OSEs) were first used in RandNLA in [Sar06] and were largely popularized by [Woo14]. There is a clean way to describe the "reliability" of a sketching distribution in this setting.

> Consider a distribution $\mathcal{D}$ over wide $d \times m$ matrices. We say that $\mathcal{D}$ has *the OSE property with parameters* (δ, n, p) if, for every n-dimensional subspace $L \subset \mathbb{R}^m$, we have
>
> $$\Pr\{\mathbf{S} \sim \mathcal{D} \text{ is a } \delta\text{-embedding for } L\} \geq 1 - p.$$

Theoretical analyses of sketching distributions often concern bounding d as a function of (δ, n, p) to ensure that $\mathcal{D}$ satisfies the OSE property. Naturally, all else equal, we would like to achieve the OSE property for smaller values of d.

Theoretical results can be used to select d in practice for very well-behaved distributions, particularly the Gaussian distribution. Results for the more sophisticated distributions (such as those of sparse sketching operators) tend to be pessimistic compared to what is observed in practice. Some of this pessimism stems from the existence of esoteric constructions which indeed call for large embedding dimensions. Setting these constructions aside, we have reason to be optimistic since distortion is actually not the ideal measure of sketch quality in many settings. Indeed, *effective distortion* is far more relevant for least squares and optimization, and it will always be no larger than the standard notion of distortion.

All in all, there is something of an art to choosing the best sketching distribution for a particular RandNLA task. Luckily, for most RandNLA algorithms it is far from necessary to choose the "best" sketching distribution; good results can be obtained even when setting distribution parameters by simple rules of thumb.

Johnson–Lindenstrauss embeddings

Let $\mathbf{S}$ be a $d \times m$ sketching operator and L be a finite point set in $\mathbb{R}^m$. We say that $\mathbf{S}$ is a *Johnson–Lindenstrauss embedding* (or "JL embedding") for L with distortion δ if, for all distinct $\boldsymbol{x}, \boldsymbol{y}$ in L, we have

$$1 - \delta \leq \frac{\|\mathbf{S}(\boldsymbol{x} - \boldsymbol{y})\|_2^2}{\|\boldsymbol{x} - \boldsymbol{y}\|_2^2} \leq 1 + \delta.$$

This property is named for a seminal result by William Johnson and Joram Lindenstrauss, who used randomization to prove the existence of operators satisfying this property where d is logarithmic in $|L|$ and linear in $1/\delta^2$ [JL84].

The JL Lemma (as the result is now known) is remarkable for two reasons. First, the requisite value for d did not depend on the ambient dimension m and was only logarithmic in $|L|$. Second, the construction of the transformation $\mathbf{S}$ was *data-oblivious* – a scaled orthogonal projection. This latter fact led to questions about how one might define alternative distributions over sketching operators, with the aim of

1. being simpler to implement than a scaled orthogonal projection, and
2. attaining similar "data-oblivious JL properties."

It so happened that *many* constructions could achieve these goals. For example, [IM98] and [DG03] relaxed the condition of being a (scaled) orthogonal projector to $\mathbf{S}$ having iid Gaussian entries, which still results in a rotationally-invariant distribution. As another example, [Ach03] relaxed the rotational invariance by choosing the entries of $\mathbf{S}$ to be scaled Rademacher random variables.

2.2.3 (In)essential properties of sketching distributions

Distributions $\mathcal{D}$ over wide sketching operators are typically designed so that, for $\mathbf{S} \sim \mathcal{D}$, the mean and covariance matrices are

$$\mathbb{E}[\mathbf{S}] = \mathbf{0} \qquad \text{and} \qquad \mathbb{E}[\mathbf{S}^*\mathbf{S}] = \mathbf{I}.$$

The property that $\mathbb{E}[\mathbf{S}] = \mathbf{0}$ is important – if not ubiquitous – in RandNLA. However, there is some flexibility in the latter property, as in most situations it suffices for the covariance matrix to be a scalar multiple of the identity.

To understand why we have flexibility in the scale of the covariance matrix, consider how $\mathbb{E}[\mathbf{S}^*\mathbf{S}] = \mathbf{I}$ is equivalent to $\mathbf{S}$ preserving squared Euclidean norms in expectation. As it happens, the vast majority of algorithms mentioned in this monograph do not need sketching operators to preserve norms. Rather, they rely on sketching preserving *relative norms*, in the sense that $\|\mathbf{S}\boldsymbol{u}\|_2/\|\mathbf{S}\boldsymbol{v}\|_2$ should be close to $\|\boldsymbol{u}\|_2/\|\boldsymbol{v}\|_2$ for all vectors $\boldsymbol{u}, \boldsymbol{v}$ in a set of interest. Such a property is clearly unaffected if every entry of $\mathbf{S}$ scaled by a fixed nonzero constant (i.e., if $\mathbf{S}$ is replaced by $t\mathbf{S}$ for some $t \neq 0$).

This section uses scale-agnosticism to help describe sketching distributions with reduced emphasis on whether the operator is wide or tall. For example, if the entries of $\mathbf{S}$ are iid mean-zero random variables of finite variance, then both $\mathbb{E}[\mathbf{S}^*\mathbf{S}]$ and $\mathbb{E}[\mathbf{S}\mathbf{S}^*]$ are scalar multiples of the identity matrix. Speaking loosely, the former property justifies using $\mathbf{S}$ to sketch from the left and the latter property justifies using $\mathbf{S}^*$ to sketch from the right.

With this observation in mind, this section ignores most matters of scaling that is applied equally to all entries of a sketching operator. This manifests in how we regularly describe sketching operators as having entries in $[-1, 1]$ even though it is more common to have entries in $[-v, v]$ for some positive v (which is set to achieve an identity covariance matrix). Note that this *does not* confer freedom to scale individual rows, columns, or entries of a sketching operator separately from one another.

The main places where scaling matters are in algorithms for norm estimation (see Section 4.3.5) and algorithms which only sketch a portion of the data in a larger problem. The subtleties in this latter situation warrant a detailed explanation.

Scale sensitivity: partial sketching

Let $\mathbf{G}$ be an $n \times n$ psd matrix and $\mathbf{A}$ be a very tall $m \times n$ matrix. Suppose that we approximate

$$\mathbf{H} = \mathbf{A}^*\mathbf{A} + \mathbf{G}$$

by a *partial sketch*

$$\mathbf{H}_{\text{sk}} = (\mathbf{S}_o\mathbf{A})^*(\mathbf{S}_o\mathbf{A}) + \mathbf{G}$$

where $\mathbf{S}_o$ is a $d \times m$ sketching operator. How should we understand the statistical properties of $\mathbf{H}_{\text{sk}}$ as an estimator for $\mathbf{H}$?

At the simplest level we can turn to the idea of subspace embedding distortion. Using the characterization of distortion in (2.2), we could study the distribution of the minimum $\delta \in (0,1)$ for which

$$(1-\delta)^2\mathbf{H} \preceq \mathbf{H}_{\text{sk}} \preceq (1+\delta)^2\mathbf{H}.$$

One can go beyond distortion by lifting to a higher-dimensional space. Letting $\sqrt{\mathbf{G}}$ denote the Hermitian square root of $\mathbf{G}$, we define the augmented sketching operator and augmented data matrix

$$\mathbf{S} = \begin{bmatrix} \mathbf{S}_o & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \qquad \text{and} \qquad \mathbf{A}_{\mathbf{G}} = \begin{bmatrix} \mathbf{A} \\ \sqrt{\mathbf{G}} \end{bmatrix}$$

This lets us express $\mathbf{H} = \mathbf{A}_{\mathbf{G}}^*\mathbf{A}_{\mathbf{G}}$ and $\mathbf{H}_{\text{sk}} = (\mathbf{S}\mathbf{A}_{\mathbf{G}})^*(\mathbf{S}\mathbf{A}_{\mathbf{G}})$. Therefore the statistical properties of $\mathbf{H}_{\text{sk}}$ as an approximation to $\mathbf{H}$ can be understood in terms of how $\mathbf{S}$ preserves (or distorts) the range of $\mathbf{A}_{\mathbf{G}}$.

Scale sensitivity: row sampling from block matrices

The concept of partial sketching can arise when sketching block matrices, which are indeed encountered in many applications. For example, it is widely appreciated that a ridge regression problem with tall $m \times n$ data matrix $\mathbf{A}$ and regularization parameter μ can be lifted to an ordinary least squares problem with data matrix $\mathbf{A}_\mu := [\mathbf{A}; \sqrt{\mu}\mathbf{I}]$.

Suppose we want to sketch $\mathbf{A}_\mu$ by a row sampling operator $\mathbf{S}$. It is natural to treat the lower n rows of $\mathbf{A}_\mu$ differently than its upper m rows. In particular, it is natural for $\mathbf{S}$ to be an operator that produces $\mathbf{S}\mathbf{A} = [\mathbf{S}_o\mathbf{A}; \sqrt{\mu}\mathbf{I}]$ with some other $d \times m$ row sampling operator $\mathbf{S}_o$. Here, even if $\mathbf{S}_o$ sampled rows from $\mathbf{A}$ uniformly at random, the map $\mathbf{A}_\mu \mapsto \mathbf{S}\mathbf{A}_\mu$ would *not* sample uniformly at random from $\mathbf{A}_\mu$. Therefore there is a sense in which partial sketching is a way of incorporating non-uniform row sampling into other sketching distributions; see [DKM06a] for the origins of this interpretation. In the context of this specific example, the nonuniformity would necessitate that $\mathbf{S}_o$ be scaled to have entries in $\{0, \pm 1/\sqrt{d}\}$. We refer the reader to Section 2.4.2 and Section 6.1.1 for more discussion on sketching operators that implement row sampling.

2.3 Dense sketching operators

The RandBLAS should provide methods for sampling sketching operators with iid entries drawn from distinguished distributions. Across this broad category, we believe the following types of operators stand out:

- *Rademacher sketching operators*: entries are ± 1 with equal probability;
- *uniform sketching operators*: entries are uniform over $[-1, 1]$;
- *Gaussian sketching operators*: entries follow the standard normal distribution.

We believe the RandBLAS should also support sampling row-orthonormal or column-orthonormal matrices uniformly at random from the set of all such matrices.

The theoretical results for Gaussian operators are especially strong. However, there is little practical difference in the performance of RandNLA algorithms between any of the three entrywise iid operators given above. This is reflected in implementations such as [LLS+17] that only use uniform sketching operators. The practical equivalence between these types of sketching operators also has theoretical support through *universality principles* in high-dimensional probability [Ver18], [OT17], [MT20, §8.8], [DLL+20]. In what follows we speak to implementation details and the intended use cases for these operators.

Sampling iid-dense sketching operators

Sampling from the Rademacher or uniform distributions is the most basic operation of random number generators. Methods for sampling from the Gaussian distribution involve transforming random variables sampled uniformly from $[0, 1]$. There are two transformations of interest for the RandBLAS: Box-Muller [BM58]; and the Ziggurat transform [MT00]. The former should be included in the RandBLAS because it is easy to implement and parallelizes well. The latter method is far more efficient on a single thread, and it has been used within RandNLA (see [MSM14]), but it does not parallelize well [Ngu07, §37.2.3]. We postpone any recommendation for whether it should be an option in the RandBLAS.

RandNLA algorithms tend to be very robust to the quality of the random number generator. As a result, it is not necessary for us to sample from the Gaussian distribution with high statistical accuracy. This is due in part to the aforementioned universality principles, and it can be seen through the success of *sub-Gaussian distributions* as an analysis framework in high-dimensional probability [Ver18, §2]. From an implementation standpoint, there is likely no need to sample from the Gaussian distribution beyond single precision [Mar22a]. It is worth exploring if even lower precisions (e.g., half-precision) would suffice for practical purposes.

Applying iid-dense sketching operators

If a dense sketching operator is realized explicitly in memory then it can (and should) be applied by an appropriate BLAS function, most likely `gemm`. Many RandNLA algorithms provide good practical performance even with such simple implementations, although there is potential for reduced memory or communication requirements if a sketching operator is applied without ever fully allocating it in-memory. There is a large design space for such algorithms with iid-dense sketching operators when using counter-based random number generators (see Section 2.1.1). Such functionality could appear in an initial version of a RandBLAS standard. The reference implementations of such functions could start as mere wrappers around routines to generate a sketching operator and then apply that operator via `gemm`.

Sampling and applying Haar operators

If we suppose left-sketching, then the Haar distribution is the uniform distribution over row-orthonormal matrices. If we instead suppose right-sketching, then it is the uniform distribution over column-orthonormal matrices. We call these operators "dense" because if one is sampled and then formed explicitly, it will be dense with probability one.

There are two qualitative approaches to sampling from this distribution. The naive approach essentially requires sampling from a Gaussian distribution and performing a QR factorization, at a total cost of $O(d^2m)$; see [Li92, §1 - §4] and more general methods in [Mez07]. A more efficient approach – which costs only $O(dm)$ time – involves constructing the operator as a composition of suitable Householder reflectors [Ste80]. This approach has the secondary benefit of not needing to form the sketching operator explicitly.

Haar operators are of interest not just for sketching in RandNLA algorithms but also for generating test data for evaluating other sketching operators. As such, we believe they are natural to include in a first version of a RandBLAS standard.

Intended use-cases

Using terminology from Section 2.2.1, dense sketching operators are commonly used for "sketching in the sampling regime." In particular, they are the workhorses of randomized algorithms for low-rank approximation. They also have applications in certain randomized algorithms for ridge regression and some full-rank matrix decomposition problems.

These distributions are much less useful for sketching dense matrices "in the embedding regime" (again in the sense of Section 2.2.1). This is because they are more expensive to apply to dense matrices than many other types of sketching operators. These types of sketching operators *might* be of interest in the embedding regime if applied to sparse or otherwise structured data matrices.

2.4 Sparse sketching operators

The RandNLA literature describes many types of sparse sketching operators, almost always under the convention of sketching from the left. We think it is important to define sketching distributions in a way that is agnostic to sketching from the left or right. Indeed, while we often focus on left-sketching for ease of exposition, asserting that this is "without loss of generality" ignores the plight of the user tasked with right-sketching.

In order to achieve our desired agnosticism, we use a taxonomy for sparse sketching operators which has not appeared in prior literature. To describe it, we use the term *short-axis vector* in reference to the columns of a wide matrix or rows of a tall matrix. The term *long-axis vector* is defined analogously, as the rows of a wide matrix or columns of a tall matrix. In these terms, we have the following families of sparse sketching operators.

- *Short-axis-sparse* sketching operators. The short-axis vectors of these operators are independent of one another. Each short-axis vector has a fixed (and very small) number of nonzeros. Typically, the indices of the nonzeros in each short-axis vector are sampled uniformly without replacement.

- *Long-axis-sparse* sketching operators. The long-axis vectors of these operators are independent of one another. For a given long-axis vector, the indices for its nonzeros are sampled with replacement according to a prescribed probability distribution (which can be uniform). The value of a given nonzero is affected by the number of times its index appears in the sample for that vector.

- *Iid-sparse* sketching operators. Mathematically, these can be described as starting with an iid-dense sketching operator and "zeroing-out" entries in an iid-manner with some high probability. (From an implementation standpoint this would work the other way around, randomly choosing a few entries to make nonzero.)

When abbreviations are necessary, we suggest that short-axis-sparse sketching operators be called *SASOs* and that long-axis-sparse sketching operators be called *LASOs*. Most of our use of such abbreviations appears here and in Appendix A.2. A visualization of these types of sketching operators is given in Figure 2.2.

Before proceeding further we should say that we are *not* in favor of including iid-sparse sketching operators in the RandBLAS. Our first reason for this is that their theoretical guarantees are not as strong as either SASOs (see the discussion at the end of [Tro20, §7.4] and remarks in [Lib09, §2.4]) or LASOs [DLD+21; DLP+21]. Our second reason is that their lack of predictable structure makes it harder to implement efficient parallel algorithms for applying these operators. Therefore in what follows we only give details on SASOs and LASOs.

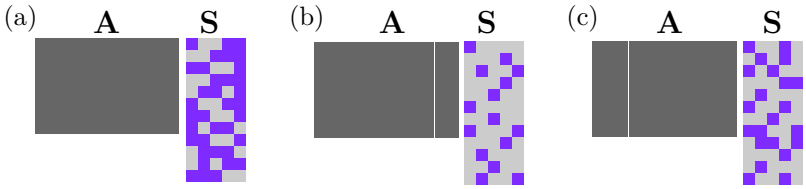

Figure 2.2: Illustration of a SASO (a) with 3 non-zero entries per row, LASO (b) with 3 non-zero entries per column, and an iid-sparse sketching operator (c) with iid non-zero entries.

2.4.1 Short-axis-sparse sketching operators

SASOs include sketching operators known as sparse Johnson–Lindenstrauss transforms, the Clarkson–Woodruff transform, CountSketch, and OSNAPs [KN12; CW13; MM13; NN13]. These constructions are all described assuming we sketch from the left, and as such, they are all stated for *wide* sketching operators. They are described as having a fixed number of nonzeros within each column. The more general notion (for sketching from the left or right) is to say there is a fixed number of nonzeros per short-axis vector.

The short-axis vectors of a SASO should be independent of one another. One can select the locations of nonzero elements in different ways; we are interested in two methods from [KN12]. For a wide $d \times m$ operator, we can

1. sample k indices uniformly from $[\![d]\!]$ without replacement, once for each column, or

2. divide $[\![d]\!]$ into k contiguous subsets of equal size, and then for each column we select one index from each of the k index sets.

These definitions are extended from wide sketching operators to tall sketching operators in the natural way.

For either method, the nonzero values in a SASO's short-axis vector are canonically independent Rademachers. Alternatively, they can be drawn from other sub-Gaussian distributions. For example, in the wide case, drawing the nonzeros independently and uniformly from a union of disjoint intervals, such as $[-2,-1]\cup[1,2]$, can protect against the possibility of a given row of $\mathbf{S}$ being orthogonal to a column of a matrix to be sketched [Tyg22].

Details SASOs are provided in Appendix A.2. This includes implementation notes, a short historical summary of relevant theory, and remarks on setting the sparsity parameter k. On the topic of theory, we note here that the state-of-the-art results for SASOs are due to Cohen [Coh16]. More information can be found in the lecture notes [Mah11; Mah16; DM18], [Tro20, §7.4] and the surveys [Woo14], [MT20, §9.2].

Remark 2.4.1 (Naming conventions). The concept of what we would call a "wide SASO" is referred to in the literature as an *OSNAP*. We have a slight preference for "SASO" over "OSNAP" for two reasons. First, it pairs naturally with the abbreviation *LASO* for long-axis-sparse sketching operators, which is valuable for taxonomizing sparse sketching operators. Second, the literature consistently describes OSNAPs as having a fixed number of nonzeros per column. While this description is appropriate for left sketching, it is not appropriate for right sketching.

2.4.2 Long-axis-sparse sketching operators

This category includes row and column sampling, LESS embeddings [DLD+21], and LESS-uniform operators [DLP+21].

In the wide case, a LASO has independent rows and a fixed upper bound on the number of nonzeros per row. All rows are sampled with reference to a distribution $\boldsymbol{p}$ over $[\![m]\!]$ (which can be uniform) and a positive integer k. Construction begins by sampling $t_1,\dots,t_k$ from $[\![m]\!]$ with replacement according to $\boldsymbol{p}$. Then we initialize

$$\mathbf{S}[i,:] = \frac{1}{\sqrt{dk}}\left(\sqrt{\frac{b_1}{p_1}},\dots,\sqrt{\frac{b_m}{p_m}}\right), \tag{2.4}$$

where b_j is the number of times the index j appeared in the sample $(t_1,\dots,t_k)$. We finish constructing the row by multiplying each nonzero entry by an iid copy of a mean-zero random variable of unit variance (e.g., a standard Gaussian random variable). Such a LASO will have at most k nonzeros per row and hence at most dk nonzeros in total. Note that this is much smaller than mk nonzeros required by a SASO with the same parameters.

The quality of sketches produced by LASOs when $\boldsymbol{p}$ is uniform depends on the properties of the matrix to be sketched. Specifically, it will depend on the *leverage scores* of the matrix. The leverage score concept, introduced in Section 6.1, is important for constructing data-aware sketching operators that implement row or column sampling. If $\boldsymbol{p}$ is the leverage score distribution of some matrix then the sketching operator is known as a Leverage Score Sparsified (LESS) embedding for that matrix [DLD+21]. The term *LESS-uniform* has been used for long-axis-sparse operators that use the uniform distribution for $\boldsymbol{p}$ [DLP+21].

Remark 2.4.2 (Scale). The scaling factor $1/\sqrt{dk}$ appearing in the initialization (2.4) is the same for all rows of $\mathbf{S}$ (in the wide case, i.e., for each long-axis vector). This factor is necessary so that once the nonzeros in $\mathbf{S}$ are multiplied by mean-zero unit-variance random variables, we have $\mathbb{E}[\mathbf{S}^*\mathbf{S}] = \mathbf{I}_m$. This scaling matters when one

cares about subspace embedding distortion or when one is only sketching a portion of the problem data (see Section 2.2.3). This scaling has no effect on effective distortion if $\boldsymbol{p}$ is uniform.

2.5 Subsampled fast trigonometric transforms

Fast trigonometric transforms (or *fast trig transforms*) are orthogonal or unitary operators that take m-vectors to m-vectors in $O(m \log m)$ time or better. The most important examples in this class are the Discrete Fourier Transform (for complex-valued inputs) and the Discrete Cosine Transform (for real-valued inputs). The Walsh-Hadamard Transform is also notable; although it only exists when m is a power of two, it is equivalent to a Kronecker product of $\log_2 m$ Discrete Fourier Transforms of size 2×2, and the standard algorithm for applying it involves no multiplications and entails no branching.

Traditionally, trig transforms are valued for their ability to map dense input vectors with a periodic structure into sparse output vectors. Within RandNLA, we are interested in them for the opposite reason: the fact that they map inputs that lack periodic structure to *dense* outputs. This behavior is useful because if we preprocess an input to destroy any periodic structure with high probability, the resulting output should be easier to approximate by random coordinate subsampling. This leads to the idea of a *subsampled randomized fast trig transforms* or *SRFTs*.

Traditional SRFTs

Formally, a $d \times m$ SRFT takes the form

$$\mathbf{S} = \sqrt{m/d}\, \mathbf{RFD},$$

where $\mathbf{D}$ is a diagonal matrix of independent Rademachers, $\mathbf{F}$ is a fast trig transform that maps m-vectors to m-vectors, and $\mathbf{R}$ randomly samples d components from an m-vector [AC06; AC09]. For added robustness one can define SRFTs slightly differently, replacing $\mathbf{S}$ by $\mathbf{S\Pi}$ for a permutation matrix $\mathbf{\Pi}$ [MT20].

SRFTs are appealing for their efficiency and theoretical guarantees. Speaking to the former aspect, a $d \times m$ SRFT can be applied to an $m \times n$ matrix in as little as $O(mn \log d)$ time by using methods for subsampled fast trig transforms [WLR+08, §3.3], [Lib09, §3.3]. Theoretical guarantees for SRFTs are usually established assuming $\mathbf{F}$ is the Walsh-Hadamard transform [DMM+11; Tro11; BG13]. These guarantees are especially appealing since they do not rely on tuning parameters such as sparsity parameters required by sketching operators from Section 2.4.

The trouble with SRFTs is that they are notoriously difficult to implement efficiently. Even their best-case $O(mn \log d)$ complexity is higher than the $O(mnk)$ complexity of a SASO that is wide with $k \ll \log d$ nonzeros per column. However, SRFTs have an advantage when it comes to memory: if one overwrites $\mathbf{A}$ by the $m \times n$ matrix $\mathbf{B} := \sqrt{m/d}\,\mathbf{FDA}$ in $O(mn \log m)$ time, then $\mathbf{SA}$ can be accessed as a submatrix of rows of $\mathbf{B}$ without losing access to $\mathbf{A}$ or $\mathbf{A}^*$ as linear operators. Further investigation is needed to determine the true value of this in-place nondestructive implementation. For the time being, we do not believe that traditional SRFTs are essential for a preliminary RandBLAS standard.

Block SRFTs

Let p be a positive integer, $r = m/p$ be greater than d, and $\mathbf{R}$ be a matrix that randomly samples d components from an r-vector. For each index $i \in [\![p]\!]$, we introduce a $d \times r$ sketching operator

$$\mathbf{S}_i = \sqrt{r/d}\, \mathbf{D}_i^{\text{post}} \mathbf{R} \mathbf{F} \mathbf{D}_i^{\text{pre}},$$

where $\mathbf{D}_i^{\text{post}}$ and $\mathbf{D}_i^{\text{pre}}$ are diagonal matrices filled with independent Rademachers. The *block SRFT* [BBG+22] is defined columnwise as $\mathbf{S} = [\mathbf{S}_1 \quad \mathbf{S}_2 \quad \dots \quad \mathbf{S}_p]$.

Block SRFTs can effectively leverage parallel hardware using *serial* implementations of the fast trig transform. For concreteness, suppose $\mathbf{A}$ is $m \times n$ and distributed block row-wise among p processors. We apply the block SRFT $\mathbf{S}$ by the formula

$$\mathbf{SA} = \sum_{i \in [\![p]\!]} \mathbf{S}_i \mathbf{A}_i,$$

where $\mathbf{A}_i$ is the block of rows of $\mathbf{A}$ stored on processor i. The multiplication $\mathbf{S}_i \mathbf{A}_i$, computed locally on each processor, is followed by a reduction operation among processors to sum the local contributions.

We are undecided as to whether block SRFTs are appropriate for a preliminary RandBLAS standard. Their comparative ease of implementation is favorable. However, they are problematic in that the definition of the distribution changes as we vary p, which complicates reproducibility across platforms.

Historical remarks and further reading

The development of SRFTs began with *fast Johnson–Lindenstrauss transforms* (FJLTs) [AC06], which replace the matrix "$\mathbf{R}$" in the SRFT construction by a particular type of sparse matrix. FJLTs were first used in RandNLA for least squares and low-rank approximation by [Sar06]. The jump from FJLTs to SRFTs was made independently in [DMM+11] and [WLR+08] for usage in least squares and low-rank approximation, respectively.

For more background on this topic we refer the reader to the book [Mah11], the lecture notes [Mah16], [DM18], [Tro20, §7.5], and the survey [MT20, §9.3]. We also note that SRFTs are sometimes called *randomized orthornomal systems* [PW16; PW17; OPA19] or (with slight abuse of terminology) FJLTs. Finally, we point out that a type of "SRFTs without subsampling" has been successfully used to approximate Gaussian matrices needed in random features approaches to kernel ridge regression [LSS13].

Remark 2.5.1 (Navigating the literature). The reader should be aware that [AC09] is the journal version of [AC06]. Additionally, while [LWM+07] also describes SRFTs, it was actually written after [WLR+08].

2.6 Multi-sketch and quadratic-sketch routines

For many years, the performance bottleneck in NLA algorithms has been data movement, rather than FLOPs performed on the data. For example, a general matrix-matrix multiply with $n \times n$ matrices would do $O(n^3)$ data movement if implemented naively with three nested loops, but can be up to a factor of $\sqrt{M}$ smaller, where M is the cache size, if appropriately implemented using loop tiling.

In our context of RandNLA, the fastest randomized algorithms for low-rank matrix approximation involve computing multiple sketches of a data matrix. Such *multi-sketching* presents new challenges and opportunities in the development of optimized implementations with minimal data movement.

We believe the RandBLAS should include functionality for at least three types of multi-sketching, listed below. The end of Section 4.2.1 points to algorithms that use these primitives. In all cases of which we are aware, these primitives are only used for sketching in the sampling regime.

1. Generate $\mathbf{S}$ and compute $\mathbf{Y}_1 = \mathbf{AS}$ and $\mathbf{Y}_2 = \mathbf{A}^*\mathbf{AS}$. We illustrate the use of this primitive explicitly in Algorithm 12.
2. Generate independent $\mathbf{S}_1, \mathbf{S}_2$, and compute $\mathbf{Y}_1 = \mathbf{AS}_1$ and $\mathbf{Y}_2 = \mathbf{S}_2\mathbf{A}$. Algorithms which use this primitive typically need to retain $\mathbf{S}_1$ or $\mathbf{S}_2$ for later use [HMT11, pg. 251], [YGL+17, Algorithm 2], [TYU+17b, § 1.4].
3. Generate independent $\mathbf{S}_1, \mathbf{S}_2, \mathbf{S}_3, \mathbf{S}_4$, and compute $\mathbf{Y}_1 = \mathbf{AS}_1$, $\mathbf{Y}_2 = \mathbf{S}_2\mathbf{A}$, and $\mathbf{Y}_3 = \mathbf{S}_3\mathbf{AS}_4$.

Having identified these operations as basic building blocks, we arrive at the following question.

> What combination of sketching distributions should be supported in multi-sketching of types 2 and 3?

For the former type (i.e., type 2), it is important to support at least the case that both $\mathbf{S}_1$ and $\mathbf{S}_2$ are dense sketching operators, and it may be useful to support when both are fast operators. At this point, we do not see an advantage for one of $(\mathbf{S}_1, \mathbf{S}_2)$ to be a fast operator and for the other to be dense. For the latter type (i.e., type 3), [TYU+17b, §7.3.2] suggests that $(\mathbf{S}_1, \mathbf{S}_2)$ be Gaussian and that $(\mathbf{S}_3, \mathbf{S}_4)$ be SRFTs. We believe that it would be reasonable to use SASOs in place of SRFTs in this context.

The RandBLAS should provide methods to compute sketches that are quadratic in the data matrix. By "quadratic sketch," we mean a linear sketch of $\mathbf{A}^*\mathbf{A}$ or $\mathbf{AA}^*$. This operation is ubiquitous in algorithms for low-rank approximation. As with multi-sketching, all uses of quadratic sketching (of which we are aware) entail sketching in the sampling regime. It is not possible to fundamentally accelerate this kind of sketching by using fast sketching operators.[2] Therefore it would be reasonable for RandBLAS's quadratic sketching methods to only support dense sketching operators. (The preceding comments also apply to type 1 multi-sketching.) In essence, this asks for a high-performance implementation of the composition of the BLAS 3 functions `syrk` and `gemm`: $(\mathbf{A}, \mathbf{S}) \mapsto \mathbf{AA}^*\mathbf{S}$. There is a substantial amount of structure in quadratic sketching that could be leveraged for reduced data movement, which suggests that the RandBLAS would benefit significantly from having optimized routines for this functionality.

[2] This point has also been made in a related setting [MT20, §11.6.1].

Section 3

Least Squares and Optimization

Numerical linear algebra is the backbone of the most widely-used algorithms for continuous optimization. Continuous optimization, in turn, is a workhorse for many scientific computing, machine learning, and data science applications.

The connections between optimization and linear algebra are often introduced with *least squares problems*. Such problems have been used as a tool for curve fitting since the days of Gauss and Legendre over 200 years ago — several decades before Cayley even defined linear algebraic concepts such as the matrix-inverse! These problems are also remarkable because algorithms for solving them easily generalize to more complicated settings. Indeed, one of this section's key messages is that, by adopting a suitable perspective, one can use randomization in essentially the same way to solve a wealth of different quadratic optimization problems.

Our perspective entails describing all least squares problems in terms of an $m \times n$ data matrix $\mathbf{A}$ with at least as many rows as columns. Specifically, we express the overdetermined problem as

$$\min_{\boldsymbol{x} \in \mathbb{R}^n} \|\mathbf{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2$$

for a vector $\boldsymbol{b}$ in $\mathbb{R}^m$, while we express the underdetermined problem as

$$\min_{\boldsymbol{y} \in \mathbb{R}^m} \{\|\boldsymbol{y}\|_2^2 \mid \mathbf{A}^*\boldsymbol{y} = \boldsymbol{c}\}$$

for a vector $\boldsymbol{c}$ in $\mathbb{R}^n$. Of course, both of these models could be expressed in the corresponding "argmin" formulation. We generally prefer the "min" formulation for the optimization problem itself and use "argmin" only for the set of optimal solutions.

Section 3.1 introduces the problems we consider: minimization of regularized quadratics and various generalizations of least squares problems. For each problem, it provides high-level comments on structures and desired outcomes that can make randomized algorithms preferable to classical ones.

Section 3.2 covers the drivers for these problems based on RandNLA. It details the problem structures that stand to benefit from a particular driver, and it highlights other linear algebra problems that largely reduce to solving problems amenable to these drivers. Section 3.3 details some essential computational routines that would power the drivers.

The rest of the Section 3 is largely supplemental. Section 3.4 reviews randomized optimization algorithms that we find notable but out-of-scope, as well as one type of deterministic computational routine that is potentially useful for (but not required by) the drivers. We conclude by describing existing RandNLA libraries for least squares and optimization in Section 3.5.

3.1 Problem classes

This section covers drivers for two related classes of optimization problems: minimizing regularized positive definite quadratics (§3.1.1) and certain generalizations of overdetermined and underdetermined least squares which we refer to as *saddle point problems* (§3.1.2). Problems in both classes can naturally be transformed to equivalent linear algebra problems.[1] Functionality for solving these problems can easily provide the foundation for managing the core linear algebra kernels of larger optimization algorithms.

How can we measure the accuracy of an approximate solution?

The problem of quantifying the error of an approximate solution to a least squares or saddle point problem is very important. While we would like to address this topic up-front, a proper discussion requires technical background that we only cover in Section 3.3. Furthermore, even given this background, there are various subtleties and special cases that would be laborious to describe here. Therefore we defer the important topic of error metrics for least squares and related problems to Appendix B.2.

[1] As we explain later, the saddle point problems are equivalent to so-called *saddle point systems*, which are well-studied in the NLA literature; see [BGL05; OA17]

3.1.1 Minimizing regularized quadratics

Let $\mathbf{G}$ be a positive semidefinite (psd) linear operator, and let μ be a positive regularization parameter. One of the main topics of this section is algorithms for computing approximate solutions to problems of the form

$$\min_{\boldsymbol{x}} \boldsymbol{x}^* \left(\mathbf{G} + \mu \mathbf{I}\right) \boldsymbol{x} - 2\boldsymbol{h}^* \boldsymbol{x}. \tag{3.1}$$

Note that solving (3.1) is equivalent to solving $\left(\mathbf{G} + \mu \mathbf{I}\right) \boldsymbol{x} = \boldsymbol{h}$. We refer to such problems in different contexts throughout this section. In some contexts, we say that $\mathbf{G}$ is $n \times n$, and in others, we say it is $m \times m$.

This section covers algorithms for solving these problems to varying degrees of accuracy.

- Methods for solving to higher accuracy will access $\mathbf{G}$ repeatedly by matrix-matrix and matrix-vector multiplication.
- Methods for solving to lower accuracy may vary in how they access $\mathbf{G}$: they may only entail selecting a subset of its columns; or they may perform a single matrix-matrix multiplication $\mathbf{GS}$ with a tall and thin matrix $\mathbf{S}$.

Note that the low accuracy methods may be useful in machine learning contexts such as kernel ridge regression (KRR), where an inaccurate solution to (3.1) can still be useful for downstream computational tasks.

Remark 3.1.1. If the linear operator $\mathbf{G}$ implements the action of an implicit Gram matrix $\mathbf{A}^*\mathbf{A}$ (with $\mathbf{A}$ known) then it would be preferable to reformulate (3.1) as (3.2), below, with $\boldsymbol{b} = \mathbf{0}$ and $\boldsymbol{c} = \boldsymbol{h}$.

Amenable problem structures

The suitability of methods we describe for problem (3.1) will depend on how many eigenvalues of $\mathbf{G}$ are larger than μ. Supposing $\mathbf{G}$ is $n \times n$, it is desirable that the number of such eigenvalues is much less than n. The data $(\mathbf{G}, \mu)$ that arise in practical KRR problems usually have this property.

In an ideal setting, the user would have an estimate for the number of eigenvalues of $\mathbf{G}$ that are larger than μ. This is not a strong requirement when $\mathbf{G}$ is accessible by repeated matrix-vector multiplication, in which case the accuracy of the estimate is unimportant. The standard RandNLA algorithm in this situation can easily be modified to recycle the work in solving (3.1) for one value of μ towards solving (3.1) for another value of μ.

3.1.2 Solving least squares and basic saddle point problems

We are interested in certain generalizations of overdetermined and underdetermined least squares problems. The generalizations facilitate natural specification of linear terms in composite quadratic objectives, which is a common primitive in many second-order optimization algorithms.

We frame these problems as complementary formulations of a common *saddle point problem*. The defining data for such a problem consists of a tall $m \times n$ matrix $\mathbf{A}$, an m-vector $\boldsymbol{b}$, an n-vector $\boldsymbol{c}$, and a scalar $\mu \geq 0$. For simplicity, our descriptions in this paragraph assume $\mathbf{A}$ is full-rank. The *primal* saddle point problem is

$$\min_{\boldsymbol{x} \in \mathbb{R}^n} \|\mathbf{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 + \mu \|\boldsymbol{x}\|_2^2 + 2\boldsymbol{c}^* \boldsymbol{x}. \tag{3.2}$$

When μ is positive, the *dual* saddle point problem is

$$\min_{\boldsymbol{y}\in\mathbb{R}^m} \|\mathbf{A}^*\boldsymbol{y} - \boldsymbol{c}\|_2^2 + \mu\|\boldsymbol{y} - \boldsymbol{b}\|_2^2. \tag{3.3}$$

In the limit as μ tends to zero, Eq. (3.3) canonically becomes

$$\min_{\boldsymbol{y}\in\mathbb{R}^m} \{\|\boldsymbol{y} - \boldsymbol{b}\|_2^2 \,:\, \mathbf{A}^*\boldsymbol{y} = \boldsymbol{c}\}. \tag{3.4}$$

Note that the primal problem reduces to ridge regression when $\boldsymbol{c}$ is zero, and it reduces to overdetermined least squares when both $\boldsymbol{c}$ and μ are zero. When $\boldsymbol{b}$ is zero, and depending on the value of μ, the dual problem amounts to ridge regression with a wide data matrix or to basic underdetermined least squares.

Pros and cons of this viewpoint

Adopting this more general optimization-based viewpoint on least squares problems has two major benefits.

- It extends least squares problems to include linear terms in the objective. The linear term in the primal problem is obvious. The linear terms in (3.3) and (3.4) are obtained by expanding $\|\boldsymbol{y} - \boldsymbol{b}\|_2^2 = \|\boldsymbol{y}\|_2^2 - 2\boldsymbol{b}^*\boldsymbol{y} + \|\boldsymbol{b}\|_2^2$ and ignoring the constant term $\|\boldsymbol{b}\|_2^2$.
- It renders the primal and dual problems equivalent for most algorithmic purposes. The equivalence is based on formulating the optimality conditions for these problems in a so-called *saddle point system* over the variables $(\boldsymbol{x}, \boldsymbol{y})$. Section 3.3.1 details this equivalence.

It must be noted that the saddle point problems we consider can be ill-posed when μ is zero and $\mathbf{A}$ is rank-deficient. Specifically, when $\mu = 0$ and $\boldsymbol{c}$ is not orthogonal to the kernel of $\mathbf{A}$, the primal problem (3.2) has no optimal solution and the dual problem (3.4) has no feasible solution. In this setting, we assign *canonical solutions* by considering the limit as μ tends to zero. Appendix B.3 addresses the existence and form of these limiting solutions. The outcome of the limiting analysis is that when $\mu = 0$, we obtain canonical solutions

$$\boldsymbol{x} = (\mathbf{A}^*\mathbf{A})^\dagger(\mathbf{A}^*\boldsymbol{b} - \boldsymbol{c}) \quad \text{and} \quad \boldsymbol{y} = (\mathbf{A}^*)^\dagger\boldsymbol{c} + (\mathbf{I} - \mathbf{A}\mathbf{A}^\dagger)\boldsymbol{b}, \tag{3.5}$$

which are related through the identity $\boldsymbol{y} = \boldsymbol{b} - \mathbf{A}\boldsymbol{x}$.

Amenable problem structures

This section focuses on methods for solving these optimization problems to high accuracy. Indeed, later in this section we make the novel observation that methods for solving problems (3.2)–(3.4) to high accuracy can be used as the core subroutine in solving (3.1) to low accuracy at extremely large scales. If $m \gg n$, then these methods are efficient regardless of numerical aspects of the problem data $(\mathbf{A}, \boldsymbol{b}, \boldsymbol{c}, \mu)$; problems such as poor numerical conditioning will be relevant insofar as they contribute to floating-point rounding errors in these efficient algorithms. If m is only slightly larger than n, then the methods we describe will only be effective when $\mathbf{G} := \mathbf{A}^*\mathbf{A}$ and μ have the properties alluded to in Section 3.1.1. These properties are detailed later in this section.

3.2 Drivers

Here we present four families of drivers for the problems described in Section 3.1. Two of the driver families belong to a paradigm in the RandNLA literature known as *sketch-and-precondition*. Algorithms in these families are capable of computing accurate approximations of a problem's true solution. The other two driver families belong to a paradigm known as *sketch-and-solve*. They are less expensive than sketch-and-precondition methods (to varying degrees) but they are only suitable for producing rough approximations of a problem's true solution. The sketch-and-solve drivers described in Section 3.2.4 are novel in that they rely on separate sketch-and-precondition methods for their core subroutines.

3.2.1 Sketch-and-solve for overdetermined least squares

Sketch-and-solve is a broad paradigm within RandNLA, and algorithms based on it have been central to early developments in the area [Mah11; Woo14; DM16; DM21a]. Its most notable manifestations have been for overdetermined least squares [DMM06; Sar06; DMM+11; CW13], overdetermined ℓ_1 and general ℓ_p regression [DDH+09; YMM16], and ridge regression [ACW17a; WGM18].

We focus here on least squares for concreteness. In this case, one samples a sketching operator $\mathbf{S}$, and returns

$$(\mathbf{SA})^\dagger(\mathbf{S}\boldsymbol{b}) \in \arg\min_{\boldsymbol{x}} \|\mathbf{S}(\mathbf{A}\boldsymbol{x} - \boldsymbol{b})\|_2^2 \tag{3.6}$$

as a proxy for the solution to $\min_{\boldsymbol{x}} \|\mathbf{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2$. The quality of this solution can be bounded with the concept of subspace embeddings from Section 2.2.2. In particular, if $\mathbf{S}$ is a subspace embedding for $V = \mathrm{range}([\mathbf{A}, \boldsymbol{b}])$ with distortion δ, then

$$\|\mathbf{A}(\mathbf{SA})^\dagger(\mathbf{S}\boldsymbol{b}) - \boldsymbol{b}\|_2 \leq \left(\frac{1+\delta}{1-\delta}\right) \|\mathbf{A}\mathbf{A}^\dagger\boldsymbol{b} - \boldsymbol{b}\|_2. \tag{3.7}$$

Note that (3.6) is invariant under scaling of $\mathbf{S}$. This implies that (3.7) also holds when δ is the effective distortion of $\mathbf{S}$ for V; see (2.3) and Appendix A.1.

Implementation considerations and a viable application are given below.

Methods for the sketched subproblem

Direct methods for (3.6) require computing an orthogonal decomposition of $\mathbf{SA}$, such as a QR decomposition or an SVD, in $O(dn^2)$ time. In this context, sketch-and-solve can be used as a preprocessing step for sketch-and-precondition methods at essentially no added cost. Indeed, this preprocessing step was used in [RT08]. Therefore if a direct method is being considered for sketch-and-solve, then sketch-and-precondition methods should also be viable when $m \in O(dn)$.

One can in principle apply an iterative solver to the problem defined by $(\mathbf{SA}, \mathbf{S}\boldsymbol{b})$. This strategy avoids the cost of factoring $\mathbf{SA}$, and it reduces the per iteration cost relative to running the iterative solver on the original problem. This is typically implemented without preconditioning (but see [YCR+18]), in which case it leaves the dependence on the condition number of the original problem, and so it can only be recommended for problems where the condition number is known to be small.

Error estimation

Since sketch-and-solve algorithms for overdetermined least squares are most suitable for computing rough approximations to a problem's true solution, it is important to have methods for estimating this error. Such estimates can either be used to inform downstream processing of the approximate solution or to determine if a more accurate solution (computed by a more expensive algorithm) might be needed. It is especially important that these methods work well in regimes where sketch-and-solve has a compelling computational profile, such as when $m \gg dn$. Appendix E.2 provides one such estimator based on the principle of *bootstrapping* from statistics.

Application to tensor decomposition

The benefits of sketch-and-solve for least squares manifest most prominently when the following conditions are satisfied simultaneously: (1) m is extremely large, so $\mathbf{A}$ is not stored explicitly, and (2) $\mathbf{A}$ supports relatively cheap access to individual rows $\mathbf{A}[i,:]$. Among other places, this situation arises in alternating least squares approaches to tensor decomposition. We touch upon that topic in Section 7.3.1, particularly in the remarks after (7.17).

3.2.2 Sketch-and-precondition for least squares and saddle point problems

The *sketch-and-precondition* approach to overdetermined least squares was introduced by Rokhlin and Tygert [RT08]. When the $m \times n$ matrix $\mathbf{A}$ is very tall, the method is capable of producing accurate solutions with less expense than direct methods. It starts by computing a $d \times n$ sketch $\mathbf{A}^{\text{sk}} = \mathbf{SA}$ in the embedding regime (i.e., $d \gtrsim n$). The sketch is decomposed by QR with column pivoting $\mathbf{A}^{\text{sk}}\mathbf{\Pi} = \mathbf{QR}$, which defines a preconditioner $\mathbf{M} = \mathbf{\Pi R}^{-1}$. If the parameters for the sketching operator distribution were chosen appropriately, then $\mathbf{AM}$ will be nearly-orthogonal with high probability.[2] The near-orthogonality of $\mathbf{AM}$ ensures rapid convergence of an iterative method for the least squares problem's preconditioned normal equations. If T_{sk} denotes the time complexity of computing $\mathbf{SA}$, then the typical asymptotic FLOP count to solve to ϵ-error is

$$O(T_{\text{sk}} + dn^2 + mn \log(1/\epsilon)) \tag{3.8}$$

Importantly, this complexity has no dependence on the condition number of $\mathbf{A}$.

This approach was extended with stronger theoretical guarantees, support for more general least squares problems, and high-performance implementations through *Blendenpik* [AMT10] and *LSRN* [MSM14]. It has also recently been used to solve positive definite systems arising in linear programming algorithms [CLA+20]. All of these methods produce preconditioners $\mathbf{M}$ where $\mathbf{AM}$ is nearly-orthogonal and are intended for the regime where $\mathbf{A}$ is very tall.

What constitutes "very tall" depends on the algorithm's implementation and the hardware that runs it. It is easy to implement these algorithms in Matlab or Python so that, on a personal laptop, they are competitive with LAPACK's direct methods when $m \geq 50n \geq 10^5$; see also Section 3.5.

[2] The condition number of $\mathbf{AM}$ and the effective distortion of $\mathbf{S}$ for range($\mathbf{A}$) completely characterize one another; see Appendices A.1 and B.1.

Your attention, please!

If a saddle point problem features regularization (i.e., if $\mu > 0$) and if $\mathbf{A}$ has rapid spectral decay, then randomized methods can be used to find a good preconditioner in far less than $O(n^3)$ time, no matter the specific value of $m \geq n$. This is possible by borrowing ideas from *Nyström preconditioning* [FTU21], which we introduce in Section 3.2.3 for the related problem of minimizing regularized quadratics. As a novel contribution, Section 3.3.3 explains how Nyström preconditioning can naturally be adapted to saddle point problems. Therefore while the material here (in Section 3.2.2) focuses on the case $m \gg n$, one should be aware that this requirement can be relaxed.

Algorithms

Sketch-and-precondition algorithms can take different approaches to sketching, preconditioner generation, and choice of the eventual iterative solver.

- Blendenpik used SRFT sketching operators, obtained its preconditioner by unpivoted QR of $\mathbf{A}^{\text{sk}}$, and used LSQR [PS82] as its underlying iterative method.
- LSRN used Gaussian sketching operators, obtained its preconditioner through an SVD of $\mathbf{A}^{\text{sk}}$, and defaulted to the Chebyshev semi-iterative method [GV61] for its iterative solver.

These two examples hint at the huge range of possibilities for the implementation of sketch-and-precondition algorithms. Indeed, we discuss preconditioners in detail over Sections 3.3.2 and 3.3.3, and we review a suite of possible deterministic iterative methods in Section 3.3.4. For now, we give Algorithms 1 and 2 (below) as footholds for understanding the various design considerations.

For simplicity's sake, both of these algorithms use a black-box function

$$\boldsymbol{z} = \texttt{iterative\_ls\_solver}(\mathbf{F}, \boldsymbol{g}, \epsilon, L, \boldsymbol{z}_o)$$

which computes an approximate solution to $\min_{\boldsymbol{z}} \|\mathbf{F}\boldsymbol{z} - \boldsymbol{g}\|_2^2$. The exact semantics of this function are unimportant for our present purpose. Its general semantics are that the solver initializes an iterative procedure at $\boldsymbol{z}_o$ and that it runs until either an implementation-dependent error tolerance ϵ is met or an iteration limit L is reached. Typical implementations would measure error with a suitably normalized version of the normal equation residual $\|\mathbf{F}^* (\mathbf{F}\boldsymbol{z} - \boldsymbol{g}) \|_2$. If κ denotes the condition number of $\mathbf{F}$ then typical convergence rates are such that error $\|\mathbf{F}(\boldsymbol{z} - \mathbf{F}^\dagger \boldsymbol{g})\|_2$ decays multiplicatively by a factor of $(\kappa - 1)/(\kappa + 1)$ with each iteration.

Besides the use of a common iterative solver, both algorithms below initialize the iterative solver at the solution from a sketch-and-solve approach in the vein of Section 3.2.1. The time needed to perform this presolve step is negligible, but it should save several iterations when solving to a prescribed accuracy. It also plays an important role in handling overdetermined least squares problems when $\boldsymbol{b}$ is in the range of $\mathbf{A}$. In such contexts, the sketch-and-solve result actually solves the least squares problem *exactly* provided that $\text{rank}(\mathbf{SA}) = \text{rank}(\mathbf{A})$; this stands in contrast to using a preconditioned iterative method initialized at the origin, which would not be able to achieve relative error guarantees for $\|\mathbf{A}\boldsymbol{x} - \boldsymbol{b}\|$ against $\|(\mathbf{I} - \mathbf{A}\mathbf{A}^\dagger)\boldsymbol{b}\| = 0$.

Algorithm 1 `SPO1`: a Blendenpik-like approach to overdetermined least squares

1: **function** `SPO1`$(\mathbf{A}, \boldsymbol{b}, \epsilon, L)$

Inputs:

$\mathbf{A}$ is $m \times n$ and $\boldsymbol{b}$ is an m-vector. We require $m \geq n$ and expect $m \gg n$. The iterative solver's termination criteria are governed by ϵ and L: it stops if the solution reaches error $\epsilon \geq 0$ according to the solver's error metric, or if the solver completes $L \geq 1$ iterations.

Output:

An approximate solution to (3.2), with $\boldsymbol{c} = \mathbf{0}$ and $\mu = 0$.

Abstract subroutines and tuning parameters:

`SketchOpGen` generates an oblivious sketching operator.

`sampling_factor` ≥ 1 is the size of the embedding dimension relative to n.

2: $d = \min\{\lceil n \cdot \texttt{sampling\_factor} \rceil, m\}$
3: $\mathbf{S} = \texttt{SketchOpGen}(d, m)$
4: $[\mathbf{A}^{\text{sk}}, \boldsymbol{b}_{\text{sk}}] = \mathbf{S}[\mathbf{A}, \boldsymbol{b}]$
5: $\mathbf{Q}, \mathbf{R} = \texttt{qr\_econ}(\mathbf{A}^{\text{sk}})$
6: $\boldsymbol{z}_o = \mathbf{Q}^* \boldsymbol{b}_{\text{sk}}$ # $\mathbf{R}^{-1}\boldsymbol{z}_o$ solves $\min_{\boldsymbol{x}}\{\|\mathbf{S}(\mathbf{A}\boldsymbol{x} - \boldsymbol{b})\|_2^2\}$
7: $\mathbf{A}_{\text{precond}} = \mathbf{A}\mathbf{R}^{-1}$ # as a linear operator
8: $\boldsymbol{z} = \texttt{iterative\_ls\_solver}(\mathbf{A}_{\text{precond}}, \boldsymbol{b}, \epsilon, L, \boldsymbol{z}_o)$
9: **return** $\mathbf{R}^{-1}\boldsymbol{z}$

While Algorithm 1 is standard, Algorithm 2 is somewhat novel. Using the same data that might be computed during a standard sketch-and-precondition algorithm for simple overdetermined least squares, it transforms any saddle point problem — primal or dual — into an equivalent primal saddle point problem with $\boldsymbol{c} = \mathbf{0}$. To our knowledge, no such conversion routines have been described in the literature. The conversion is advantageous because it opens the possibility of using iterative solvers with excellent numerical properties that are specific to least squares problems. The validity of the algorithm's transformation is explained towards the end of Section 3.3.1.

Algorithm 2 `SPS2` : sketch, transform a saddle point problem to least squares, and precondition. A more efficient version of this algorithm can be obtained using our observations on SVD-based preconditioning in Section 3.3.2.

function `SPS2`$(\mathbf{A}, \boldsymbol{b}, \boldsymbol{c}, \mu, \epsilon, L)$

Inputs:

$\mathbf{A}$ is $m \times n$, $\boldsymbol{b}$ is an m-vector, $\boldsymbol{c}$ is an n-vector, and μ is a nonnegative regularization parameter. We require $m \geq n$ and expect $m \gg n$. The iterative solver's termination criteria are governed by ϵ and L: it stops if the solution reaches error $\epsilon \geq 0$ according to its internal error metric, or if it completes $L \geq 1$ iterations.

Output:

Approximate solutions to (3.2) *and* its dual problem.

Abstract subroutines and tuning parameters:

`SketchOpGen` generates an oblivious sketching operator.

`sampling_factor` ≥ 1 is the size of the embedding dimension relative to n.

$d = \min\{\lceil n \cdot \texttt{sampling\_factor} \rceil, m\}$

$\mathbf{S} = \texttt{SketchOpGen}(d, m)$

if $\mu > 0$ **then**

$$\mathbf{S} = \begin{bmatrix} \mathbf{S} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_n \end{bmatrix}, \qquad \mathbf{A} = \begin{bmatrix} \mathbf{A} \\ \sqrt{\mu}\mathbf{I}_n \end{bmatrix}, \qquad \boldsymbol{b} = \begin{bmatrix} \boldsymbol{b} \\ \mathbf{0} \end{bmatrix}$$

$\mathbf{A}^{\text{sk}} = \mathbf{S}\mathbf{A}$

$\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}^* = \texttt{svd}(\mathbf{A}^{\text{sk}})$

$\mathbf{M} = \mathbf{V}\mathbf{\Sigma}^\dagger$

$\boldsymbol{b}_{\text{mod}} = \boldsymbol{b}$

if $\boldsymbol{c} \neq \mathbf{0}$ **then**

$\hat{\boldsymbol{v}} = \mathbf{U}\mathbf{\Sigma}^\dagger\mathbf{V}^*\boldsymbol{c}$ # $\hat{v}$ solves $\min_{\boldsymbol{v}}\{\|\boldsymbol{v}\|_2^2 : \mathbf{A}^*\mathbf{S}^*\boldsymbol{v} = \boldsymbol{c}\}$

$\boldsymbol{b}_{\text{shift}} = \mathbf{S}^*\hat{\boldsymbol{v}}$ # $\mathbf{A}^*\boldsymbol{b}_{\text{shift}} = \boldsymbol{c}$

$\boldsymbol{b}_{\text{mod}} = \boldsymbol{b}_{\text{mod}} - \boldsymbol{b}_{\text{shift}}$

$\boldsymbol{z}_o = \mathbf{U}^*\mathbf{S}\boldsymbol{b}_{\text{mod}}$ # $\mathbf{M}\boldsymbol{z}_o$ solves $\min\{\|\mathbf{S}(\mathbf{A}\boldsymbol{x} - \boldsymbol{b}_{\text{mod}})\|_2^2\}$

$\mathbf{A}_{\text{precond}} = \mathbf{A}\mathbf{M}$ # define implicitly, as a linear operator

$\boldsymbol{z} = \texttt{iterative\_ls\_solver}(\mathbf{A}_{\text{precond}}, \boldsymbol{b}_{\text{mod}}, \epsilon, L, \boldsymbol{z}_o)$

$\boldsymbol{x} = \mathbf{M}\boldsymbol{z}$

$\boldsymbol{y} = \boldsymbol{b}[:m] - \mathbf{A}[:m, :]\boldsymbol{x}$

return $\boldsymbol{x}$, $\boldsymbol{y}$

We wrap up our introduction to sketch-and-precondition algorithms by speaking to their tradeoffs with sketch-and-solve. It is easy to see that if $m \ll n^2$ and we perform sketch-and-solve using a direct method for Eq. (3.6), then performing an additional constant number of steps of sketch-and-precondition's iterative phase does not increase the FLOP count by even so much as a constant factor. However, if we are in the regime where $m \geq n^2$, then even a single step of an iterative method in sketch-and-precondition can cost as much as an entire sketch-and-solve algorithm. Therefore when an accurate solution is not required and $m \geq n^2$, it may be preferable to use sketch-and-solve rather than sketch-and-precondition.

Applications

One application of these algorithms is to carry out the core subroutine in iterative methods for solving linear systems by block projection. We explain the nature of this connection later on, in Section 5.2.2.

To explain the next application, we need some context. Classical linear algebra techniques to solve a KRR problem with m datapoints require $O(m^2)$ storage and $O(m^3)$ time. Rahimi and Recht's *random feature maps* provide a framework for replacing such a KRR problem with a more tractable ridge regression problem [RR07]. A data matrix in a *random features ridge regression problem* is $m \times n$ (for a tuning parameter $n < m$) and is characterized by the KRR datapoints and functions $f_1, \ldots, f_n$ drawn from a suitable random distribution. The i^{th} row in this matrix is obtained by evaluating $f_1, \ldots, f_n$ on the i^{th} KRR datapoint.

The randomness in random features ridge regression is not "sketching" in the sense meant by this monograph. Still, this approach is notable in our context because it provides a source of models that are amenable to the methodology described above. The Nyström preconditioning methodology (see Sections 3.2.3 and 3.3.3) has been reported to be especially effective for such problems when $n \lesssim m$ [FTU21].

When is sketch-and-precondition asymptotically faster than QR?

Here we detail the runtime of sketch-and-precondition algorithms under the assumption of sketching with SRFTs. These sketching operators were used in the original sketch-and-precondition paper [RT08] and subsequently by [AMT10]. We focus on them here because they have no tuning parameters besides their embedding dimension. Minimizing the number of tuning parameters helps us make comparisons to direct solvers based on QR decomposition that run in time $O(mn^2)$.

Recall from Section 2.5 that it takes $O(mn \log d)$ time to apply a $d \times m$ SRFT to an $m \times n$ matrix. We can plug $T_{\text{sk}} = mn \log d$ into (3.8) to see that the "typical" runtime for sketch-and-precondition with an SRFT is

$$O(mn \log d + dn^2 + mn \log(1/\epsilon)). \tag{3.9}$$

This runtime is only "typical" because it does not address subtleties stemming from randomness. In the algorithm's true runtime, there is a random multiplicative factor F on the $mn \log(1/\epsilon)$ term in (3.9). The distribution of F depends on (d, n) in a complicated way. In formal algorithm analysis, one describes how to choose d to upper-bound the probability that F exceeds some universal constant C. Then one can say that (3.9) *does* describe the algorithm's true runtime with some probability. The convention in the field is to describe how to choose d so the probability that $F \leq C$ tends to one as problem size increases.

[RT08] observed that taking $d = sn$ for small constants s (e.g., $s = 4$) sufficed for (3.9) to accurately describe algorithm runtime in practice. However, the theoretical analysis in [RT08] needed to take $d \in \Omega(n^2)$ to bound F with high probability. Therefore the best theoretical runtime guarantee for sketch-and-precondition was originally obtained by plugging $d = n^2$ into (3.9). The theoretical guarantees improved following developments in the analysis of SRFTs. Specifically, [AMT10] observed that a transparent application of a result by [NDT09] could be used to prove that $d \in \Omega(n \log n)$ sufficed to bound F with high probability. Therefore one can plug $d = n \log n$ into (3.9) to obtain a bound for algorithm runtime in terms of (m, n, ϵ) that holds with high probability. This is the appropriate bound to use when comparing the theoretical asymptotic runtime of SRFT-based sketch-and-precondition to other algorithms. However, in practice, it is still preferred to use $d = sn$ for some small $s > 1$, since the resulting preconditioned matrices tend to be extremely well-conditioned.

3.2.3 Nyström PCG for minimizing regularized quadratics

Nyström preconditioned conjugate gradient (Nyström PCG) is a recently-proposed method for solving problems of the form (3.1) to fairly high accuracy [FTU21]. We describe it as a method to compute approximate solutions to linear systems $(\mathbf{G} + \mu\mathbf{I})\boldsymbol{x} = \boldsymbol{h}$ where $\mathbf{G}$ is $n \times n$ and psd.

The randomness in Nyström PCG is encapsulated in an initial phase where it computes a low-rank approximation of $\mathbf{G}$ by a so-called "Nyström approximation." We defer discussion on such approximations (including the potentially-confusing naming convention) to Section 4.2.2. For our purposes, what matters is that a rank-ℓ Nyström approximation leads to a preconditioner $\mathbf{P}$ which can be stored in $O(\ell n)$ space and applied in $O(\ell n)$ time.

Now let κ denote the condition number of $\mathbf{G}_p := \mathbf{P}^{-1/2}(\mathbf{G} + \mu\mathbf{I})\mathbf{P}^{-1/2}$. It is well-known that each iteration of PCG requires one matrix-vector multiply with $\mathbf{G}$, one matrix-vector multiply with $\mathbf{P}^{-1}$, and reduces the error of the candidate solution to (3.1) by a multiplicative factor $(\sqrt{\kappa}-1)/(\sqrt{\kappa}+1)$. As we discuss below, one can expect that κ will be $O(1)$ if the ℓ^{th}-largest eigenvalue of $\mathbf{G}$ is smaller than μ. Indeed, Nyström PCG is most effective for problems when this threshold is crossed at some $\ell \ll n$. As a practical matter, users will not need to select the approximation rank parameter ℓ manually in order to use Nyström PCG; [FTU21, Algorithm E.2] is a specialized adaptive method for Nyström approximation that can determine an appropriate value for ℓ given $(\mathbf{G}, \mu)$.

Details on the preconditioner

We presume access to a low-rank approximation

$$\hat{\mathbf{G}} = \mathbf{V}\,\mathrm{diag}(\boldsymbol{\lambda})\mathbf{V}^* \tag{3.10}$$

where $\mathbf{V}$ is a column-orthonormal $n \times \ell$ matrix that approximates the dominant ℓ eigenvectors of $\mathbf{G}$ and $\lambda_1 \geq \cdots \geq \lambda_\ell > 0$ are the approximated eigenvalues. The data $(\mathbf{V}, \boldsymbol{\lambda}, \mu)$ is then used to define a preconditioner

$$\mathbf{P}^{-1} = \mathbf{V}\,\mathrm{diag}(\boldsymbol{\lambda} + \mu)^{-1}\mathbf{V}^* + (\mu + \lambda_\ell)^{-1}(\mathbf{I}_n - \mathbf{V}\mathbf{V}^*). \tag{3.11}$$

Alternatively, following [FTU21] to the letter, $\mathbf{P}^{-1}$ can be the result of multiplying the expression above by $(\mu + \lambda_\ell)$. Under this latter convention, $\mathbf{P}^{-1}$ acts as the identity on range$(\mathbf{V})^\perp$.

While the form of this preconditioner may appear mysterious, its appropriateness can be seen by considering a simple idealized setting. To make a precise statement on this topic we adopt notation where $\lambda_i(\mathbf{G})$ is the i^{th}-largest eigenvalue of $\mathbf{G}$. Assuming that $(\mathbf{V}, \boldsymbol{\lambda})$ are very good estimates for the top ℓ eigenpairs of $\mathbf{G}$ *and* that $\lambda_\ell(\mathbf{G}) \approx \lambda_{\ell+1}(\mathbf{G})$, the condition number of $\mathbf{G}_p$ should be near

$$\kappa_\ell(\mathbf{G}, \mu) := (\lambda_\ell(\mathbf{G}) + \mu)/(\lambda_n(\mathbf{G}) + \mu).$$

Taking this for granted, the preconditioner (3.11) can only be effective if $\ell \ll n$ is large enough so that $\kappa_\ell(\mathbf{G}, \mu)$ is bounded by a small constant. Using the fact that $\kappa_\ell(\mathbf{G}, u) \leq 1 + \lambda_\ell(\mathbf{G})/\mu$, we can simplify the criteria and say that a good preconditioner is possible when $\lambda_\ell(\mathbf{G})/\mu$ is $O(1)$.

Remark 3.2.1. The argument above can be made more rigorous by assuming that $\mathbf{V}$ is an $n \times (\ell - 1)$ matrix that contains the *exact* leading $\ell - 1$ eigenvectors of $\mathbf{G}$, and that $\lambda_1, \ldots, \lambda_\ell$ are the *exact* leading ℓ eigenvalues of $\mathbf{G}$. In this case, the condition number of $\mathbf{G}_p$ will be equal to $\kappa_\ell(\mathbf{G}, \mu)$, which will be at most $1 + \lambda_\ell/\mu$.

3.2.4 Sketch-and-solve for minimizing regularized quadratics

Randomization offers several avenues for solving problems of the form (3.1) to modest accuracy. We describe two possible methods here through novel interpretations of existing work on KRR. Our descriptions of the methods keep the focus on linear algebra, and we refer the reader to Appendix B.4.1 for information on the KRR formalism. We note that our formulations of these methods are novel in how they apply sketch-and-precondition as the core subroutine in what is otherwise a sketch-and-solve style driver. Such "nested randomization" is a relatively under-explored and potentially powerful algorithm design paradigm.

For notation, we shall say that $\mathbf{G}$ is $m \times m$, that $\mu = m\lambda$ for some $\lambda > 0$, and that the optimization variable in (3.1) is denoted by "$\boldsymbol{\alpha}$" rather than "$\boldsymbol{x}$."

A one-shot fallback on Nyström approximations

Rather than solving (3.1) directly, it has been suggested that one solve

$$(\mathbf{A}\mathbf{A}^* + m\lambda\mathbf{I})\,\hat{\boldsymbol{\alpha}} = \boldsymbol{h},$$

where $\mathbf{A}\mathbf{A}^*$ is a Nyström approximation of $\mathbf{G}$ [AM15]. The computation of $\mathbf{A}$ only requires access to $\mathbf{G}$ by a single sketch $\mathbf{GS}$ for a tall $m \times n$ sketching operator $\mathbf{S}$. In the KRR context, it is especially popular for $\mathbf{S}$ to be a column sampling operator [WS00; KMT09b; GM16]. Section 6.1.3 discusses how such column-selection sketches $\mathbf{GS}$ can be computed adaptively using the concept of *ridge leverage scores*. Regardless of how the approximation is obtained, there is an equivalence between computing $\hat{\boldsymbol{\alpha}}$ and solving a dual saddle point problem with matrix $\mathbf{A}$ and other data $(\boldsymbol{b}, \boldsymbol{c}, \mu) = (\boldsymbol{h}, \mathbf{0}, m\lambda)$. That dual saddle point problem can naturally be approached by sketch-and-precondition methods from Section 3.2.2. The preconditioner generation steps in this context are subtle and addressed in Appendix B.4.2.

Applying a random subspace constraint

By taking the gradient of the objective function in (3.1) and multiplying the gradient by the positive definite matrix $\mathbf{G}$, we can recast (3.1) as minimizing

$$Q(\boldsymbol{\alpha}) = \boldsymbol{\alpha}^*(\mathbf{G}^2 + m\lambda\mathbf{G})\boldsymbol{\alpha} - 2\boldsymbol{h}^*\mathbf{G}\boldsymbol{\alpha}.$$

In [YPW17], a sketch-and-solve approach to the problem of minimizing this loss function is proposed. Specifically, one minimizes $Q(\boldsymbol{\alpha})$ subject to a constraint that $\boldsymbol{\alpha}$ is in the range of a very tall $m \times n$ sketching operator $\mathbf{S}$. The constrained minimization problem is equivalent to minimizing $\boldsymbol{z} \mapsto Q(\mathbf{S}\boldsymbol{z})$ over n-vectors $\boldsymbol{z}$. This in turn is equivalent to solving a highly overdetermined least squares problem, with an $(m+n) \times n$ data matrix $\mathbf{A} = [\mathbf{GS}; \sqrt{m\lambda}\mathbf{R}]$ where $\mathbf{R}$ is any matrix for which $\mathbf{R}^*\mathbf{R} = \mathbf{S}^*\mathbf{GS}$. This problem can clearly be handled by our methods from Section 3.2.2.

Remark 3.2.2. We note that [YPW17] presumes access to the sketches $\boldsymbol{h}^*\mathbf{GS}$, $\mathbf{S}^*\mathbf{GS}$, and $\mathbf{S}^*\mathbf{G}^2\mathbf{S}$, and advocates for solving the resulting n-dimensional minimization problem by a direct method in $O(n^3)$ time. However, no guidance is given on how to compute the sketch $\mathbf{S}^*\mathbf{G}^2\mathbf{S}$. From what we can tell, the most efficient way of doing this would be to form the Gram matrix at cost $O(mn^2)$ assuming access to the sketch $\mathbf{GS}$. (Our usage of (m, n) is swapped relative to [YPW17].)

3.3 Computational routines

To contextualize the computational routines that follow, we begin in Section 3.3.1 with a brief discussion of optimality conditions for saddle point problems. From there, we present in Sections 3.3.2 and 3.3.3 two families of methods for generating preconditioners needed by saddle point drivers; our presentation of both families includes novel observations that lead to improved efficiency and numerical stability. Then in Section 3.3.4 we discuss deterministic preconditioned iterative methods for positive definite systems and saddle point problems. Such iterative methods are applicable to all drivers from the previous section (although less so for Section 3.2.1).

Routines not detailed here

The driver from Section 3.2.3 requires methods to compute Nyström approximations, which are described in Section 4. In addition, the drivers from Section 3.2.4 would benefit from specialized data-aware methods for sketching kernel matrices, which are discussed in Section 6. We also note that this section does not describe computational routines for sketch-and-solve type drivers. This is because those drivers are extraordinarily simple to implement and there is no need to isolate their building blocks into separate computational routines.

3.3.1 Technical background: optimality conditions for saddle point problems

Here, we give a handful of characterizations of optimal solutions for saddle point problems. Let us begin by calling an n-vector $\boldsymbol{x}$ *primal-optimal* if it solves (3.2). Analogously, an m-vector $\boldsymbol{y}$ shall be called *dual-optimal* if it solves (3.3) when μ is positive or (3.4) when μ is zero.

Primal-dual optimal solutions can be characterized with *saddle point systems*. These are a class of 2×2 block linear systems that arise broadly in computational mathematics and especially in optimization. General introductions to these systems can be found in the survey [BGL05] and the book [OA17]. We are interested in saddle point systems of the form

$$\begin{bmatrix} \mathbf{I} & \mathbf{A} \\ \mathbf{A}^* & -\mu\mathbf{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{x} \end{bmatrix} = \begin{bmatrix} \boldsymbol{b} \\ \boldsymbol{c} \end{bmatrix}. \tag{3.12}$$

A solution to such a system always exists when μ is positive or when the tall matrix $\mathbf{A}$ is full-rank. Given that assumption, it can be shown that a point $\tilde{\boldsymbol{x}}$ is primal-optimal if and only if there is a $\tilde{\boldsymbol{y}}$ for which $(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}})$ solve (3.12). Similarly, a point $\tilde{\boldsymbol{y}}$ is dual-optimal if and only if there is an $\tilde{\boldsymbol{x}}$ for which $(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}})$ solve (3.12).

Saddle point systems are often reformulated into equivalent positive semidefinite systems. The reformulation takes the system's upper block to *define* $\boldsymbol{y} = \boldsymbol{b} - \mathbf{A}\boldsymbol{x}$, and then substitutes that expression into the system's lower block. This gives us the *normal equations*

$$(\mathbf{A}^*\mathbf{A} + \mu\mathbf{I})\boldsymbol{x} = \mathbf{A}^*\boldsymbol{b} - \boldsymbol{c}. \tag{3.13}$$

Therefore one can solve (3.12) by first solving (3.13) and then setting $\boldsymbol{y} = \boldsymbol{b} - \mathbf{A}\boldsymbol{x}$. Such an approach to underdetermined least squares is suggested by Björck in his books [Bjö96; Bjö15].

Thinking in terms of the normal equations helps with the design of preconditioners. When accurate solutions are desired, however, it is preferable to employ reformulations that reduce the need for matrix-vector products with the linear operator $\mathbf{A}^*\mathbf{A}$. Such reformulations start by defining an augmented data matrix $\mathbf{A}_\mu = [\mathbf{A}; \sqrt{\mu}\mathbf{I}_n]$. For dual saddle point problems, one solves

$$\min\{\, \|\Delta\boldsymbol{y}\|_2^2 \;:\; \Delta\boldsymbol{y} \in \mathbb{R}^{m+n},\ (\mathbf{A}_\mu)^*\Delta\boldsymbol{y} = \boldsymbol{c} - \mathbf{A}^*\boldsymbol{b}\}, \tag{3.14}$$

and subsequently recovers the dual-optimal solution $\boldsymbol{y} = [b_1 + \Delta y_1; \ldots; b_m + \Delta y_m]$. For primal saddle point problems, one computes *some* $\boldsymbol{b}_{\text{shift}} \in \mathbb{R}^{m+n}$ satisfying $(\mathbf{A}_\mu)^*\boldsymbol{b}_{\text{shift}} = \boldsymbol{c}$ and then defines $\boldsymbol{b}_\mu = [\boldsymbol{b}; \mathbf{0}_n] - \boldsymbol{b}_{\text{shift}}$. Any solution to the resulting problem

$$\min_{\boldsymbol{x} \in \mathbb{R}^n} \left\{ \|\mathbf{A}_\mu\boldsymbol{x} - \boldsymbol{b}_\mu\|_2^2 \right\} \tag{3.15}$$

is primal-optimal. Of course, this reformulation is only useful if we have a cheap way to compute $\boldsymbol{b}_{\text{shift}}$. As it happens, however, randomized methods for preconditioner generation provide methods to compute a near-minimum-norm solution to $\mathbf{A}^*\boldsymbol{u} = \boldsymbol{c}$ in $O(mn)$ extra time compared to when $\boldsymbol{c} = \mathbf{0}$. We illustrated this process earlier with an SVD-based preconditioner in Algorithm 2.

Inconsistent saddle point systems

Suppose that μ is zero, so as to allow for the possibility that (3.12) is consistent. Under this assumption, (3.12) is inconsistent if and only if $\boldsymbol{c}$ is not in the range of $\mathbf{A}^*$. When framed in this way, we have that (3.12) is inconsistent if and only if (3.4) has no feasible solution. What's more, since $\boldsymbol{c} \notin \text{range}(\mathbf{A}^*)$ is equivalent to $\boldsymbol{c} \notin \ker(\mathbf{A})^\perp$, we see that inconsistency of (3.12) is equivalent to (3.2) having no optimal solution. Therefore a saddle point system is consistent if and only if its associated saddle point problems are well-posed; for ill-posed problems, recall that we canonically assign solutions per (3.5).

3.3.2 Preconditioning least squares and saddle point problems: tall data matrices

There is a simple unifying framework for preconditioner generation of the kind used in [RT08; AMT10; MSM14]. The framework is applicable to any least squares or saddle point problem (3.2)–(3.4) in the regime $m \gg n$. We describe its general form below and then turn to its concrete instantiations.

Sketch and orthogonalize

To describe our framework, begin by defining a sketch $\mathbf{A}^{\mathrm{sk}} = \mathbf{SA}$ where the sketching operator $\mathbf{S}$ has $d \gtrsim n$ rows. We also define the augmented matrices

$$\mathbf{A}_\mu = \begin{bmatrix} \mathbf{A} \\ \sqrt{\mu}\mathbf{I} \end{bmatrix} \quad \text{and} \quad \mathbf{A}_\mu^{\mathrm{sk}} = \begin{bmatrix} \mathbf{A}^{\mathrm{sk}} \\ \sqrt{\mu}\mathbf{I} \end{bmatrix}.$$

These augmented matrices are only used as a formalism. They reflect the influence of the normal equations (3.13) on preconditioner design. We emphasize that we specifically allow for $\mu = 0$ and one need not form these augmented matrices explicitly in memory.

Next, we introduce two key terms.

> We say that a matrix $\mathbf{M}$ *orthogonalizes* $\mathbf{A}_\mu^{\mathrm{sk}}$ if the columns of $\mathbf{A}_\mu^{\mathrm{sk}}\mathbf{M}$ are an orthonormal basis for the range of $\mathbf{A}_\mu^{\mathrm{sk}}$. Such a matrix is called a *valid preconditioner* for $\mathbf{A}_\mu$ if, in addition, $\mathrm{rank}(\mathbf{A}_\mu^{\mathrm{sk}}) = \mathrm{rank}(\mathbf{A}_\mu)$.

We note that the rank requirement of a valid preconditioner is nearly universal in practice. For example, it holds with probability one for uniform and Gaussian operators (§2.3). We conjecture that it holds with exponentially-high probability for suitable SASOs (§2.4.1) and for SRFTs (§2.5).

How good are these preconditioners? In our context, $\mathbf{M}$ is a good preconditioner if the spectrum of $\mathbf{A}_\mu\mathbf{M}$ can be divided into a small number of tightly clustered groups. Given the tools at our disposal in RandNLA, we mostly aim for the spectrum of this matrix to be tightly clustered into a single group, i.e., for its condition number to be small. In this regard, we can provide the following principle.

> *If $\mathbf{M}$ is a valid preconditioner for $\mathbf{A}_\mu$, then the condition number of $\mathbf{A}_\mu\mathbf{M}$ does not depend that of $\mathbf{A}_\mu$.*

This principle can be formalized with the following proposition, which we state without regularization for the sake of clarity.

Proposition 3.3.1. *Let $\mathbf{U}$ be a matrix whose columns form an orthonormal basis for the range of $\mathbf{A}$. If $\mathbf{M}$ is a valid preconditioner for $\mathbf{A}$, then the spectrum of $\mathbf{AM}$ is equal to that of $(\mathbf{SU})^\dagger$.*

[RT08, Theorem 1] gives a very similar statement under the assumption that $\mathbf{A}$ is full-rank. [MSM14, Lemma 4.2] improved upon [RT08] by supporting the rank-deficient case, at the price of strong assumptions on the sketching operator and the form of the preconditioner. In Appendix B.1 we provide what to our knowledge is the first proof of Proposition 3.3.1 in its general form; we also explain its application to regularized problems.

Up next. We now turn to how one can compute orthogonalizers. To keep things at a reasonable length we only speak to QR-based and SVD-based methods, although others could also be used. Our goal is to give a general overview that includes time and space complexity considerations. As to the latter consideration, we must note that these preconditioners have insubstantial space requirements when $\mathbf{A}$ is dense and $m \gg d \gtrsim n$. Separately, we note that details of the preconditioner generation process can affect the sketch-and-solve preprocessing step in sketch-and-precondition algorithms. For more information on theoretical properties of these preconditioners in a RandNLA context, we refer the reader to [CFS21].

QR-based preconditioning in the full-rank case

QR-based preconditioning when $\mu = 0$ is very simple; one need only run Householder QR on $\mathbf{A}^{\mathrm{sk}}$ and return $\mathbf{M} = \mathbf{R}^{-1}$ as a linear operator. We note that specialized methods for QR decomposition of very tall matrices would not be appropriate here, since the $d \times n$ matrix $\mathbf{A}^{\mathrm{sk}}$ will have $d \gtrsim n$. Householder-type representations of $\mathbf{A}^{\mathrm{sk}}$'s QR decomposition are especially useful since they require a modest amount of added workspace on top of storing $\mathbf{A}^{\mathrm{sk}}$.

The case with $\mu > 0$ is more complicated if we want to avoid forming $\mathbf{A}_\mu^{\mathrm{sk}}$ explicitly. To describe it, suppose we have an initial QR decomposition $\mathbf{A}^{\mathrm{sk}} = \mathbf{Q}_o\mathbf{R}_o$. It is easy to show that the factor $\mathbf{R}$ from a QR decomposition of $\mathbf{A}_\mu^{\mathrm{sk}}$ is the same as the triangular factor from a QR decomposition of $\hat{\mathbf{R}} := [\mathbf{R}_o; \sqrt{\mu}\mathbf{I}]$. This observation is useful because there are specialized algorithms for QR decomposition of matrices given by an implicit vertical concatenation of a triangular matrix and a diagonal matrix; these specialized algorithms only require $O(n)$ additional workspace. The factor $\mathbf{Q}$ from a QR decomposition of $\mathbf{A}_\mu^{\mathrm{sk}}$ can also be recovered with this approach, although the representation would be somewhat complicated.

If $\mathbf{A}$ is not too ill-conditioned then the same preconditioner can be obtained by Cholesky-decomposing the regularized Gram matrix

$$(\mathbf{A}_\mu^{\mathrm{sk}})^*(\mathbf{A}_\mu^{\mathrm{sk}}) = (\mathbf{A}^{\mathrm{sk}})^*(\mathbf{A}^{\mathrm{sk}}) + \mu\mathbf{I},$$

since the upper-triangular Cholesky factor of that matrix is the same as the factor $\mathbf{R}$ from the QR decomposition of $\mathbf{A}_\mu^{\mathrm{sk}}$. This approach is simple to implement, and its time and space requirements are unaffected by whether or not μ is zero. A sophisticated implementation could even try to form the regularized Gram matrix without allocating dn space for $\mathbf{A}^{\mathrm{sk}}$ as an intermediate quantity. Although, it is clear that unless such a sophisticated implementation is used, there is no material memory savings compared to the Q-less QR approach described above. This approach also affects sketch-and-solve preprocessing by requiring that we solve the normal equations, which is not a numerically stable approach [Bjö96].

QR-based preconditioning in the rank-deficient case

Suppose for ease of exposition that $\mu = 0$ and let $k = \mathrm{rank}(\mathbf{A}^{\mathrm{sk}}) \lesssim n$. One can use a variety of methods to compute preconditioners that are *morally triangular* in the sense that they are of the form $\mathbf{M} = \mathbf{P}\mathbf{R}^{-1}$ for an $n \times k$ partial-permutation matrix $\mathbf{P}$ and a triangular matrix $\mathbf{R}$. As long as the preconditioner orthogonalizes $\mathbf{A}^{\mathrm{sk}}$, we can postprocess $\boldsymbol{z}_{\mathrm{sol}} = \mathrm{argmin}\, \|\mathbf{A}\mathbf{M}\boldsymbol{z} - \boldsymbol{b}\|_2^2$ to obtain $\boldsymbol{x}_{\mathrm{sol}} = \mathbf{M}\boldsymbol{z}$ which solves $\min \|\mathbf{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2$.

The subtlety here is that when $k < n$ there is a nontrivial affine subspace of optimal solutions to $\min \|\mathbf{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2$. Our stated goal in the rank-deficient case is to find the minimum-norm solution to the least squares problem (see Eq. (3.5)). Unfortunately, if we assume that $\boldsymbol{b}$ has no role in defining $\mathbf{M}$, then it is clearly impossible to guarantee that the norm of the recovered solution is anywhere near the minimum possible among all minimizers of $\|\mathbf{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2$.

SVD-based preconditioners

Let us denote the SVD of $\mathbf{A}^{\text{sk}}$ by $\mathbf{U}\,\text{diag}(\boldsymbol{\sigma})\mathbf{V}^*$.

First we consider preconditioner generation when $\mu = 0$. In this case we must account for the fact that $\mathbf{A}^{\text{sk}}$ might be rank-deficient. Letting k denote the rank of $\mathbf{A}^{\text{sk}}$, the SVD-based preconditioner is the $n \times k$ matrix

$$\mathbf{M} = \left[\frac{\boldsymbol{v}_1}{\sigma_1}, \ldots, \frac{\boldsymbol{v}_k}{\sigma_k} \right].$$

This construction is important, because it can be shown that if $\boldsymbol{z}_\star$ solves

$$\min_{\boldsymbol{z}} \|\mathbf{A}\mathbf{M}\boldsymbol{z} - \boldsymbol{b}\|_2^2 + \boldsymbol{c}^*\mathbf{M}\boldsymbol{z} \tag{3.16}$$

then $\boldsymbol{x} = \mathbf{M}\boldsymbol{z}_\star$ satisfies (3.5). We note in particular that (3.16) has a unique optimal solution and so computing $\boldsymbol{z}_\star$ is a well-posed problem.

SVD-based preconditioning is conceptually simpler when μ is positive, since in that case it does not matter if $\mathbf{A}^{\text{sk}}$ is rank-deficient. However, it is harder to efficiently implement compared to when $\mu = 0$. Here we present an efficient construction based on the relationship between the SVD of a matrix and the eigendecomposition of its Gram matrix. Specifically, recall that the right singular vectors of a matrix $\mathbf{F}$ are the eigenvectors of $\mathbf{F}^*\mathbf{F}$, and that the singular values of $\mathbf{F}$ are the square roots of the eigenvalues of $\mathbf{F}^*\mathbf{F}$.

When used in our context this fact implies that the right singular vectors of $\mathbf{A}_\mu^{\text{sk}}$ are equal to those of $\mathbf{A}^{\text{sk}}$, and that its singular values are

$$\hat{\sigma}_i = \sqrt{\sigma_i^2 + \mu}.$$

These observations alone are sufficient to recover the preconditioner

$$\mathbf{M} = \mathbf{V}\,\text{diag}\left(\frac{1}{\hat{\sigma}_1}, \ldots, \frac{1}{\hat{\sigma}_n} \right)$$

which orthogonalizes $\mathbf{A}_\mu^{\text{sk}}$.

As a final point we consider the problem of recovering the left singular vectors of $\mathbf{A}_\mu^{\text{sk}}$ given the SVD of $\mathbf{A}^{\text{sk}}$. This is useful in settings such as Algorithm 2 for presolve and problem transformation purposes. Moreover, it can actually be done efficiently. If we define

$$\mathbf{D}_1 = \text{diag}\left(\frac{\sigma_1}{\hat{\sigma}_1}, \ldots, \frac{\sigma_n}{\hat{\sigma}_n} \right) \quad \text{and} \quad \mathbf{D}_2 = \text{diag}\left(\frac{\sqrt{\mu}}{\hat{\sigma}_1}, \ldots, \frac{\sqrt{\mu}}{\hat{\sigma}_n} \right)$$

then by assumption on $\mathbf{M}$ the left singular vectors of $\mathbf{A}_\mu^{\text{sk}}$ are given by

$$\begin{bmatrix} \mathbf{A}^{\text{sk}} \\ \sqrt{\mu}\mathbf{I} \end{bmatrix} \mathbf{M} = \begin{bmatrix} \mathbf{A}^{\text{sk}}\mathbf{M} \\ \sqrt{\mu}\mathbf{M} \end{bmatrix} = \begin{bmatrix} \mathbf{U}\mathbf{D}_1 \\ \mathbf{V}\mathbf{D}_2 \end{bmatrix}.$$

We note that the column-orthonormality of this matrix can easily be verified by its rightmost representation.

To recap, we have introduced three key benefits of SVD-based preconditioning for tall least squares and saddle point problems. First, it can be used to find the minimum norm solutions in (3.5) in the rank-deficient case. Second, an SVD-based preconditioner can be computed in the presence of regularization given only the singular values and right singular vectors of $\mathbf{A}^{\text{sk}}$. Third, the SVD of $\mathbf{A}^{\text{sk}}$ is sufficient to recover the SVD of $\mathbf{A}^{\text{sk}}_{\mu}$, which facilitates sketch-and-solve as a preprocessing step in sketch-and-precondition.

Remark 3.3.2 (Computational complexity). The default algorithm for SVD is currently divide-and-conquer [GE95]. Two somewhat-outdated algorithms for computing the SVD are described in [GV13, §8.6]; [GV13, Figure 8.6.1] gives complexity estimates for these algorithms depending on whether the left singular vectors need to be computed.

3.3.3 Preconditioning least squares and saddle point problems: data matrices with fast spectral decay

Interpreting the Nyström preconditioner. Recall from Section 3.2.3 that the Nyström preconditioning approach to solving $(\mathbf{G} + \mu\mathbf{I})\boldsymbol{x} = \boldsymbol{h}$ starts by constructing a low-rank approximation of $\mathbf{G}$. That approximation defines a preconditioner $\mathbf{P}$ satisfying three properties:

1. $\mathbf{P}$ is positive definite.
2. $(\mathbf{G} + \mu\mathbf{I})\mathbf{P}^{-1}$ is well-conditioned on a subspace L that contains $\mathbf{G}$'s dominant eigenspaces.
3. $\mathbf{P}$ acts as the identity on $L^{\perp}$ (the orthogonal complement of L).

Such a preconditioner will be effective when the action of $\mathbf{G}$ on $L^{\perp}$ is "not too pronounced" compared to that of $\mu\mathbf{I}$. More formally, if we define the restricted spectral norm of $\mathbf{G}$ on $L^{\perp}$

$$\|\mathbf{G}\|_{L^{\perp}} = \max\{\|\mathbf{G}\boldsymbol{z}\|_2 \,:\, \boldsymbol{z} \in L^{\perp}, \|\boldsymbol{z}\|_2 = 1\}$$

then the preconditioner will be effective if $\|\mathbf{G}\|_{L^{\perp}}/\mu$ is $O(1)$.

Adaptation to saddle point problems. Nyström preconditioners can naively be used for regularized saddle point problems by taking $\mathbf{G} = \mathbf{A}^*\mathbf{A}$ and considering the normal equations (3.13). However, the numerical properties of iterative least squares solvers that only access $\mathbf{A}^*\mathbf{A}$ tend to be less robust than those of iterative solvers that access $\mathbf{A}$ and $\mathbf{A}^*$ separately (i.e., solvers such as LSQR). This motivates having an extension of the Nyström preconditioner to be compatible with the latter type of solver.

Towards this end, let us express $\mathbf{P}^{-1}$ with a (possibly non-symmetric) matrix square-root $\mathbf{M}$, satisfying the relation $\mathbf{P}^{-1} = \mathbf{M}\mathbf{M}^*$. We appeal to the well-known fact that running PCG on $(\mathbf{G} + \mu\mathbf{I})\boldsymbol{z} = \boldsymbol{h}$ with preconditioner $\mathbf{P}$ is equivalent to running the "unpreconditioned" CG algorithm on

$$\mathbf{M}^*(\mathbf{G} + \mu\mathbf{I})\mathbf{M}\boldsymbol{z} = \mathbf{M}^*\boldsymbol{h}.$$

When framed in this way, we can ask how $\mathbf{M}$ should relate to $\mathbf{A}$ and μ so that it *would* be a good preconditioner *if* it were used on the normal equations.

To answer this question we work with the augmented matrix $\mathbf{A}_\mu = [\mathbf{A}; \sqrt{\mu}\mathbf{I}_n]$. The basic criteria for a $\mathbf{P}$ as a Nyström preconditioner for the normal equations (3.13) can be stated with $\mathbf{M}$ as follows:

1. $\mathbf{A}_\mu\mathbf{M}$ should be well-conditioned on a subspace L that includes the dominant right singular vectors of $\mathbf{A}_\mu$.
2. we should have $\mathbf{A}_\mu\mathbf{M}\boldsymbol{x} = \mathbf{A}_\mu\boldsymbol{x}$ for all $\boldsymbol{x} \in L^\perp$.

Whether such a preconditioner will be effective can be stated with the "restricted spectral norm" as defined above. Specifically, $\mathbf{M}$ should be effective if the above conditions hold and $\|\mathbf{A}\|_{L^\perp}/\sqrt{\mu}$ is $O(1)$. We note that the requisite matrix $\mathbf{M}$ can be constructed efficiently by similar principles as methods for low-rank SVD in Section 4, and we leave the details to future work.

3.3.4 Deterministic preconditioned iterative solvers

Most of the drivers described in Section 3.2 amount to using randomization to obtain a preconditioner and then calling a traditional iterative solver that can make use of that randomized preconditioner. Here we list some iterative solvers that could be of use for these drivers. We note up front that many factors can affect the ideal choice of iterative method in a given setting.

- **CG** [HS52] is the most broadly applicable solver in our context. It applies to the regularized positive definite system (3.1) and the normal equations of the primal saddle point problem (3.13).
- **CGLS** [HS52] applies when $\boldsymbol{c}$ is zero. It is equivalent to CG on the normal equations in exact arithmetic, but is more stable than CG in finite-precision arithmetic.
- **LSQR** [PS82] applies when at least one of $\boldsymbol{c}$ or $\boldsymbol{b}$ is zero. When considered for overdetermined problems it is algebraically (but not numerically) equivalent to CGLS. It is more stable than CG [Bjö96, § 7.6.3], [Bjö15, § 4.5.4] and CGLS [PS82, § 9] for ill-conditioned problems.
- **CS** (the Chebyshev semi-iterative method) [GV61] applies to the same class of systems as CG. It has fewer synchronization points in each iteration and so can take better advantage of parallelism. It requires knowledge of an upper bound and a lower bound on the eigenvalues of the system matrix. We refer the reader to [Bjö96, § 7.2.5], [Bjö15, § 4.1.7] for information on this method.
- **LSMR** [FS11] applies to the same problems as LSQR. For overdetermined least squares it is algebraically equivalent to MINRES on the normal equations. In that context, the residual of the normal equations will decrease with each iteration, which makes it safer to stop early compared to LSQR.

These algorithms vary in how they accommodate preconditioners. Some require implicitly preconditioning the problem data, calling the "unpreconditioned" solver, then applying some (cheap) postprocessing to the returned solution. We note that it is necessary to "precondition" any regularization term in the problem's objective

when using such an algorithm.[3] For other algorithms, a preconditioner is supplied alongside the problem data, and the algorithm returns a solution that requires no postprocessing. The difference between these situations is that different quantities end up being available for use in termination criteria (at least for off-the-shelf implementations). We emphasize that appropriate choices of termination criteria can be crucial for iterative solvers to work effectively, and we refer the reader to Appendix B.2 for discussion on this and other topics.

Any standard library implementing the drivers from Section 3 should include computational methods for (preconditioned) CG and LSQR. LSQR is most naturally applied to dual saddle point problems by reformulation to (3.14) and to primal saddle point problems by reformulation to (3.15). More full-featured RandNLA libraries would do well to include implementations of CS or LSMR, and "blocked" versions of iterative solvers. Such blocked methods apply to linear systems and least squares problems with multiple right-hand sides; they take better advantage of parallel hardware and have slightly faster convergence rates than their non-blocked counterparts.

3.4 Other optimization functionality

Here, we briefly discuss other RandNLA algorithms of note for least squares or optimization, often commenting on how they fit into our plans for RandLAPACK. Some of these algorithms are out-of-scope for a linear algebra library but can be directly facilitated by the drivers we described in Section 3.

Facilitating second-order optimization algorithms

Many second-order optimization algorithms need to solve sequences of saddle point systems, where $\mathbf{A}, \boldsymbol{b}, \boldsymbol{c}$ vary continuously from one iteration to the next. RandLAPACK will support such use-cases *indirectly* through methods that help amortize the dominant computational cost of a single randomized algorithm across multiple saddle point solves. See [PW17; RM19] for uses of RandNLA for second-order optimization.

The most common way for $\mathbf{A}$ to vary is by a reweighting: when $\mathbf{A} = \mathbf{W}\mathbf{A}_o$ for a fixed matrix $\mathbf{A}_o$ and an iteration-dependent matrix $\mathbf{W}$. The matrix $\mathbf{W}$ is typically (but not universally) a matrix square root of the Hessian of some separable convex function. The randomized algorithms described in this section will only be useful for such problems when $\mathbf{W}$ and its adjoint can be applied to m-vectors in $O(m)$ time. This condition is satisfied in limited but important situations such as in algorithms for logistic regression, linear programming, and iteratively-reweighted least squares.

Stochastic Newton and subsampled Newton methods

Newton Sketch is a prototype algorithm developed over two papers [PW16; PW17] which is closely related to subsampled Newton methods [XRM17; YXR+18; RM19]. Each is suited to optimization problems that feature non-quadratic objective functions or problems with constraints other than linear equations. These methods entail sampling a new sketching operator (and applying it to a new data matrix) in

[3] That is, if we precondition a ridge regression problem, then it is necessary to precondition the *augmented matrix* $[\mathbf{A}; \sqrt{\mu}\mathbf{I}]$ in an unregularized version of the problem.

each iteration, with the aim of approximating the Hessian of the objective at the given iterate. The algorithms described in this section can easily serve as the main subroutine in Newton Sketch and subsampled Newton methods.

Newton Sketch has a natural specialization for least squares which entails sampling and applying only one sketching operator. This specialization can be viewed as sketch-and-precondition, where the iterative method for solving the saddle point system is based on preconditioned steepest-descent. The asymptotic convergence of this approach can be established in various ways [OPA19; LP19; Tro20]. It has been shown that "traditional" sketch-and-precondition methods (based on CG or the Chebyshev semi-iterative method) exhibit faster convergence [LP19]. Therefore we do not expect to incorporate this method into RandLAPACK.

There are two recently proposed extensions of Newton Sketch that may be suitable for solving the saddle point problems described in Section 3.1.2: Hessian averaging [NDM22] and stochastic variance reduced Newton (SVRN) [Der22b]. The performance profiles of these methods are better when $\mathbf{A}$ is very tall. When specialized to least squares, the former method amounts to preconditioned steepest-descent where the preconditioner is updated at each iteration. By comparison (again in the least squares setting), SVRN amounts to steepest-descent with a fixed preconditioner that incorporates variance-reduced sketching methods (adapted from [JZ13]) to approximate the gradient at each iteration.

Random features preconditioning for KRR

A random-features approach for computing accurate solutions to problems of the form (3.1) in the context of KRR is proposed in [ACW17b]. Specifically, [ACW17b] advocates for using random features to obtain a preconditioner for use in an iterative method such as PCG. Since any such iterative solver requires access to $\mathbf{G}$ by matrix-vector multiplication, Nyström PCG can be applied to the same problems as this *random-features preconditioning*. Empirical results strongly suggest that the Nyström approach has better performance than random-features preconditioning on shared-memory machines [FTU21]. Thus, we do not plan for RandLAPACK to support random-features preconditioning at this time.

Utilities for iterative refinement

Iterative refinement can be used as a tool to compensate for rounding errors in otherwise reliable linear system solvers. These methods typically work by computing residuals to higher precision than that used by the solver, running the solver with the updated residual, and then adding the new solution to the original solution [Bjö96, §2.9.2].[4] LAPACK has some procedures of this kind. See [Hig97] and [DHK+06] for theoretical and practical analyses.

The best way to use iterative refinement routines to support randomized algorithms in this section is yet to be determined. On the one hand, it may suffice to include methods similar to those in LAPACK. However, sketch-and-precondition algorithms might pose unique numerical problems that require different techniques. See [AMT10, §5.7] for some discussion of numerical issues in the sketch-and-precondition context. It might also be natural for a RandNLA library to have more iterative re-

[4] In some situations, it can suffice to recompute the residual with the same precision used by the underlying solver [Bjö96, §2.9.3].

finement methods than LAPACK in order to better exploit low-precision arithmetic and accelerators.

3.5 Existing libraries

We know of four high-performance libraries with sketch-and-precondition methods for least squares: Blendenpik [AMT10], LSRN [MSM14], LibSkylark [KAI+15], and Ski-LLS [CFS21].[5] To our knowledge, LibSkylark is the only RandNLA library which supports least squares *and* low-rank approximation (see Section 4.5). None of these libraries support saddle point problems of the kind we consider, and none of them make use of Nyström preconditioning.

Blendenpik. This library is written in C and callable from Matlab; it is currently available on the Matlab File Exchange. It uses LSQR as the deterministic iterative solver, and obtains the preconditioner by running QR on a sketch $\mathbf{A}^{\mathrm{sk}} = \mathbf{SA}$, where $\mathbf{S}$ is an SRFT. Blendenpik also adaptively calls LAPACK if a problem is deemed too poorly scaled or if the iterative method performs poorly. It was shown to outperform an unspecified LAPACK least squares solver on a machine with 8GB RAM and an AMD Opteron 242 processor [AMT10].

LSRN. This comprises a C++ implementation callable from Matlab and a Python implementation. The C++ implementation was shown to outperform LAPACK's `DGELSD` on large dense problems, and Matlab's backslash (SuiteSparseQR) on sparse problems. The Python implementation has demonstrated that LSRN scales well on Amazon Elastic Compute Cloud clusters. We note that the Python implementation relies on an auxiliary Python package with a custom C-extension for sampling from the Gaussian distribution via the ziggurat method.

LibSkylark. This library is written in C++ and is available on GitHub. Its support for least squares problems is very general and includes a few deterministic preconditioned iterative solvers. Its sketch-and-precondition functionality includes implementations in the styles of Blendenpik and LSRN. LibSkylark has a Python interface, but only for Python 2.7. Its linear algebra kernels are implemented partly in the Elemental distributed linear algebra library [PMG+13]. Unfortunately, Elemental is no longer maintained.

Ski-LLS. This is a recently developed C++ library for solving dense and sparse highly overdetermined least squares problems. It is distinguished by its flexibility in preconditioner generation. In particular, it supports sketching by SRFTs, Gaussian operators, and SASOs. It also supports factoring the sketch $\mathbf{SA}$ by several methods, including a standard SVD algorithm, a randomized algorithm for full-rank column-pivoted QR (see Section 5.1.2), and a standard algorithm for sparse QR. We record the following (adapted) quote from the GitHub repository that hosts this software:

> Ski-LLS is faster and more robust than Blendenpik and LAPACK on large over-determined data matrices, e.g., matrices having 40,000 rows and 4,000 columns. Ski-LLS is 10 times faster than Sparse QR and

[5] Note that "Blendenpik" and "LSRN" are names for algorithms *and* libraries.

incomplete-Cholesky preconditioned LSQR on sparse data matrices that are ill-conditioned and sufficiently large, e.g., with 120,000 rows, 5,000 columns, and 1% non-zeros.

Falkon. Finally, we note the recently developed *Falkon* library for sketch-and-solve approaches to KRR powered by multi-GPU machines [MCR+20; MCD+22]. While this library works outside of our primary data model, it is of interest to anyone developing software for KRR based on RandNLA.

Section 4

Low-rank Approximation

Modern scientific computing, machine learning, and data science applications generate massive matrices that need to be processed for reduced run time, reduced storage requirements, or improved interpretability. *Low-rank approximation* is a workhorse approach for achieving these goals. Here, given a target matrix $\mathbf{A}$, the task is to produce a suitably factored representation of a low-rank matrix $\hat{\mathbf{A}}$ of the same dimensions which approximates the matrix $\mathbf{A}$.

We can express the main aspects of a low-rank approximation as computing factor matrices $\mathbf{E}$ and $\mathbf{F}$ where

$$\underset{m \times n}{\mathbf{A}} \approx \underset{m \times n}{\hat{\mathbf{A}}} := \underset{m \times k}{\mathbf{E}} \; \underset{k \times n}{\mathbf{F}} \tag{4.1}$$

for some $k \ll \min\{m, n\}$. We note that it is very common to have a $k \times k$ "inner factor" that appears in between $\mathbf{E}$ and $\mathbf{F}$ above.

Such representations facilitate data interpretation by choosing the factors to have useful structure, such as having orthonormal columns or rows, or being submatrices of the target. The extent of storage reduction from low-rank approximation depends on whether $\mathbf{A}$ is dense or sparse. In the dense case, $\hat{\mathbf{A}}$ is stored in $O(mk + nk)$ space. In the sparse case, one representation consists of a dense $k \times k$ inner factor, a slice of k rows of $\mathbf{A}$, and a slice of k columns of $\mathbf{A}$.

The rank k used in a low-rank approximation is a tuning-parameter that the user can control to trade-off between approximation accuracy and data compression. The best choice of this parameter depends on context. For instance, one may want to choose k small enough to graphically visualize coherent structure in the target. In such a setting one would not expect that $\hat{\mathbf{A}}$ is close to $\mathbf{A}$ in an absolute sense, but one can still ask that the distance is near the minimum among all approximations with the desired structure and rank. Alternatively, one might know that $\mathbf{A}$ can be well-approximated by a low-rank matrix, and yet not know the rank necessary to achieve a good approximation. Such matrices are called *numerically low-rank* and arise in applications across the social, physical, biological, and ecological sciences. For example, they can arise as discretizations of differential operators, where the extent to which the matrix is numerically low-rank depends on the details of the operator and the discretization; and they can arise as noisy corruptions of general (hypothesized) data matrices with low exact rank. When dealing with such matrices one can iteratively build $\hat{\mathbf{A}}$ until a desired distance $\|\mathbf{A} - \hat{\mathbf{A}}\|$ is small. This section covers a variety of efficient and reliable low-rank approximation algorithms for both of these scenarios.

4.1 Problem classes

Low-rank approximation is naturally formalized as an optimization problem; one chooses $\hat{\mathbf{A}}$ and its factors to minimize a loss function subject to some constraints. The most common loss functions are distances $\hat{\mathbf{A}} \mapsto \|\mathbf{A} - \hat{\mathbf{A}}\|$ induced by the Frobenius or spectral norms. Alternatively, one can use the discontinuous loss function $\hat{\mathbf{A}} \mapsto \operatorname{rank}(\hat{\mathbf{A}})$ as a measure of the storage requirements for $\hat{\mathbf{A}}$. Constraints depend on the loss function in a complementary way. When minimizing a norm-induced distance, one imposes rank constraints by limiting the dimensions of the factors. When minimizing the rank of $\hat{\mathbf{A}}$ (i.e., when seeking an approximation that admits the smallest-possible representation) one constrains the approximation error $\|\mathbf{A}-\hat{\mathbf{A}}\|$ to be at most some specified value. One can also impose *structural* constraints on the factors of $\hat{\mathbf{A}}$, such as being orthogonal, diagonal, or a submatrix of the target.

Our overview of randomized algorithms for low-rank matrix approximation is organized around such structural constraints. Accordingly, we use the term *problem class* for loose groups of low-rank approximation problems wherein the factors facilitate similar downstream tasks. Currently, our two problem classes are the following.

- Spectral decompositions (§4.1.1): this consists of low-rank SVD and Hermitian eigendecomposition.
- Submatrix-oriented decompositions, i.e., decompositions with factors based on submatrices of the target matrix (§4.1.2): this consists of so-called *CUR* and *interpolative decompositions*.

Optimal decompositions in the first class often serve as baselines in theoretical analyses of randomized algorithms for low-rank decomposition in both classes. That is, such comparisons are made *regardless* of whether the approximation is spectral or submatrix-oriented. This fact can blur the distinction between the two problem classes, and the distinctions can blur even further when one considers methods for efficiently converting from one decomposition to another. Still, keeping the problem classes separate is useful as an organizing principle for the most fundamental low-rank approximation problems in RandNLA.

Remark 4.1.1. Low-rank approximations that impose no requirements on $\hat{\mathbf{A}}$'s representation are briefly addressed in Section 4.3.6 in the context of computational routines. Methods for low-rank approximation with other representations (e.g., QR, UTV, LU, nonnegative factorization) are discussed in Section 4.4.

4.1.1 Spectral decompositions

In what follows we give an overview of the SVD and Hermitian eigendecomposition, with emphasis on the roles of these decompositions in low-rank approximation. After covering these concepts we explain how they provide two perspectives on principal component analysis (PCA). We advise the reader to at least skim this overview material even if they are already familiar with the relevant concepts; low-rank approximation is much more prominent in RandNLA than it is in classical NLA.

Singular value decomposition

The SVD is widely used to compute low-rank approximations and as a workhorse algorithm for PCA. Given a $m \times n$ matrix $\mathbf{A}$, where $m \geq n$ (without loss of generality), its SVD is

$$\underset{m \times n}{\mathbf{A}} = \underset{m \times n}{\mathbf{U}} \quad \underset{n \times n}{\mathbf{\Sigma}} \quad \underset{n \times n}{\mathbf{V}^*}, \tag{4.2}$$

where $\mathbf{U} = [\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n]$ and $\mathbf{V} = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n]$ are column-orthonormal matrices that contain the left and right singular vectors of $\mathbf{A}$. The matrix $\mathbf{\Sigma} = \operatorname{diag}(\sigma_1, \ldots, \sigma_n)$ contains the corresponding singular values; we use the convention that they appear in decreasing order $\sigma_1 \geq \ldots \geq \sigma_n \geq 0$. We can also think about the SVD as expressing $\mathbf{A}$ as the sum of n rank-one matrices

$$\mathbf{A} = \sum_{i=1}^{n} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^*. \tag{4.3}$$

In applications it is common to encounter data matrices with low-rank structure, i.e., matrices for which $r = \operatorname{rank}(\mathbf{A})$ is smaller than the ambient dimensions m and n of $\mathbf{A}$. In this case, the singular values $\{\sigma_i : i \geq r+1\}$ are zero, the corresponding singular vectors span the left and right null spaces, and it is natural to consider the *compact SVD* where the sum in (4.3) is truncated at $i = r$. For a matrix $\mathbf{A}$ with *approximate* low-rank structure, we can obtain approximations with low *exact* rank by truncating this sum even earlier, at some $k < r$:

$$\begin{aligned} \mathbf{A} \approx \hat{\mathbf{A}}_k &:= \sum_{i=1}^{k} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^* \\ &= [\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k] \operatorname{diag}(\sigma_1, \ldots, \sigma_k) [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k]^* = \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^*, \end{aligned} \tag{4.4}$$

Truncating trailing singular values yields optimal rank-constrained approximations, in the sense of solving

$$\hat{\mathbf{A}}_k \in \operatorname*{arg\,min}_{\operatorname{rank}(\hat{\mathbf{A}}')=k} \|\mathbf{A} - \hat{\mathbf{A}}'\|. \tag{4.5}$$

This holds for every $k \in [\![r]\!]$. In other words, if $\mathbf{A}$ is approximated by a rank-k matrix $\hat{\mathbf{A}}_k$ given through its SVD, no further computation is needed to canonically obtain approximations of $\mathbf{A}$ with any rank $k \leq r$.

The optimality result of (4.5) holds for any unitarily invariant matrix norm, and it is known as the *Eckart-Young-Mirsky Theorem* when considered for the spectral norm or Frobenius norm. The reconstruction errors according to these norms are

$$\|\mathbf{A} - \hat{\mathbf{A}}_k\|_2 = \sigma_{k+1}(\mathbf{A}) \quad \text{and} \quad \|\mathbf{A} - \hat{\mathbf{A}}_k\|_{\mathrm{F}} = \sqrt{\sum_{j>k} \sigma_j^2(\mathbf{A})}. \tag{4.6}$$

These facts are important in applications, where it is common to see $\operatorname{rank}(\mathbf{A}) = \min\{m, n\}$ in exact arithmetic and yet many of the trailing singular values are so small that they can be presumed to be noise. That is, the truncation introduced in (4.4) is often used as a denoising technique.

Hermitian eigendecomposition

A matrix is called *Hermitian* if it is equal to its adjoint, i.e., if $\mathbf{A} = \mathbf{A}^*$. For real matrices, being Hermitian is the same as being symmetric. The eigendecomposition of a Hermitian matrix $\mathbf{A}$ is

$$\underset{n \times n}{\mathbf{A}} = \underset{n \times n}{\mathbf{V}} \quad \underset{n \times n}{\mathbf{\Lambda}} \quad \underset{n \times n}{\mathbf{V}^*}, \tag{4.7}$$

where $\mathbf{V}$ is an orthogonal matrix of eigenvectors and $\mathbf{\Lambda} = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$ is a real matrix containing the eigenvalues of $\mathbf{A}$. A Hermitian matrix is further called *positive semidefinite* (or "psd") if $\lambda_i \geq 0$ for all i.

We use the convention of sorting eigenvalues in decreasing order of absolute value: $|\lambda_1| \geq \cdots \geq |\lambda_n|$. This allows for a more direct comparison to the SVD, since we obtain low-rank approximations

$$\begin{aligned} \mathbf{A} \approx \hat{\mathbf{A}}_k &:= \sum_{i=1}^{k} \lambda_i \boldsymbol{v}_i \boldsymbol{v}_i^* \\ &= [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k] \operatorname{diag}(\lambda_1, \ldots, \lambda_k) [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k]^* = \mathbf{V}_k \mathbf{\Lambda}_k \mathbf{V}_k^* \end{aligned} \tag{4.8}$$

for which the spectral and Frobenius-norm distances to $\mathbf{A}$ match those from (4.6). Indeed, a (truncated) eigendecomposition can be converted to a (truncated) SVD by taking the columns of $\mathbf{V}$ as the right singular vectors, setting the left singular vectors according to

$$\boldsymbol{u}_i = \begin{cases} \boldsymbol{v}_i & \text{if } \lambda_i > 0 \\ -\boldsymbol{v}_i & \text{otherwise} \end{cases}$$

and setting the singular values to $\boldsymbol{\sigma} = |\boldsymbol{\lambda}|$ (elementwise).

If a matrix is Hermitian then it is better to compute (and work with) its eigendecomposition, rather than its SVD. The first reason for this is that a rank-k eigendecomposition requires almost half the storage of a rank-k SVD. The second reason

is that algorithms for computing low-rank eigendecompositions are able to leverage structure in the matrix for improved efficiency. These efficiency improvements can be dramatic for psd matrices, where an eigendecomposition is technically also an SVD.

Connections to principal component analysis

PCA is a linear dimension reduction technique that is widely used in data science applications for extracting features, or for visualizing and summarizing complicated datasets. The idea of PCA is to form k new variables (components) $\mathbf{Z} = [\boldsymbol{z}_1, \dots, \boldsymbol{z}_k]$ as linear combinations of the variables $\mathbf{X} = [\boldsymbol{x}_1, \dots, \boldsymbol{x}_n] \in \mathbb{R}^{m \times n}$ (that are assumed to have been preprocessed to have column-wise zero empirical mean). Specifically, given the data matrix $\mathbf{X}$, one forms the variables as $\mathbf{Z} = \mathbf{X}\mathbf{W}$ where the weights $\mathbf{W} = [\boldsymbol{w}_1, \dots, \boldsymbol{w}_k] \in \mathbb{R}^{n \times k}$ are chosen so that the first component $\boldsymbol{z}_1$ accounts for most of the variability in the data, the second component $\boldsymbol{z}_2$ for most of the remaining variability, and so on.

Formally, we can formulate this problem as a variance maximization problem

$$\boldsymbol{w}_1 := \operatorname*{arg\,max}_{\|\boldsymbol{w}\|_2^2 = 1} \operatorname{Var}(\mathbf{X}\boldsymbol{w}) \tag{4.9}$$

where we define the variance operator as $\operatorname{Var}(\mathbf{X}\boldsymbol{w}) := \frac{1}{m-1}\|\mathbf{X}\boldsymbol{w}\|_2^2$. Defining the sample covariance matrix $\mathbf{C} := \frac{1}{m-1}\mathbf{X}^*\mathbf{X}$, this problem can be stated as

$$\boldsymbol{w}_1 := \operatorname*{arg\,max}_{\|\boldsymbol{w}\|_2^2 = 1} \boldsymbol{w}^*\mathbf{C}\boldsymbol{w}. \tag{4.10}$$

We recognize (4.10) as the variational formulation of the dominant eigenvector of a Hermitian matrix. That is, $\boldsymbol{w}_1$ satisfies

$$\mathbf{C}\boldsymbol{w}_1 = \lambda_1(\mathbf{C})\boldsymbol{w}_1. \tag{4.11}$$

More generally, PCA finds the weights $\boldsymbol{w}_1, \dots, \boldsymbol{w}_k$ by diagonalizing the empirical sample covariance matrix $\mathbf{C}$ as $\mathbf{C} = \mathbf{W}\mathbf{\Lambda}\mathbf{W}^*$, and retaining only the top k eigenvectors.

How one computes the PCA depends on how the data is presented and accessed. There are two situations of interest. In situations where the covariance matrix $\mathbf{C}$ is given by the problem at hand—and thus where we access $\mathbf{C}$ directly, but do *not* directly access the data matrix $\mathbf{X}$—one can directly employ a low-rank Hermitian eigendecomposition to compute the dominant k eigenvectors. If instead we are presented with the variables in form of a mean-centered data matrix $\mathbf{X}$, then the low-rank SVD becomes the preferable approach to computing the weights $\mathbf{W}$. This is because we can relate the eigenvalue decomposition of the inner product $\mathbf{X}^*\mathbf{X}$ to the SVD of $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*$ by

$$\mathbf{X}^*\mathbf{X} = (\mathbf{V}\mathbf{\Sigma}\mathbf{U}^*)(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^*) = \mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^*. \tag{4.12}$$

Hence, we obtain the k weights $\mathbf{W} = [\boldsymbol{w}_1, \dots, \boldsymbol{w}_k]$ by computing the top k right singular vectors $\mathbf{V}_k = [\boldsymbol{v}_1, \dots, \boldsymbol{v}_k]$, and the eigenvalues of the sample covariance matrix $\mathbf{C}$ are given by the diagonal elements $\frac{1}{m-1}\mathbf{\Sigma}^2$.

Fast randomized algorithms for computing the SVD and eigenvalue decomposition enable scaling PCA to especially large problems. However, one does not need a large problem to benefit from these randomized algorithms. From 2016 until at least time of writing, the `pca` function `SciKit-Learn` defaults to a randomized algorithm when $d = \min\{m, n\} \geq 500$ and k is less than $0.8d$ [Gri16].

4.1.2 Submatrix-oriented decompositions

Here we describe four types of submatrix-oriented decompositions: a *CUR decomposition* and three types of *interpolative decompositions*. Historically, these have been used far less often than spectral decompositions. However, their value propositions have become much more compelling in recent years:

- They can offer reduced storage requirements compared to spectral decompositions, especially for sparse data matrices. This can be very valuable in processing massive data sets.
- They provide for more transparent data interpretation. This is especially true when data is modeled as a matrix as a matter of convenience, rather than as a statement about the data defining a meaningful linear operator $\mathbf{A} \mapsto \mathbf{A}\boldsymbol{v}$.

Remark 4.1.2. The following material is dense. We encourage the reader to return to it as needed while reading later parts of this section.

CUR decomposition

Definition. A CUR decomposition is a low-rank approximation of the form

$$\underset{m \times n}{\mathbf{A}} \approx \underset{m \times k}{\mathbf{C}} \underset{k \times k}{\mathbf{U}} \underset{k \times n}{\mathbf{R}}, \tag{4.13}$$

where the factors $\mathbf{C}$ and $\mathbf{R}$ are formed by small subsets of actual columns and rows of $\mathbf{A}$, and the *linking matrix* $\mathbf{U}$ is chosen so that some norm of $\mathbf{A} - \mathbf{CUR}$ is small.

Motivation. The literature on CUR decomposition traces back to work by Goreĭnov, Zamarashkin, and Tyrtyshnikov, who proved existential results for CUR decompositions with certain approximation error bounds [GZT95; GTZ97]. Goreĭnov et al. motivated their investigations by pointing out that CUR decompositions have far lower storage requirements than partial SVDs when $\mathbf{A}$ is sparse.More specifically, they advocated for applying CUR decompositions for low-rank approximation of off-diagonal blocks in block matrices.

The usage of CUR as a data-analysis tool was popularized by Mahoney and Drineas [MD09], following the development of efficient randomized algorithms for computing CUR decompositions with good approximation guarantees [DMM08]. The argument of Mahoney and Drineas was that experts often have a clear understanding of the actual meaning of certain columns and rows in a matrix, and this meaning is preserved by the CUR. In contrast, SVD (or PCA) forms linear combinations of the columns or rows of the input matrix; these linear combinations can prove difficult to interpret and destroy structures such as sparsity or nonnegativity.

Words of warning. Despite its simple definition, there are important subtleties in working with and understanding CUR decompositions.

First, CUR is unique among submatrix-oriented decompositions in that it involves taking products of submatrices of $\mathbf{A}$. Therefore if $\mathbf{A}$ has low numerical rank and its CUR decomposition is computed to high accuracy, then all three factors ($\mathbf{C}$, $\mathbf{U}$, and $\mathbf{R}$) will be ill-conditioned. This can have detrimental effects on numerical behavior, particularly in the computation of $\mathbf{U}$, which will behave qualitatively like inverting a matrix of low numerical rank.

Second, out of all the submatrix-oriented decompositions, CUR is of the least interest for providing exact decompositions of full-rank matrices. For example, if $\mathbf{A} = \mathbf{CUR}$ is a tall matrix of rank n, then we would necessarily have $\mathbf{C} = \mathbf{AP}$ and $\mathbf{U} = \mathbf{P}^*\mathbf{R}^{-1}$ for some permutation matrix $\mathbf{P}$. Therefore the only real freedom in full-rank CUR of a tall matrix is in choosing the spanning set of rows that define $\mathbf{R}$. A similar conclusion applies when $\mathbf{A}$ is a wide matrix of rank m; an exact decomposition would necessarily have $\mathbf{R} = \mathbf{P}^*\mathbf{A}$ and $\mathbf{U} = \mathbf{C}^{-1}\mathbf{P}$ for some permutation matrix $\mathbf{P}$, and the only real freedom would be in choosing the spanning set of columns that define $\mathbf{C}$.

The consequences of this second fact will be seen when we discuss randomized algorithms for computing CUR decompositions. In particular, we will see that randomized algorithms for (low-rank) CUR generally *do not* involve computing "full-rank CUR decompositions" on smaller matrices. This will stand in contrast to randomized algorithms for (low-rank) SVD and interpolative decompositions, which usually *do* involve computing the corresponding full-rank decomposition on smaller matrices.

Interpolative decompositions

Low-rank interpolative decompositions (IDs) come in three different flavors that share two common notes. The first shared note of these flavors is that exactly one of the factors that define $\hat{\mathbf{A}}$ is a submatrix of $\mathbf{A}$. The second shared note concerns the factors of $\hat{\mathbf{A}}$ that are not submatrices of $\mathbf{A}$. These factors, called *interpolation matrices*, are subject to certain regularity conditions.

Our crash course on ID comes in three parts. First, we define versions of ID that only involve one interpolation matrix, so-called *one-sided IDs*. After that, we explain how accuracy guarantees of low-rank ID are affected by regularity conditions on interpolation matrices. This explanation is important; we reference it repeatedly when we cover randomized algorithms for one-sided ID (§4.3.4). We wrap up by introducing the *two-sided ID*.

The one-sided IDs: column ID and row ID. In the low-rank case, a *column ID* is an approximation of the form

$$\underset{m \times n}{\mathbf{A}} \approx \underset{m \times k}{\mathbf{C}} \underset{k \times n}{\mathbf{X}} \tag{4.14}$$

where $\mathbf{C}$ is given by a small number of columns of $\mathbf{A}$ and $\mathbf{X}$ is a wide interpolation matrix. Full-rank column IDs can be of interest to us when $\mathbf{A}$ is very wide, in which case we have $k = m \ll n$ and $\mathbf{X} = \mathbf{C}^{-1}\mathbf{A}$. Next, we consider *row IDs*. In the low-rank case, these are approximations of the form

$$\underset{m \times n}{\mathbf{A}} \approx \underset{m \times k}{\mathbf{Z}} \underset{k \times n}{\mathbf{R}}. \tag{4.15}$$

where $\mathbf{R}$ is given by a small number of rows of $\mathbf{A}$ and $\mathbf{Z}$ is a tall interpolation matrix. The submatrices $\mathbf{C}$ and $\mathbf{R}$ can be represented by ordered column and row index sets, which we denote by J and I, that satisfy

$$\mathbf{C} = \mathbf{A}[:, J] \quad \text{and} \quad \mathbf{R} = \mathbf{A}[I, :].$$

These ordered index sets are called *skeleton indices*.

Our definitions of row and column IDs are not complete until we specify the regularity conditions on $\mathbf{X}$ and $\mathbf{Z}$. The skeleton indices (J, I) play a central role here. Most importantly, we require that the interpolation matrices satisfy

$$\mathbf{X}[:, J] = \mathbf{I}_k = \mathbf{Z}[I, :].$$

These regularity conditions have two direct consequences, namely

$$\begin{aligned}\mathbf{A}[:, J] &= \mathbf{A}[:, J]\mathbf{X}[:, J] \text{ , and} \\ \mathbf{A}[I, :] &= \mathbf{Z}[I, :]\mathbf{A}[I, :],\end{aligned}$$

In addition to these conditions, the literature on ID typically requires that the entries of $\mathbf{X}$ and/or $\mathbf{Z}$ are bounded in modulus by a small constant M. While we do not subscribe to this requirement in our definition of IDs, there is good motivation behind it. We address this motivation next.

Quality-of-approximation in low-rank IDs. Suppose that $\hat{\mathbf{A}}$ is a low-rank column ID of $\mathbf{A}$. It is immediate from our definition of low-rank ID that there are at most $\binom{n}{k}$ possible values for the subspace range($\hat{\mathbf{A}}$). In general, it can happen that none of these subspaces coincides with a dominant k-dimensional left singular subspace of $\mathbf{A}$. When this happens, it will necessarily be the case that $\|\mathbf{A} - \hat{\mathbf{A}}\|_2 > \sigma_{k+1}(\mathbf{A})$, whereas a general rank-k approximation could achieve an error equal to $\sigma_{k+1}(\mathbf{A})$. This raises a question.

> *How much of a price do we pay by imposing that requirement that* $\hat{\mathbf{A}}$ *be a low-rank ID?*

The following proposition (which we prove in Appendix C.1.1 by standard techniques) gives a partial answer to this question. The answer is notable in that it reveals the value of bounding the interpolation matrices.

Proposition 4.1.3. *Let* $\tilde{\mathbf{A}}$ *be any rank-k approximation of* $\mathbf{A}$ *that satisfies the spectral norm error bound* $\|\mathbf{A} - \tilde{\mathbf{A}}\|_2 \leq \epsilon$. *If* $\tilde{\mathbf{A}} = \tilde{\mathbf{A}}[:, J]\mathbf{X}$ *for some* $k \times n$ *matrix* $\mathbf{X}$ *and an index vector* J, *then* $\hat{\mathbf{A}} = \mathbf{A}[:, J]\mathbf{X}$ *is a low-rank column ID that satisfies*

$$\|\mathbf{A} - \hat{\mathbf{A}}\|_2 \leq (1 + \|\mathbf{X}\|_2)\epsilon. \tag{4.16}$$

Furthermore, if $|X_{ij}| \leq M$ *for all* (i, j), *then*

$$\|\mathbf{X}\|_2 \leq \sqrt{1 + M^2 k(n - k)}. \tag{4.17}$$

We note that the assumptions on the matrix $\tilde{\mathbf{A}}$ from the proposition statement imply that $\tilde{\mathbf{A}}[:, J]$ is full column-rank. This implies that $(\tilde{\mathbf{A}}[:, J])^\dagger(\tilde{\mathbf{A}}[:, J]) = \mathbf{I}_k$ and hence that $\mathbf{X}$ is completely determined from the index vector J. Since a rank-k matrix will typically have multiple subsets of k columns that span its range, we finally see that the matrix $\tilde{\mathbf{A}}$ does not uniquely determine the interpolation matrix $\mathbf{X}$ used in the proposition. Indeed, the range of possibilities for the interpolation matrix is rather remarkable. While it is known that there always exists an index set for which $\max_{ij} |X_{ij}| = 1$ [Pan00], it is NP-hard to compute this index set [ÇM09]. At the same time, algorithms such as strong rank-revealing QR can be applied to $\tilde{\mathbf{A}}$ with typical runtime $O(mnk)$, while ensuring $\max_{ij} |X_{ij}| \leq 2$ [GE96].

All-in-all, the main point of an interpolative decomposition is to provide a low-rank approximation that prominently features a submatrix of the target. Therefore while an upper bound M on the entries of an interpolation matrix gives the bound (4.16) indirectly by way of (4.17), such a bound should not the end of the story. The spectral norm $\|\mathbf{X}\|_2$ is ultimately more informative for this purpose, even if it is harder to compute.

Of course, all of the points we have raised here for column IDs apply to row IDs with minor modifications.

Two-sided ID. The concept of a one-sided ID can be extended to a *two-sided ID* by considering simultaneous row and column IDs. In the low-rank case, a two-sided ID is an approximation of the form

$$\underset{m\times n}{\mathbf{A}} \approx \underset{m\times k}{\mathbf{Z}} \underset{k\times k}{\mathbf{A}[I,J]} \underset{k\times n}{\mathbf{X}}. \tag{4.18}$$

Methods for full-rank one-sided ID can be useful in computing low-rank two-sided IDs. That is, if we first compute a low-rank column ID $\hat{\mathbf{A}} = \mathbf{CX}$ by some black-box method, then after obtaining a full-rank row ID of the tall matrix $\mathbf{C} = \mathbf{A}[:, J] = \mathbf{ZA}[I, J]$ we have $\hat{\mathbf{A}} = \mathbf{ZA}[I, J]\mathbf{X}$.

On relationships between submatrix-oriented decompositions

The landscape of methods for submatrix-oriented decompositions is very intertwined. As we have already indicated, algorithms for one-sided ID are the main building blocks of algorithms for low-rank two-sided ID. Later in this section we also mention several algorithms for CUR decomposition which depend on algorithms for one-sided ID. In view of this, we emphasize the following point.

> For our purposes, it is helpful to introduce hierarchical relationships among submatrix-oriented decompositions. As such, we designate algorithms for one-sided ID as *computational routines*, while methods for CUR and two-sided ID are designated as *drivers*.

Next, let us point out a sense in which CUR and two-sided ID are "dual" to one another. Both are submatrix-oriented decompositions that have three factors. For CUR, the outer factors are submatrices of $\mathbf{A}$ and no requirements are placed on the inner factor (the linking matrix). In particular, if we specify the outer factors by ordered index sets J and I, then a CUR decomposition is expressed as

$$\underset{m\times n}{\mathbf{A}} \approx \underset{m\times k}{\mathbf{A}[:,J]} \underset{k\times k}{\mathbf{U}} \underset{k\times n}{\mathbf{A}[I,:]}. \tag{4.19}$$

This can be contrasted with two-sided ID as defined in (4.18). There, the inner factor is a submatrix of $\mathbf{A}$, and moderate requirements are placed on the outer factors (the interpolation matrices).

The properties of "linking matrices" and "interpolation matrices" are different enough to warrant their different names. The problem of computing a low-rank approximation via two-sided ID can be better numerically behaved, compared to low-rank approximation by CUR [MT20, §13]. However, CUR offers far greater potential for storage reduction when dealing with sparse matrices. The differences

between two-sided ID and CUR can become less pronounced when one considers methods for losslessly converting one such representation to the other, as we mention in Section 4.2.3.

4.1.3 On accuracy metrics

The problems from Section 3 were mostly unconstrained minimization of convex quadratics. Such problems are very nice, since the gradient of the quadratic loss function constitutes a canonical error metric that can be driven to zero. Low-rank approximation problems can likewise be framed as optimization problems. However, these formulations either involve constraints or a nonconvex objective function. This distinction is important, since these structures rule out checking for a zero gradient as a cheap optimality condition.

The main error metrics in low-rank approximation are norm-induced distances. For reasons that we give under the next two headings it is not appropriate to consider distances from a computed approximation $\hat{\mathbf{A}}$ to some nominally "optimal" approximation. Instead, one measures the distance from the approximation to the target, most often in the spectral or Frobenius norms.

Distance to optimal approximations

Non-unique solutions and sensitivity to perturbations. Recall from Section 4.1.1 how truncating $\mathbf{A}$'s SVD at rank k gives an optimal rank-k approximation in any unitarily invariant norm. Unfortunately, this truncation will be non-unique when $\mathbf{A}$ has more than one singular value equal to σ_k. This is easiest to see when $\mathbf{A} = \mathbf{I}$ is the identity matrix, in which case every diagonal $\{0,1\}$-matrix of rank k is an optimal rank-k approximation to $\mathbf{A}$.

More generally, if $\mathbf{A}$ has multiple singular values that are *close to* σ_k, then extremely small perturbations to $\mathbf{A}$ can result in large changes to the singular vectors corresponding to these singular values; see [Bha97, §6 – §8] for details. This has a secondary complication: it is harder to estimate the dominant k singular vectors of a matrix than it is to find a rank-k approximation that is "near optimal" in the sense of (4.5).

Intractability of computing optimal approximations. When working with submatrix-oriented decompositions, we do not even have the luxury of defining "optimal" approximations in the manner of truncated SVDs. Indeed, the problem of finding an "optimal" ID necessitates specifying any regularity conditions such as the bound M in a constraint $|X_{ij}| \leq M$. As we mentioned before, even when $\hat{\mathbf{A}}$ has exact rank k, a rank-k column ID with $M = 1$ always exists but is NP-hard to find [ÇM09].

Going to another extreme, we could set aside the matter of M and simply set $\mathbf{X} = \mathbf{C}^{\dagger}\mathbf{A}$ for a matrix $\mathbf{C}$ containing k columns of $\mathbf{A}$. In this case it is not known if the columns can be chosen to minimize Frobenius- or spectral-norm error $\|\hat{\mathbf{A}} - \mathbf{A}\|$ in time less than $O(n^k)$. Still, there are theoretical guarantees for approximation quality by CUR relative to approximation quality achievable by SVD. We refer the reader to [DMM08; BMD09], [VM16, §1 - §2], and Appendix C.1.1 for more information about CUR and ID in our context.

Distance relative to that of a reference approximation

It is problematic to use a distance from $\hat{\mathbf{A}}$ to $\mathbf{A}$ as an error metric for $\hat{\mathbf{A}}$. This is because there are situations when any such distance will be large even when $\hat{\mathbf{A}}$ is close to an "optimal" approximation. The simplest example of this is PCA, in which cases the approximation rank is $O(1)$, independent of the dimensions of the matrix or properties of its spectrum. More generally, it can be hard to obtain a low-rank approximation that is very close to $\mathbf{A}$ when $\mathbf{A}$ has *slow spectral decay*, in the sense that the distribution of its singular values has a heavy tail. Accurate approximations can also be hard to come by if the factors of $\hat{\mathbf{A}}$ are highly constrained.

One handles this situation by considering the distance between $\mathbf{A}$ and $\hat{\mathbf{A}}$ *relative to* that between $\mathbf{A}$ and some reference matrix $\mathbf{A}_r$. Formally, we concern ourselves with the smallest value of ϵ needed to achieve

$$\|\mathbf{A} - \hat{\mathbf{A}}\| \leq (1 + \epsilon)\|\mathbf{A} - \mathbf{A}_r\|.$$

The reference matrices $\mathbf{A}_r$ used in RandNLA theory are not available to us when performing computations. In fact, they are usually not optimal for the formal low-rank approximation problem at hand. The most common source of non-optimality is that the reference is subject to a more stringent rank constraint: $\text{rank}(\mathbf{A}_r) < \text{rank}(\hat{\mathbf{A}}) \leq \text{rank}(\mathbf{A})$. Another source of non-optimality is that it may not be possible to decompose the reference into factors with the required structure (e.g., the structure required by low-rank CUR). For example, an approximation of $\mathbf{A}$ obtained by a rank-k truncated SVD cannot (in general) be converted into a rank-k CUR decomposition using submatrices of $\mathbf{A}$.

4.2 Drivers

There exist many randomized algorithms for computing low-rank approximations of matrices. This section focuses on low-rank approximation algorithms that take the two-stage approach popularized by [HMT11], because this approach has been demonstrated to be efficient and highly reliable over the years. The high-level idea of the two-stage approach is the following: first one constructs a "simple" representation of $\hat{\mathbf{A}}$ with the aid of randomization, and then one deterministically converts that representation of $\hat{\mathbf{A}}$ into a more useful form.

In order to discuss these drivers for low-rank approximation, it is necessary to mention briefly the following two concepts (these are handled by computational routines, to be discussed in detail in Section 4.3):

- A *QB decomposition* is a simple representation that is useful for SVD and eigendecomposition. The representation takes the form $\hat{\mathbf{A}} = \mathbf{QB}$ for a tall matrix $\mathbf{Q}$ with orthonormal columns and $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$. The important point here is that the QB decomposition involves explicit construction of and access to both $\mathbf{Q}$ and $\mathbf{B}$. We discuss QB algorithms in Section 4.3.2.

- *Column subset selection* (CSS) is the problem of selecting from a matrix a set of columns that is "good" in some sense. CSS algorithms largely characterize algorithms for *one-sided ID*. We discuss methods for these two problems in Section 4.3.4. They are important here because a one-sided ID can be used for the simple representation of $\hat{\mathbf{A}}$ when working toward an SVD, eigendecomposition, two-sided ID, or CUR decomposition.

4.2.1 Methods for SVD

The are several families of randomized algorithms for computing low-rank SVDs. Here we describe a few families that give $\hat{\mathbf{A}} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*$ through its compact SVD.

A flexible method

We begin with Algorithm 3. This algorithm uses randomization to compute a QB decomposition of $\mathbf{A}$, then deterministically computes $\mathbf{QB}$'s compact SVD, and finally truncates that SVD to a specified rank.

This algorithm assumes that the `QBDecomposer` is iterative in nature. It assumes that each iteration adds some number of columns to $\mathbf{Q}$ and rows to $\mathbf{B}$, and that the algorithm can terminate once an implementation-dependent error metric for $\mathbf{QB} \approx \mathbf{A}$ falls below ϵ or once $\mathbf{QB}$ reaches a rank limit. Here we have set the rank limit to $k + s$ where s is a nonnegative "oversampling parameter."

Algorithm 3 `SVD1` : QB-backed low-rank SVD (see [HMT11] and [RST10])

1: **function** `SVD1`$(\mathbf{A}, k, \epsilon, s)$

Inputs:

$\mathbf{A}$ is an $m \times n$ matrix. The returned approximation will have rank *at most* k. The approximation produced by the randomized phase of the algorithm will attempt to $\mathbf{A}$ to within ϵ error, but will not produce an approximation of rank greater than $k + s$.

Output:

The compact SVD of a low-rank approximation of $\mathbf{A}$.

Abstract subroutines:

`QBDecomposer` generates a QB decomposition of a given matrix; it tries to reach a prescribed error tolerance but may stop early if it reaches a prescribed rank limit.

2: $\mathbf{Q}, \mathbf{B} =$ `QBDecomposer`$(\mathbf{A}, k + s, \epsilon)$ # $\mathbf{QB} \approx \mathbf{A}$
3: $r = \min\{k,$ number of columns in $\mathbf{Q}\}$
4: $\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}^* =$ `svd`$(\mathbf{B})$
5: $\mathbf{U} = \mathbf{U}[: , : r]$
6: $\mathbf{V} = \mathbf{V}[: , : r]$
7: $\mathbf{\Sigma} = \mathbf{\Sigma}[: r , : r]$
8: $\mathbf{U} = \mathbf{QU}$
9: **return** $\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}^*$

The literature recommends setting s to a small positive number (e.g., $s = 5$ or $s = 10$) to account for the fact that the trailing singular vectors of $\mathbf{QB}$ may not be good estimates for the corresponding singular vectors of $\mathbf{A}$. However, using any positive oversampling parameter complicates the interpretation of the error tolerance ϵ. If a user deems this problematic then they can simply set $k \leftarrow k + s$ and $s \leftarrow 0$. Such an approach can be reasonable if tuning parameters for the QB algorithm are chosen appropriately. Specifically, if techniques such as power iteration are used (see Section 4.3.1) then the trailing singular vectors of $\mathbf{QB}$ can be reasonably good approximations to the corresponding singular vectors of $\mathbf{A}$.

Sacrificing accuracy for speed

Converting from an ID. Setting our sights beyond Algorithm 3, it is noteworthy that if $\hat{\mathbf{A}}$ is given in *any* compact representation then it can be losslessly converted into an SVD without ever accessing $\mathbf{A}$. For example, given a column ID $\hat{\mathbf{A}} = \mathbf{CX}$, we would compute a QR decomposition $\mathbf{C} = \mathbf{QR}$, set $\mathbf{B} = \mathbf{RX}$, and then proceed with $(\mathbf{Q}, \mathbf{B})$ as in Algorithm 3. As another example, conversion from a row ID to an SVD is illustrated implicitly in [HMT11, Algorithm 5.2].

Such approaches are potentially useful because one-sided ID can easily be implemented in a way that accesses $\mathbf{A}$ with a single matrix-matrix multiplication and then selects a row or column submatrix. However, this comes at a cost of a much less accurate solution compared to typical QB methods.

Single-pass algorithms. For very large problems the main measure of an algorithm's complexity is the number of times it moves $\mathbf{A}$ through fast memory. Besides the above ID-based method, there are three algorithms for low-rank SVD which move $\mathbf{A}$ through fast memory only once. Each of them uses multi-sketching in the sense of Section 2.6. The first and second options simply use Algorithm 3, but specifically with single-pass QB methods based on type 1 or type 2 multi-sketching. Discussion of such QB algorithms is deferred to Section 4.3.2. The third option is the algorithm described in [TYU+17b, §7.3.2], which relies on type 3 multi-sketching.

Remark 4.2.1. Algorithms designed to minimize the number of views of a matrix are usually analyzed in the *pass efficient model* for algorithm complexity [DKM06a]. Early work on randomized pass-efficient and single-pass algorithms can be found in [FKV04; DKM06a; DKM06b].

Error estimation in sketched one-sided SVD. Single-pass algorithms are unlikely to produce highly accurate approximations of singular vectors or singular values. However, their results may be accurate enough to be useful in certain applications. This motivates methods for estimating the errors of approximations returned by these algorithms. Appendix E.3 provides a bootstrap-based error estimator for a simple randomized algorithm that recovers approximate singular vectors from one side of the matrix.

4.2.2 Methods for Hermitian eigendecomposition

Each randomized algorithm for low-rank SVD has a corresponding version that is specialized to Hermitian matrices. We recount those specialized algorithms here, and we mention an additional algorithm that is unique to the approximation of psd matrices. In general, we shall say that $\mathbf{A}$ is $n \times n$ and that the algorithms represent $\hat{\mathbf{A}} = \mathbf{V}\,\mathrm{diag}(\boldsymbol{\lambda})\mathbf{V}^*$, where $\mathbf{V}$ is a tall column-orthonormal matrix and $\boldsymbol{\lambda}$ is a vector with entries sorted in decreasing order of absolute value.

A flexible method for Hermitian indefinite matrices

Algorithm 4 is a variation of [HMT11, Algorithm 5.3]. Its parameters (k, ϵ, s) have essentially the same interpretations as Algorithm 3. When $s = 0$, its output is simply is a compact eigendecomposition of $\mathbf{QCQ}^*$, where $\mathbf{C} = \mathbf{Q}^*\mathbf{AQ}$ and $\mathbf{Q}$ is obtained from a black-box `QBDecomposer`. The main difference between this method and Algorithm 3 is that ϵ is scaled down by a factor $1/2$ before being passed to

`QBDecomposer`. This change is needed to so that if $s = 0$ and $\|\mathbf{QB} - \mathbf{A}\| \leq \epsilon$ then the final approximation satisfies $\|\hat{\mathbf{A}} - \mathbf{A}\| \leq \epsilon$; see [HMT11, §5.3].

Algorithm 4 `EVD1` : QB-backed low-rank eigendecomposition; see [HMT11]

function `EVD1`($\mathbf{A}, k, \epsilon, s$)

Inputs:

$\mathbf{A}$ is an $n \times n$ Hermitian matrix. The returned approximation will have rank *at most* k. The approximation produced by the randomized phase of the algorithm will attempt to $\mathbf{A}$ to within ϵ error, but will not produce an approximation of rank greater than $k + s$.

Output:

Approximations of the dominant eigenvectors and eigenvalues of $\mathbf{A}$.

Abstract subroutines:

`QBDecomposer` generates a QB decomposition of a given matrix; it tries to reach a prescribed error tolerance but may stop early if it reaches a prescribed rank limit.

$\mathbf{Q}, \mathbf{B} =$ `QBDecomposer`($\mathbf{A}$, $k + s$, $\epsilon/2$)
$\mathbf{C} = \mathbf{BQ}$ # since $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$, we have $\mathbf{C} = \mathbf{Q}^*\mathbf{AQ}$
$\mathbf{U}, \boldsymbol{\lambda} =$ `eigh`($\mathbf{C}$) # full Hermitian eigendecomposition
$r = \min\{k,$ number of entries in $\boldsymbol{\lambda}\}$
$P = \text{argsort}(|\boldsymbol{\lambda}|)[:r]$
$\mathbf{U} = \mathbf{U}[:, P]$
$\boldsymbol{\lambda} = \boldsymbol{\lambda}[P]$
$\mathbf{V} = \mathbf{QU}$
return $\mathbf{V}, \boldsymbol{\lambda}$

Sacrificing accuracy for speed with Hermitian indefinite matrices

Converting from an ID. [HMT11, Algorithm 5.4] is a second approach to Hermitian eigendecomposition, based on postprocessing a low-rank row ID of $\mathbf{A}$. We do not include pseudocode for this algorithm in this monograph. However, the basic observation underlying the approach is that one can use the symmetry of $\mathbf{A}$ to canonically approximate an initial row ID $\tilde{\mathbf{A}} = \mathbf{ZA}[I, :] \approx \mathbf{A}$ by the Hermitian matrix $\hat{\tilde{\mathbf{A}}} = \mathbf{ZA}[I, I]\mathbf{Z}^*$. The compact representation of this Hermitian matrix makes it easy to compute its eigendecomposition by a lossless process.

When should one use [HMT11, Algorithm 5.4] over Algorithm 4? Our answer is the same as for using [HMT11, Algorithm 5.2] over Algorithm 3. That is, the ID approach is only of interest when it moves $\mathbf{A}$ through fast memory *once*, and it should be considered alongside other low-rank eigendecomposition algorithms with similar data movement patterns.

Single-pass algorithms. Just as with fast algorithms for low-rank SVD, one can obtain fast algorithms for low-rank Hermitian eigendecomposition by using Algorithm 4 with the QB methods based on type 1 or type 2 multi-sketching.

Besides those approaches, we make note of [HMT11, Algorithm 5.6], which accesses $\mathbf{A}$ *exclusively* through a single sketch $\mathbf{Y} = \mathbf{AS}$ and makes no assumptions

on the representation of $\mathbf{A}$ in-memory. This access pattern is possible because the algorithm solves a least squares problem involving $\mathbf{Y}$, $\texttt{orth}(\mathbf{Y})$, and $\mathbf{S}$ to project a small $k \times k$ "core matrix" onto the set of Hermitian matrices.

Nyström approximations for positive semidefinite matrices

Now we suppose that the $n \times n$ matrix $\mathbf{A}$ is psd, in which case we can define the *Nyström approximation of* $\mathbf{A}$ *with respect to a matrix* $\mathbf{X}$ as

$$\hat{\mathbf{A}} = (\mathbf{AX}) \, (\mathbf{X}^* \mathbf{AX})^\dagger \, (\mathbf{AX})^*. \tag{4.20}$$

When framed this way, the Nyström approximation is defined for any matrix $\mathbf{X}$ with $k \leq n$ columns. Indeed, it does not even presume that $\mathbf{X}$ is random. However, in RandNLA, we ultimately set $\mathbf{X}$ to a sketching operator and produce a compact spectral decomposition $\hat{\mathbf{A}} = \mathbf{V} \operatorname{diag}(\boldsymbol{\lambda}) \mathbf{V}^*$. For any given type of sketching operator, low-rank approximation of psd matrices by Nyström approximations tend to be more accurate than approximations produced by comparable algorithms for general Hermitian eigendecomposition.

What's in a name? Disambiguating "Nyström approximation." The literature on randomized algorithms for Nyström approximation *heavily* emphasizes using column selection operators [WS00; DM05; Pla05; KMT09a; KMT09b; LKL10; Bac13; GM16; RCR15; DKM20]. This stems from an analogy between sampling columns of kernel matrices in machine learning and the Nyström method from integral equation theory. See [DM05, §5] for a detailed discussion. In view of the prominence of column sampling in Nyström approximations, part of Section 6 is dedicated to specialized methods for sampling columns from psd matrices.

When $\mathbf{X}$ is a general sketching operator, the approximation (4.20) has been called a "projection-based SPSD approximation" [GM16]. The different terminology for general sketching operators $\mathbf{X}$ is helpful for distinguishing the resulting approximations from those referred to as "Nyström approximations" in the machine learning literature. However, it is not in line with our philosophy that RandNLA concepts should be described with minimal assumptions on the nature of the sketching distribution. This philosophy, first advocated for by Drineas and Mahoney [DMM+11; MD16], leads us to adopt the following convention.

> *The term "Nyström approximation" shall be used for any approximation of the form (4.20), even when* $\mathbf{X}$ *is a general sketching operator.*

We note that this also follows the convention used by El Alaoui and Mahoney in their work on kernel ridge regression (see [AM15, Theorem 1]).

Algorithms. Algorithm 5 is a practical approach to low-rank eigendecomposition by Nyström approximation. It includes a function call $\texttt{TallSketchOpGen}(\mathbf{A}, k+s)$ which returns a sketching operator $\mathbf{S}$ with n rows and $k+s$ columns. Our reason for specifying a sketching operator in this way is to provide flexibility in whether the sketching operator is data-oblivious or data-aware. In this context, the main type of data-aware sketching operator would be based on so-called *power iteration*; see Section 4.3.1 and [GM16].

Algorithm 5 `EVD2` : for psd matrices only; adapts [TYU+17a, Algorithm 3]

function `EVD2`($\mathbf{A}, k, s$)

Inputs:

$\mathbf{A}$ is an $n \times n$ psd matrix. The returned approximation will have rank at most k, but the sketching operator used in the algorithm can have rank as high as $k + s$.

Output:

Approximations of the dominant eigenvectors and eigenvalues of $\mathbf{A}$.

Abstract subroutines:

`TallSketchOpGen` generates a sketching operator with a prescribed number of columns, for use in sketching a given matrix from the right.

$\mathbf{S} = \texttt{TallSketchOpGen}(\mathbf{A}, k + s)$
$\mathbf{Y} = \mathbf{AS}$
$\nu = \sqrt{n} \cdot \epsilon_{\text{mach}} \cdot \|\mathbf{Y}\|$ # ϵ_{mach} is machine epsilon for current numeric type
$\mathbf{Y} = \mathbf{Y} + \nu\mathbf{S}$ # regularize for numerical stability
$\mathbf{R} = \texttt{chol}(\mathbf{S}^*\mathbf{Y})$ # $\mathbf{R}$ is upper-triangular and $\mathbf{R}^*\mathbf{R} = \mathbf{S}^*\mathbf{Y} = \mathbf{S}^*(\mathbf{A} + \nu\mathbf{I})\mathbf{S}$
$\mathbf{B} = \mathbf{Y}(\mathbf{R}^*)^{-1}$ # $\mathbf{B}$ has n rows and $k + s$ columns
$\mathbf{V}, \mathbf{\Sigma}, \mathbf{W}^* = \texttt{svd}(\mathbf{B})$ # can discard $\mathbf{W}$
$\boldsymbol{\lambda} = \text{diag}(\mathbf{\Sigma}^2)$ # extract the diagonal
$r = \min\{k,$ number of entries in $\boldsymbol{\lambda}$ that are greater than $\nu\}$
$\boldsymbol{\lambda} = \boldsymbol{\lambda}[:r] - \nu$ # undo regularization
$\mathbf{V} = \mathbf{V}[:, :r]$.
return $\mathbf{V}, \boldsymbol{\lambda}$

The role of the parameter s in Algorithm 5 is analogous to that in Algorithms 3 and 4 – the algorithm effectively computes data needed for a rank $k + s$ eigendecomposition before truncating that approximation to rank k. However, unlike Algorithms 3 and 4, Algorithm 5 has no control of approximation error. We refer the reader to [FTU21, Algorithm E.2] for a more sophisticated version of this algorithm which can accept an error tolerance.

4.2.3 Methods for CUR and two-sided ID

Here we describe two approaches to CUR and one approach to two-sided ID. The descriptions are largely qualitative in that they are stated in terms of algorithms for low-rank column ID and CSS (which are detailed in Section 4.3.4).

CUR by falling back on CSS

Perhaps the simplest approach to CUR computes the row and column indices (I, J) in one stage and then computes the linking matrix $\mathbf{U}$ in a second stage. The column indices J are obtained by a randomized algorithm for CSS on $\mathbf{A}$, then the row indices I are obtained by some CSS algorithm on $\mathbf{C}^* = \mathbf{A}[:, J]^*$.[1] Because the matrix $\mathbf{C}$ is

[1] Of course, this process could be reversed to compute I and then J.

so much smaller than $\mathbf{A}$, it is often practical to use a deterministic algorithm when performing CSS on $\mathbf{C}^*$.

There are two canonical choices for the linking matrix in this context: one obtained by projection

$$\mathbf{U}_{\text{proj}} = (\mathbf{A}[:, J])^\dagger \, \mathbf{A} \, (\mathbf{A}[I, :])^\dagger$$

and one obtained by submatrix inversion

$$\mathbf{U}_{\text{sub}} = \mathbf{A}[I, J]^\dagger.$$

It should be clear that the approximation error incurred by using the former matrix will never be larger than when using the latter. Furthermore, the process of computing the former matrix is better conditioned than the process of computing the latter. Therefore it is generally preferable to use $\mathbf{U}_{\text{proj}}$ as the linking matrix when implementing CUR via randomized CSS.

Remark 4.2.2. Randomized algorithms for CUR based on the pattern above were first proposed in [DMM08; BMD09], particularly with linking matrices closer to the form $\mathbf{U}_{\text{sub}}$. Deterministic analyses of CUR approximation quality with various linking matrices can be found in [GZT95; GTZ97].

CUR by a combination of column ID and CSS

Suppose we have access to data $(\mathbf{X}, J)$ from a column ID of an initial low-rank approximation of $\mathbf{A}$. Given this data, we can recover the row index set I and $\mathbf{U}$ for a CUR decomposition by running CSS on $\mathbf{C}^* = \mathbf{A}[:, J]^*$ and setting $\mathbf{U} = \mathbf{X} \, (\mathbf{A}[I, :])^\dagger$.

This approach only requires the application of one pseudo-inverse, which compares favorably to the two applications of pseudo-inverses needed to compute $\mathbf{U}_{\text{proj}}$ in the first approach to CUR. At the same time, if the randomized algorithm for computing $(\mathbf{X}, J)$ happens to return an interpolation matrix satisfying $\mathbf{X} = \mathbf{C}^\dagger \mathbf{A}$, then the resulting decomposition could have been obtained by the elementary CUR algorithm with linking matrix $\mathbf{U}_{\text{proj}}$. Therefore there is a sense in which this template algorithm generalizes the elementary approach to CUR.

We instantiate this template algorithm in Algorithm 6. The CSS step of the algorithm calls a deterministic function for computing a QR decomposition with column pivoting, with the semantics indicated in Table 1.1. Whether the algorithm starts with a row ID or column ID depends on the aspect ratio of the data matrix; [MT20, §13.3] recommend starting with a row ID when $\mathbf{A}$ is wide in the related context of computing two-sided IDs.

Two-sided ID via one-sided ID

Two-sided IDs are canonically computed by a simple reduction to one-sided ID: first obtain $(\mathbf{X}, J)$ by a column ID of $\mathbf{A}$ and then obtain $(I, \mathbf{Z})$ by a row ID of $\mathbf{A}[:, J]$. The initial column ID of $\mathbf{A}$ will be computed by a randomized algorithm and hence will always be low-rank. However, it is not expensive to compute a *full-rank* row ID $\mathbf{A}[:, J] = \mathbf{Z}\mathbf{A}[I, J]$ by a deterministic method under the standard assumption that $|J| \ll \min\{m, n\}$. Such an approach is described in [VM16, §2.4, §4].

Finally, we note that a two-sided ID can be naturally repurposed for CUR decomposition by either of the two qualitative approaches to CUR described above. In the first case one only needs the index sets (I, J) and computes the linking matrix by any desired method. In the second case one needs the index sets *and* one of the

Algorithm 6 `CURD1` : CUR by randomizing an initial ID [VM16; DM21b]

function `CURD1`$(\mathbf{A}, k, s)$

Inputs:

$\mathbf{A}$ is an $m \times n$ matrix. The returned approximation will have rank at most k. The `ColumnID` abstract subroutine can use sketching operators of rank up to $k + s$ in its internal calculations.

Output:

A low-rank CUR decomposition of $\mathbf{A}$.

Abstract subroutines:

`ColumnID` produces a low-rank column ID of a given matrix, up to some specified rank.

if $m \geq n$ **then**
  $\mathbf{X}, J = \texttt{ColumnID}(\mathbf{A},\ k,\ s)$ # $|J| = k$ and $\mathbf{A}[:, J]\mathbf{X} \approx \mathbf{A}$
  $\mathbf{Q}, \mathbf{T}, I = \texttt{qrcp}(\mathbf{A}[:, J]^*)$ # only care about the indices I
  $I = I[: k]$
  $\mathbf{U} = \mathbf{X}\,(\mathbf{A}[I, :])^\dagger$
else
  $\mathbf{Z}^*, I = \texttt{ColumnID}(\mathbf{A}^*,\ k,\ s)$ # $|I| = k$ and $\mathbf{ZA}[I, :] \approx \mathbf{A}$.
  $\mathbf{Q}, \mathbf{T}, J = \texttt{qrcp}(\mathbf{A}[I, :])$ # only care about the indices J
  $J = J[: k]$
  $\mathbf{U} = (\mathbf{A}[:, J])^\dagger\,\mathbf{Z}$
return $J, \mathbf{U}, I$

interpolation matrices (i.e., one of $\mathbf{Z}$ or $\mathbf{X}$). The latter approach for converting from two-sided ID to CUR is used in [VM16, §3 and §4]. General discussion on converting from two-sided ID to CUR can be found in [Mar18, §11.2], [MT20, §13.2], and [DM21b, §4.1].

4.3 Computational routines

The last section explained how randomized algorithms for low-rank approximation exhibit a great deal of modularity. Here we summarize the design spaces for the constituent modules.

Sections 4.3.1 to 4.3.4 cumulatively cover QB, column ID, CSS, and building blocks for the same. We acknowledge up-front that we treat column ID and CSS in far detail than we do QB. This imbalance is not a statement about the importance of column ID or CSS over QB. Rather, it stems from our desire to clarify broader concepts surrounding column ID that can be difficult to tease out from other literature.

From there, Section 4.3.5 lists methods for norm estimation which are important for solving low-rank approximation problems to fixed accuracy. Our last topic in the realm of computational routines for low-rank approximation is the notion of low-rank approximations from *oblique projections* (§4.3.6). This framework motivates a type of low-rank approximation that is cheap to compute but that does not have meaningfully-structured factors.

We emphasize that this section mentions *many* algorithms for a wide variety of problems. Due to practical constraints we only address a handful of these algorithms in detail. Pseudocode for select algorithms can be found in Appendix C; these algorithms have been selected based on some combination of their conceptual significance and their practicality.

4.3.1 Power iteration

Given a matrix $\mathbf{A}$, suppose we sketch $\mathbf{Y} = \mathbf{AS}$ using a very tall sketching operator $\mathbf{S}$. In a low-rank approximation context – regardless of whether we work with spectral or submatrix-oriented decompositions – it is generally preferable for range($\mathbf{Y}$) to be well-aligned with the span of $\mathbf{A}$'s dominant left singular vectors. This, in turn, is facilitated by having range($\mathbf{S}$) be well-aligned with the span of $\mathbf{A}$'s dominant *right* singular vectors. To accomplish this, RandNLA libraries should include methods for generating such sketching operators based on power iteration.

A basic approach to power iteration makes alternating applications of $\mathbf{A}$ and $\mathbf{A}^*$ to an initial data-oblivious sketching operator $\mathbf{S}_o$, to obtain a data-aware sketching operators such as

$$\mathbf{S} = (\mathbf{A}^*\mathbf{A})^q \mathbf{S}_o \quad \text{or} \quad \mathbf{S} = (\mathbf{A}^*\mathbf{A})^q \mathbf{A}^*\mathbf{S}_o, \tag{4.21}$$

for some parameter $q \geq 0$. Practical implementations need to incorporate some form of stabilization in between the successive applications of $\mathbf{A}$ and $\mathbf{A}^*$). We give a general formulation of such a method in Appendix C.2.1 with Algorithm 8. Notably, this general method allows an arbitrary number of passes over the data matrix (including zero passes, as an API convenience).

Remark 4.3.1. The closest relative to Algorithm 8 in the literature is probably [ZM20, Algorithm 3.3]. However, the core idea behind this algorithm was explored earlier by Bjarkason [Bja19].

4.3.2 Orthogonal projections: QB and rangefinders

We begin with two definitions.

> Given a matrix $\mathbf{A}$, a *QB decomposition* is given by a pair of matrices $(\mathbf{Q}, \mathbf{B})$ where $\mathbf{Q}$ is column-orthonormal and $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$. It is intended that $\mathbf{QB}$ serve as an approximation of $\mathbf{A}$. An algorithm that computes only the factor $\mathbf{Q}$ from a QB decomposition is called a *rangefinder*.

The value of QB decompositions stems from how they define approximations by orthogonal projection: $\hat{\mathbf{A}} = \mathbf{QB} = \mathbf{QQ}^*\mathbf{A}$. It is important to note that QB algorithms *do not necessarily* first compute $\mathbf{Q}$ in one phase and then set $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$ in a second phase. Indeed, the benefit of the rangefinder abstraction is that it considers an equivalent problem while setting aside the potentially-nuanced matter of computing $\mathbf{B}$.

Before proceeding to algorithms, we note that these concepts are not useful in the "full-rank" setting. Consider, for example, when $\mathbf{A}$ has full row-rank. Here, *any* orthogonal matrix $\mathbf{Q}$ and accompanying $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$ are valid outputs of rangefinders and QB decomposition algorithms. Therefore full-rank QB decompositions can be *entirely unstructured*. Despite this caveat, QB decompositions are of fundamental importance in randomized algorithms for low-rank approximation.

Rangefinder algorithms and basic QB

The simplest rangefinders are based on power iteration. For example, one can prepare a data-aware sketching operator $\mathbf{S}$ of the form (4.21), compute $\mathbf{Y} = \mathbf{AS}$, and then return $\mathbf{Q} = \texttt{orth}(\mathbf{Y})$; this is formalized as Algorithm 9 in Appendix C.2.2. More advanced rangefinders use block Krylov subspace methods. Specific examples of such rangefinders can be found in [MM15], [Bja19, §7], and [MT20, §11.7].

Algorithm 10 in Appendix C.2.2 is the simplest approach to QB. It obtains $\mathbf{Q}$ by calling an abstract rangefinder and obtains $\mathbf{B}$ by explicitly computing $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$.

Iterative QB algorithms

The most effective QB algorithms work by building $(\mathbf{Q}, \mathbf{B})$ *iteratively* [MV16]. Generically, each iteration of such a QB method entails some number of matrix-matrix multiplications with $\mathbf{A}$ (as a rangefinder step), adds a specified number of columns to $\mathbf{Q}$ and rows to $\mathbf{B}$, and makes a suitable in-place update to $\mathbf{A}$. Iterations terminate once some metric of approximation error $\mathbf{A} \approx \mathbf{QB}$ (e.g., $\|\mathbf{A} - \mathbf{QB}\|_2$ or an approximation thereof) falls below a certain level.

Iterative QB methods were improved by [YGL18]. In particular, Algorithm 2 of [YGL18] does not modify $\mathbf{A}$, it uses a power-iteration-based rangefinder to compute new blocks for $(\mathbf{Q}, \mathbf{B})$, and it efficiently updates the Frobenius error $\|\mathbf{A} - \mathbf{QB}\|_{\mathrm{F}}$ as the iterations proceed. This method is useful because it has complete control over the Frobenius norm error of the returned approximation. Algorithm 11 in Appendix C.2.2 generalizes this method by allowing an abstract rangefinder in the iterative loop that updates $(\mathbf{Q}, \mathbf{B})$.

Meanwhile, Algorithm 4 of [YGL18] performs power iteration *before* entering its main iterative loop, it does not access $\mathbf{A}$ while in the iterative loop, and it can terminate early if a target accuracy is met before a pre-specified rank-limit. This algorithm has the advantage of reducing the number of times $\mathbf{A}$ is moved through fast memory. In fact, when power iteration is omitted, it can be implemented as a single-pass method based on Type 1 multi-sketching. The downside is that this algorithm may waste a substantial amount of work if the rank limit is much higher than necessary. This downside is compounded when power iteration is omitted. We reproduce this method with slight modifications in Appendix C.2.2 as Algorithm 12.

Stopping criteria for iterative QB algorithms

The Frobenius norm is easily computed for sparse matrices and dense matrices that are stored explicitly in memory. However, it can be difficult to compute for abstract linear operators when the matrix is accessed only via matrix-vector multiplies, and this can pose problems in computing QB decompositions to specified accuracy. One approach to address this situation is by careful application of a well-known randomized Frobenius norm estimator as part of the QB decomposition [GCG+19, §3.4, §3.5, Eq. (3.26)]. We also note that those looking for high-quality approximations often prefer that error be bounded in *spectral norm* (and only use the Frobenius norm because it is usually very cheap to compute). The problem of estimating spectral norms is well-studied in the NLA literature. Section 4.3.5 reviews randomized algorithms for estimating matrix norms.

Approximate single-pass QB via Type 2 multi-sketching

Recall that a Type 2 multi-sketch of $\mathbf{A}$ is a sketch of the form $(\mathbf{Y}_1, \mathbf{Y}_2) = (\mathbf{A}\mathbf{S}_1, \mathbf{S}_2\mathbf{A})$ for independent sketching operators $(\mathbf{S}_1, \mathbf{S}_2)$. This sketch can be used to compute a QB decomposition that is *approximate*, in the sense that $\mathbf{Q}\mathbf{B} \approx \mathbf{A}$ holds for column-orthonormal $\mathbf{Q}$, but we drop the hard requirement that $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$.

Put simply, the method is to compute $\mathbf{Q} = \texttt{orth}(\mathbf{Y}_1)$ and then $\mathbf{B} = (\mathbf{S}_2\mathbf{Q})^\dagger\mathbf{Y}_2$. The intuition behind this approach is that if $\mathbf{Q}\mathbf{Q}^*\mathbf{A}$ is a good approximation for $\mathbf{A}$, then we would have $\mathbf{Y}_2 \approx \mathbf{S}_2\mathbf{Q}\mathbf{Q}^*\mathbf{A}$, which would imply $\mathbf{B} \approx \mathbf{Q}^*\mathbf{A}$. We refer the reader to [TYU+17b, §4.2] for a proper explanation of this method.

4.3.3 Column-pivoted matrix decompositions

Throughout this section we work with an $r \times c$ matrix $\mathbf{G}$. As before, we begin with some definitions.

> A *column-pivoted decomposition* of $\mathbf{G}$ is any decomposition of the form
>
> $$\mathbf{G}\mathbf{P} = \mathbf{F}\mathbf{T} \tag{4.22}$$
>
> where $\mathbf{P}$ is a permutation matrix and $\mathbf{T}$ is upper-triangular. The permutation matrix is encoded by a vector of *pivots*, J, so that $\mathbf{G}[:, J] = \mathbf{G}\mathbf{P}$.

There are many ways of producing such decompositions. We are only interested in the ways where rank-k matrices obtained by truncation

$$\hat{\mathbf{G}} := (\mathbf{F}[:, :k])(\mathbf{T}[:k, :])\mathbf{P}^* \tag{4.23}$$

provide reasonably good rank-k approximations of $\mathbf{G}$. The meaning of "reasonably good" is subjective. It depends on the computational cost of the algorithm, and it depends on how well $\hat{\mathbf{G}}$ approximates $\mathbf{G}$ compared to the best approximation obtained by a truncated column-pivoted matrix decomposition. Interestingly, some randomized algorithms for CSS (see §4.3.4) do not use the factors that appear in (4.23); instead, these algorithms only care that the pivots J *could make* approximations form (4.23) reasonably accurate.

How much do we truncate? When a randomized algorithm uses this primitive for low-rank approximation, the matrix $\mathbf{G}$ is usually a sketch of the target data matrix $\mathbf{A}$, and k is close to $\min\{r, c\}$. It helps to consider different situations when trying to build intuition for why these randomized algorithms work. Specifically, it helps to consider when $\mathbf{G}$ is equal to $\mathbf{A}$, or a low-rank approximation thereof. In both such cases one should have $k \ll \min\{r, c\}$.

Basics of column-pivoted decomposition algorithms

There are two main families of algorithms for producing these decompositions: those based on QR with column pivoting (QRCP) and those based on LU with partial pivoting (LUPP).[2] Algorithms in the former family define the decomposition (4.22) in the natural way, where $\mathbf{F}$ is column-orthonormal. For algorithms in the latter family, one must consider how pivoted LU traditionally uses *row pivoting*. Therefore,

[2] The standard process of computing an LU decomposition with partial pivoting is called *Gaussian elimination with partial pivoting* and is abbreviated as GEPP.

to compute (4.22) via LUPP, one must compute the transposed factors $(\mathbf{P}^*, \mathbf{F}^*, \mathbf{T}^*)$ in a row-pivoted decomposition,

$$\mathbf{P}^*\mathbf{G}^* = \mathbf{T}^*\mathbf{F}^*,$$

where $\mathbf{T}^*$ is lower-triangular (with unit diagonal) and $\mathbf{F}^*$ is upper-triangular.

Roughly speaking, QRCP-based algorithms prioritize accuracy, while LUPP-based algorithms prioritize speed. The extent to which a specific algorithm does well on these performance metrics depends on the algorithm's pivoting rule. The most widely used QRCP-based methods use the same pivoting rule as LAPACK's `GEQP3`. Meanwhile, the most widely used LUPP-based methods use the pivoting rule from LAPACK's `GETRF`. It is valid to rely on either of these functions for the column-pivoted decomposition steps that arise in randomized algorithms. However, one should be aware of two potential sources of error when using `GETRF` for a column-pivoted decomposition rather than `GEQP3`.

What can we expect of `GETRF`'s pivots? When `GETRF` is applied to $\mathbf{G}^*$, the process of computing the pivots up to and including the ℓ^{th} pivot makes decisions based only on the first ℓ rows of $\mathbf{G}$. Therefore it is unwise to use `GETRF` unless one has reason to believe that the information in $\mathbf{G}$'s trailing $r-k$ rows would not drastically alter the columns chosen as pivots based on the first k rows. Similarly, it is unwise to use $\hat{\mathbf{G}}$ as an approximation of $\mathbf{G}$ when $k \ll \min\{r, c\}$, since this would suppose that $\mathbf{G}$'s leading k rows are significantly more important than all others. One is most likely to find meaningful information in the column-pivoted LU decomposition when $\mathbf{G}$ is very wide ($r \ll c$) and k is close to r.

What isn't in the pivots? Suppose $\mathbf{F}$ has $w = \min\{r, c\}$ columns. It is easy to verify that for any nonsingular upper-triangular matrix $\mathbf{U}$ of order w, the decomposition produced after a change-of-basis

$$(\mathbf{F}, \mathbf{T}) \leftarrow (\mathbf{F}\mathbf{U}^{-1}, \mathbf{U}\mathbf{T})$$

will preserve (4.22) for the same permutation matrix $\mathbf{P}$.

It is informative to consider how such changes of basis affect $\hat{\mathbf{G}}$. For example, in simplest case, it is easy to see that $\hat{\mathbf{G}}$ would not change if $\mathbf{U}$ were diagonal. This simple case shows that conditioning of the factors $(\mathbf{F}, \mathbf{T})$ is unimportant in our formalism of column-pivoted decomposition.

To speak to a more interesting case, let us partition $\mathbf{F}$ and $\mathbf{T}$ into blocks $[\mathbf{F}_1, \mathbf{F}_2]$ and $[\mathbf{T}_1; \mathbf{T}_2]$ so that $\mathbf{F}_1$ has k columns and $\mathbf{T}_1$ has k rows. Straightforward calculations show that

$$\|\mathbf{G} - \mathbf{F}_1\mathbf{T}_1\mathbf{P}^*\| = \|\mathbf{F}_2\mathbf{T}_2\| \tag{4.24}$$

holds in any unitarily-invariant norm. Meanwhile, less straightforward calculations[3] show that there is always an upper-triangular nonsingular matrix $\mathbf{U}$ for which

$$\|\mathbf{G} - (\mathbf{F}\mathbf{U}^{-1})[:, :k](\mathbf{U}\mathbf{T}[:k, :])\| = \|(\mathbf{I} - \mathbf{F}_1\mathbf{F}_1^{\dagger})\mathbf{F}_2\mathbf{T}_2\| \leq \|\mathbf{F}_2\mathbf{T}_2\|. \tag{4.25}$$

Note that if there is substantial overlap between range$(\mathbf{F}_1)$ and range$(\mathbf{F}_2)$, then the inequality in (4.25) will be strict by a significant margin. Therefore *if* accuracy of the approximation (4.23) is important, then our decomposition should make sure that the columns of $\mathbf{F}$ are orthogonal to one another. This is ensured by QR-based methods, but it is not ensured by LU-based methods.

[3] See Proposition C.1.3 in Appendix C.1.2.

Partial decompositions

Standard algorithms for computing a column-pivoted decomposition of an $r \times c$ matrix require $\Theta(\min\{rc^2, cr^2\})$ operations. One can get away with spending less effort when only a *partial* decomposition is needed. Formally, in a (k-step) partial column-pivoted decomposition, we relax the requirement that $\mathbf{T}$ be triangular. Instead, we require that $\mathbf{T}$ can be partitioned into a 2-by-2 block triangular matrix where the upper-left block is $k \times k$ and triangular in the proper sense.

The aforementioned standard algorithms for column-pivoted decomposition can be modified to compute k-step partial decompositions of $r \times c$ matrices in $\Theta(rck)$ operations. There are plans for a version of LAPACK subsequent to 3.10 to support this functionality as it pertains to QRCP.[4] At a practical level, it is certainly worth using this functionality when it is available. However, this functionality is not critical, since randomized algorithms for low-rank approximation rarely need to compute k-step partial decompositions with $k \ll \min\{r, c\}$.

More details on column-pivoted decomposition algorithms

The pivots chosen by the strong rank-revealing QR (strong RRQR) algorithm from [GE96] lead to the best theoretical guarantees for low-rank approximation by a partial column-pivoted QR decomposition. In practice, it is more common to truncate the output of LAPACK's `GEQP3`, which is faster than strong RRQR.

Algorithms based on LUPP are typically faster than those based on pivoted QR. While the LUPP approach comes with weaker guarantees (as explained above), these limitations are less significant in a randomized context where we seek nearly full-rank decompositions of *wide sketches*. Indeed, there is little practical difference in solution quality between LUPP-based and QRCP-based versions of some randomized algorithms for CSS and column ID [Mar22b; DM21b].

Other possibilities for column-pivoted matrix decomposition include LU or QR with tournament pivoting [GDX11; DGG+15]. Algorithms based on tournament pivoting exhibit reduced communication and hence can be more efficient without significant loss of accuracy.

Finally, we note that Section 5.1.2 includes a randomized algorithm for full-rank QRCP. It is easy enough to modify that algorithm to support early termination. Some variants of this algorithm specifically focus on low-rank approximation (e.g., the SRQR algorithm from [XGL17]).

4.3.4 One-sided ID and CSS

Column ID and CSS are nearly equivalent problems. That is, a method for CSS can canonically be extended to a method for column ID by taking $\mathbf{X} = (\mathbf{A}[:, J])^{\dagger}\mathbf{A}$. Conversely, a method for column ID can be adapted to CSS by discarding any calculations that are only needed to form $\mathbf{X}$. This section covers deterministic and randomized algorithms for both of these problems. Readers who are particularly interested in theoretical aspects of these algorithms should consult [BMD09].[5]

[4] See https://github.com/Reference-LAPACK/lapack/issues/661.

[5] We frame all of our discussion of one-sided ID around column ID, rather than row ID.

Template deterministic algorithms

Suppose we want to compute a rank-k column ID of an $r \times c$ matrix $\mathbf{G}$. There is a template deterministic algorithm for handling this problem based on the notion of column-pivoted decompositions, as discussed in Section 4.3.3.

The template algorithm works in two phases. The first phase produces the decomposition $\mathbf{GP} = \mathbf{FT}$ where $\mathbf{P}$ is a permutation matrix and $\mathbf{T}$ is upper-triangular. The second phase is a matter of simple postprocessing. The postprocessing begins by partitioning $\mathbf{T}$ into a 2-by-2 block triangular matrix

$$\mathbf{T} = \begin{bmatrix} \mathbf{T}_{11} & \mathbf{T}_{12} \\ \mathbf{0} & \mathbf{T}_{22} \end{bmatrix}.$$

where $\mathbf{T}_{11}$ is $k \times k$ and triangular in the usual sense. From here, one sets the interpolation matrix to $\mathbf{X} = [\mathbf{I}_{k\times k}, \mathbf{T}_{11}^{-1}\mathbf{T}_{12}]\mathbf{P}^*$, and one sets the skeleton indices to the vector J that provides $\mathbf{X}[:, J] = \mathbf{I}_k$.

The importance of this algorithm stems from how its output can equivalently be analyzed as a truncated column-pivoted matrix decomposition. That is, the column ID induced by $(J, \mathbf{X})$ satisfies

$$\mathbf{G}[:, J]\mathbf{X} = \mathbf{F}[:, :k]\mathbf{T}[:k, :]\mathbf{P}^*.$$

We can therefore gain insights into the behavior of this column ID algorithm by appealing to results such as Proposition C.1.3, which we alluded to earlier in our discussion of LU and QR based pivoting methods.

This approach to column ID is illustrated more formally as Algorithm 13 in Appendix C.2.3. It is easy to see that the CSS version of this algorithm does not need to compute $\mathbf{X}$, since the definition of $\mathbf{X}$ implies that J can be determined from the first k rows of $\mathbf{P}$.

Randomized algorithms

We list five randomized algorithms for CSS and column ID below. With some exception for the fifth algorithm, we do not comment on the theoretical guarantees of these methods.

1. For CSS, one can sample columns with probability proportional to their norms, where column norms are updated by projecting out selected columns as a QRCP-like factorization proceeds [DV06]. Applying the standard post-processing scheme to the (partial) factorization yields the interpolation matrix $\mathbf{X}$ needed for a column ID.

2. Also for CSS, one can sample columns according to a probability distribution related to so-called *leverage scores* of the matrix under consideration. We discuss leverage score sampling in detail in Section 6. For now, we note that this approach especially useful for computing Nyström approximations.

3. The algorithm in [BMD09] approaches CSS with a combination of leverage score sampling and postprocessing by deterministic QRCP. The factorization produced by this postprocessing can be processed further to produce the interpolation matrix for a column ID.

4. [XGL17, §V.D] suggests solving CSS by taking the pivots from a randomized algorithm for QRCP. The output of the randomized algorithm for QRCP can, of course, be processed to recover the interpolation matrix for a column ID.

5. [VM16, §5.1] approaches low-rank column ID by computing a (nearly) full-rank column ID of a sketch $\mathbf{Y} = \mathbf{SA}$. The unmodified data $(\mathbf{X}, J)$ is used to define the low-rank column ID $\mathbf{A}[:, J]\mathbf{X} \approx \mathbf{A}$.

The last of these methods is simple and practical. It appears with slight modifications in Appendix C.2.3 as Algorithm 14, while the corresponding CSS version appears as Algorithm 15. For both the column ID and CSS versions, it is recommended that $\mathbf{S}$ be a data-aware sketching operator based on power iteration. To gain intuition for this method, one should first verify that if $(\mathbf{X}, J)$ defines a full-rank column ID of $\mathbf{Y}$, then it also defines a full-rank column ID of $\tilde{\mathbf{A}} = (\mathbf{A}\mathbf{Y}^{\dagger})\mathbf{Y}$. With that given, we can apply Proposition 4.1.3 to see that the induced low-rank column ID satisfies an error bound

$$\|\mathbf{A} - \mathbf{A}[:, J]\mathbf{X}\|_2 \leq (1 + \|\mathbf{X}\|_2)\|\mathbf{A} - \tilde{\mathbf{A}}\|_2.$$

This bound is noteworthy for the following reason: using power iteration to prepare a data-aware sketching operator $\mathbf{S}$ will drive $\tilde{\mathbf{A}}$ closer to a rank-k approximation of $\mathbf{A}$ obtained by a truncated SVD. That, in turn, would give $\|\mathbf{A} - \tilde{\mathbf{A}}\|_2 \approx \sigma_{k+1}(\mathbf{A})$.

Remark 4.3.2. The value of power iteration in the context of CSS / column ID *and* in the context of rangefinders / QB is a key reason for considering power iteration as a basic primitive of RandNLA.

On fixed-accuracy one-sided ID

Standard implementations of deterministic QRCP-based algorithms for one-sided ID can compute approximations to specified accuracy. *Randomized* algorithms for low-rank one-sided ID do not possess this capability to the same extent. In some respects, this is a principal disadvantage of low-rank approximation by ID compared to QB. However, there are partial workarounds.

For example, suppose we approximate $\mathbf{A}$ via a QB decomposition, $\tilde{\mathbf{A}} = \mathbf{QB}$. If we computed $(\mathbf{X}, J)$ by a full-rank column ID of $\mathbf{B}$, then we would also have a full-rank column ID of $\tilde{\mathbf{A}}$. If $\mathbf{X}$ was obtained by the standard postprocessing of output from strong rank-revealing QR, then we would have $|X_{ij}| \leq 2$. A straightforward application of Proposition 4.1.3 shows that if we had

$$\|\mathbf{A} - \mathbf{QB}\|_2 \leq \frac{\epsilon}{(1 + \sqrt{1 + 4k(n-k)})} \tag{4.26}$$

for a rank-k QB decomposition, then we could be certain that $\|\mathbf{A} - \mathbf{A}[:, J]\mathbf{X}\|_2$ was at most ϵ. Therefore, in principle, one could compute the QB decomposition iteratively, and only compute the ID of $\mathbf{B}$ once (4.26) is satisfied.

The above approach is not without its shortcomings. For one thing, reducing $\|\mathbf{A} - \mathbf{QB}\|_2$ entails increasing k, and so the termination criterion is a moving target. As another issue, it needs to bound spectral norms of implicitly represented linear operators. We address the problem of estimating matrix norms next.

4.3.5 Estimating matrix norms

Norm estimation plays an important role in stopping criteria for iterative low-rank approximation algorithms, particularly for QB and Nyström approximations. Here we summarize methods that would be appropriate for expensive norms or norms of abstract linear operators that are only accessible by matrix-vector multiplications.

Remark 4.3.3. The material presented here is covered in greater detail in [HMT11, §4.3 - §4.4] and [MT20, §5 - §6, §12.0 - §12.4].

A cheap spectral norm bound. Let the vectors $\boldsymbol{z}_1, \dots, \boldsymbol{z}_r \in \mathbb{R}^n$ be vectors with components drawn iid from the standard normal distribution and let $\beta > 1$ be a tuning parameter. Then, for any $\mathbf{A}$, it is known that the inequality

$$\|\mathbf{A}\|_2 \;\leq\; \beta\sqrt{\frac{2}{\pi}} \max_{j \in [\![r]\!]} \|\mathbf{A}\boldsymbol{z}_j\|_2 \tag{4.27}$$

holds with probability at least $1 - \beta^{-r}$ [HMT11; WLR+08]. Furthermore, this bound is easy to compute because the necessary vectors $\mathbf{A}\boldsymbol{z}_j$ can be formed with a single matrix-matrix product with $\mathbf{A}$.

A basic Frobenius norm estimator. Let $\mathbf{Z} \in \mathbb{R}^{n \times r}$ be the matrix whose columns are the random vectors $\boldsymbol{z}_1, \dots, \boldsymbol{z}_r$ mentioned above. Then, it turns out that the quantity $\frac{1}{r}\|\mathbf{AZ}\|_{\mathrm{F}}^2$ is an unbiased estimate for the squared Frobenius norm, in the sense that

$$\mathbb{E}\left[\tfrac{1}{r}\|\mathbf{AZ}\|_{\mathrm{F}}^2\right] \;=\; \|\mathbf{A}\|_{\mathrm{F}}^2. \tag{4.28}$$

In addition to being unbiased, the variance of the error estimate can also be controlled according to

$$\mathrm{var}\left(\tfrac{1}{r}\|\mathbf{AZ}\|_{\mathrm{F}}^2\right) \;\leq\; \tfrac{2}{r}\|\mathbf{A}\|_2^2\|\mathbf{A}\|_{\mathrm{F}}^2, \tag{4.29}$$

as shown in [Gir89]. Hence, as long as r is sufficiently large, then the error estimate $\frac{1}{r}\|\mathbf{AZ}\|_{\mathrm{F}}^2$ is likely to be close to $\|\mathbf{A}\|_{\mathrm{F}}^2$. From a computational standpoint, this error estimate is similar to the one described above for the spectral norm, insofar as it only requires r matrix-vector products with $\mathbf{A}$.

A cheap Schatten p-norm estimator. Letting $\boldsymbol{\sigma}$ denote the vector of singular values of $\mathbf{A}$, the *Schatten 2p-norm* of $\mathbf{A}$ is $\|\mathbf{A}\|_{(S,2p)} := \left(\sum_{i=1}^{\min\{m,n\}} \sigma_i^{2p}\right)^{1/2p}$. Taking $p = 1$ reduces to the Frobenius norm. The spectral norm is obtained in the limit as $p \to \infty$. In fact, deterministic bounds show that the spectral norm and Schatten p-norm more or less coincide when $p \gtrsim \log \min\{m, n\}$.

The Kong-Valiant estimator [KV17b] can be used to cheaply estimate these norms. It only accesses $\mathbf{A}$ by multiplication with an $n \times k$ data-oblivious sketching operator, where k can be materially smaller than $\min\{m, n\}$. See [MT20, §5.4] for a statement of the algorithm and remarks on its theoretical guarantees.

Accurate spectral norm estimators. There is a large literature on deterministic and randomized algorithms for estimating spectral norms. Much of this literature is based on methods designed for estimating the largest eigenvalue of a positive definite matrix (which can naively be applied since $\sqrt{\|\mathbf{A}^*\mathbf{A}\|_2} = \|\mathbf{A}\|_2$). Most notably, Dixon

was the first to study the randomized power method [Dix83], and Kuczyński and Woźniakowski were the first to study randomized Lanczos methods [KW92]. See [MT20, Algorithm 5] for a basic randomized Lanczos method and the subsequent remarks on block randomized Lanczos [MT20, §6.5].

4.3.6 Oblique projections

Low-rank approximations can be expressed in a manner resembling the triple-sketch from Section 2.6. For sketching operators $\mathbf{S}_1 \in \mathbb{R}^{n\times k}$ and $\mathbf{S}_2 \in \mathbb{R}^{d\times m}$, we can define

$$\hat{\mathbf{A}} = \mathbf{A}\mathbf{S}_1(\mathbf{S}_2\mathbf{A}\mathbf{S}_1)^\dagger\mathbf{S}_2\mathbf{A} = \mathbf{Y}_1\mathbf{Y}_3^\dagger\mathbf{Y}_2,$$

where

$$\mathbf{Y}_1 = \mathbf{A}\mathbf{S}_1, \quad \mathbf{Y}_2 = \mathbf{S}_2\mathbf{A}, \quad \text{and} \quad \mathbf{Y}_3 = \mathbf{S}_2\mathbf{A}\mathbf{S}_1.$$

This construction obtains each column of $\hat{\mathbf{A}}$ by projecting the corresponding column of $\mathbf{A}$ onto the range of $\mathbf{Y}_1$, where the projection is orthogonal with respect to the possibly degenerate inner product $(\boldsymbol{u}, \boldsymbol{v}) \mapsto \langle \mathbf{S}_2\boldsymbol{u}, \mathbf{S}_2\boldsymbol{v}\rangle$. We call $\hat{\mathbf{A}}$ an *oblique projection of* $\mathbf{A}$.

The simplest oblique projections use column and row selection operators for $(\mathbf{S}_1, \mathbf{S}_2)$. This provides a CUR decomposition where $\mathbf{Y}_3^\dagger$ is the linking matrix $\mathbf{U}$. The connection to CUR foreshadows a more general fact: the sketching operators used in oblique projection are not necessarily independent of one another [DMM08]. An example in this regard is that Nyström approximations amount to oblique projections that use $\mathbf{S}_2 = \mathbf{S}_1^*$.

It is natural to consider oblique projections where $\mathbf{S}_1$ and $\mathbf{S}_2$ are independent (e.g., independent Gaussian operators). Such approximations can entail extremely ill-conditioned computations if one is not careful. This ill-conditioning can be avoided through the numerically stable approach described by Nakatsukasa [Nak20]. These approximations employ oversampling for $\mathbf{S}_2$ (relative to $\mathbf{S}_1$) and split $\mathbf{Y}_3$ (or a regularized variant thereof) into two factors. The representation returned by this approach consists of four matrices.

Historical notes

Oblique projections for low-rank approximation are closely related to the rank reduction formula described in [CFG95]. Drineas et al. first used oblique projections for low-rank approximation via CUR decomposition [DMM08], wherein $\mathbf{S}_1, \mathbf{S}_2$ are column and row selection matrices respectively. Clarkson and Woodruff pioneered the use of general oblique projections in randomized algorithms for low-rank approximation [CW09, Theorem 4.7]. Oblique projections have since been discussed in the context of a generalized LU factorization [DGR19].

4.4 Other low-rank approximations

Here we review a handful of other low-rank approximation problems and algorithms, particularly speaking to our development plans for RandLAPACK.

Domain-specific representations. Several low-rank approximation problems of interest involve specialized factorizations. We plan for RandLAPACK to eventually support nonnegative matrix factorization [EMW+18], dynamic mode decomposition (DMD) [EMK+19; EBK19], and possibly sparse PCA [EZM+20]. Among these methods, we expect that DMD will have highest priority, since full-rank DMD is slated for inclusion into LAPACK in the near future [Drm22]. For a general introduction to DMD we refer the reader to [TRL+14].

Low-rank Cholesky. As a separate topic, there is also a longstanding algorithm for "low-rank Cholesky" decompositions [XG16]. We are unsure of its eventual role in RandLAPACK, since a representation of the form $\hat{\mathbf{A}} = \mathbf{L}\mathbf{L}^*$ for a very tall lower-triangular matrix $\mathbf{L}$ offers almost no benefit over $\mathbf{L}$ being dense. Still, it will be considered in the near future alongside the recently proposed algorithm by [CET+22] for randomly pivoted partial Cholesky decomposition.

Low-rank QR. Suppose $\mathbf{A}$ is a large full column-rank matrix with QR decomposition $\mathbf{A} = \mathbf{Q}\mathbf{R}$. This decomposition has two especially prominent uses: (1) it facilitates application of a pseudoinverse $\mathbf{A}^\dagger \boldsymbol{v} = \mathbf{R}^{-1}\mathbf{Q}^*\boldsymbol{v}$ in $O(mn)$ time, and (2) it can be used as preprocessing for more complicated orthogonal decompositions such as SVD. Unfortunately, low-rank QR decomposition, which is simply the economic QR decomposition of a rank-k approximation of $\mathbf{A}$, does *not* fully realize either of these use-cases.

The trouble with low-rank QR is that a $k \times n$ upper-triangular matrix with $k \ll n$ is effectively a full matrix. That is, the mere *representation* of a low-rank matrix by a QR decomposition is not much more useful than representation by QB decomposition. Note also that unpivoted QR makes no effort to produce a rank-revealing representation, compared to pivoted QR. Therefore RandLAPACK will not offer methods for low-rank approximation by unpivoted QR.

Low-rank UTV. A UTV decomposition $\hat{\mathbf{A}} = \mathbf{U}\mathbf{T}\mathbf{V}^*$ uses column-orthogonal matrices $\mathbf{U}, \mathbf{V}$ and a triangular matrix $\mathbf{T}$. UTV (also called QLP) can be thought of as a cheaper alternative to SVD. As we discuss in the next section, RandLAPACK might include algorithms for UTV when $\hat{\mathbf{A}}$ is full-rank [GM18; MQH19; KCL21]. Some of those algorithms (e.g., that in [MQH19]) proceed iteratively and can be terminated early. If RandLAPACK supports full-rank UTV by such an algorithm then it will expose the low-rank variant.

Several algorithms for producing low-rank approximations represented by UTV are given in [DG17; FXG19; WX20; RB20; KC21]. We would need a better understanding of those methods, particularly how they compare to our planned methods for low-rank SVD, before making decisions on which of them to support.

Low-rank LU. LU is central to solving systems of linear equations in the full-rank case. There is a small literature on low-rank LU within the field of RandNLA: [SSA+18; DGR19; ZM20]. In RandLAPACK we anticipate restricting our attention to algorithms that are related to a Gaussian elimination process (that is, where the error matrix can be expressed as a Schur complement of a block matrix), along the lines of [DGR19]. These algorithms are likely more useful for low-rank approximation with a fixed accuracy requirement rather than with a fixed rank requirement. They are based on an oblique projection with $k = d$, that is $\mathbf{S}_2\mathbf{A}\mathbf{S}_1$ is square.

RandLAPACK might include the LU algorithms from [SSA+18] if they can be proven to be significantly faster than high-quality implementations of QB algorithms. If proven useful, we will consider in the future generalized LU-based low-rank approximation, as introduced in [DGR19]. The algorithms for low-rank LU in [ZM20] are based on QB and so are unlikely to be included in RandLAPACK.

4.5 Existing libraries

Here we review established numerical libraries that support randomized low-rank approximation. All of the libraries that focus on RandNLA (save for one) implement advanced sketching operators such as SRFTs.

NLA and data science packages that use RandNLA. There are a few packages for NLA that include a method for low-rank SVD based on QB decomposition:

- `MLSVD_RSI` in the Tensorlab MATLAB toolbox
- `rsvd` in SciKit-CUDA,
- `cusolverDnXgesvdr` in NVIDIA's cuSOLVE,
- and `randomized_svd` in SciKit-Learn.

The last of these functions warrants special emphasis. SciKit-Learn's `pca` function actually *defaults* to `randomized_svd` for sufficiently large matrices [Gri16]. In this way, one of the most important functions in the most widely-used Python package for data science already relies on RandNLA.

ID. ID is a Fortran library for ID/CUR [MRS+14]. It is callable as part of the SciPy Python library. ID provides indirect support for SVD as part of its methods for converting one low-rank factorization into another. It also includes routines for rank estimation and norm estimation. RandLAPACK will include many of these same utilities as ID while expanding its scope of driver-level functions.

RSVDPACK. RSVDPACK is a C and MATLAB library for low-rank SVD and ID/CUR [VM15]. It is callable after building from source code which is provided on GitHub. Its SVD algorithms are based on a particular QB implementation [VM15, §3.4] and its ID/CUR algorithms follow [VM16]. RSVDPACK comes in different implementations which target different architectures.

By comparison, RandLAPACK will take more general approaches to QB and ID/CUR, and it will include methods for other factorizations such as eigendecomposition via Nyström approximations. RandLAPACK will target different architectures by building on LAPACK++ as a portability layer [GLA+17].[6]

Ristretto. Ristretto is available on the Python Package Index. This library is based on the rsvd package implemented in R [EVB+19]. It supports low rank SVD, ID/CUR, LU, Nyström, PCA, Hermitian eigendecomposition, nonnegative matrix factorization [EMW+18], dynamic mode decomposition [EMK+19; EBK19; ED16],

[6] This library is developed as part of SLATE [KWG+17; AAB+17].

and sparse PCA [EZM+20]. One algorithm is provided for each distinct type of factorization. Many of these algorithms are based on QB [EVB+19, §3.3], while its ID/CUR algorithms also follow [VM16]. This library has also been demonstrated to be useful for finding patterns in large-scale climate data [VEK+19], and for providing routines for randomized tensor decompositions [EMB+20].

We plan for RandLAPACK to eventually support the same range of factorizations as Ristretto (with the exception of low-rank LU). However, our priority is to focus on the factorizations in Section 4.2, and to offer a range of algorithms for computing each of these decompositions. Our longer-term plans include making RandLAPACK's C++ implementation callable from Python.

LibSkylark. LibSkylark [KAI+15] is written in C++ and callable after installing from source, which is available on GitHub. To our knowledge, it is the only RandNLA library that supports both least squares and low-rank approximation. Its low-rank approximation functionality is restricted to SVD through a QB approach. See Section 3.5 for its least squares functionality.

LowRankApprox.jl. LowRankApprox.jl is a Julia library for low-rank SVD, QR, ID, CUR, and Hermitian eigendecomposition. It is callable after installation with the Julia package manager. Most of its algorithms are based on first computing an ID, rather than a QB decomposition. Note that this is quite different from the plans we have outlined for RandLAPACK over Sections 4.2 and 4.3.

Other implementations. The many algorithms considered in [Bja19] are accompanied by Python implementations hosted on GitHub. The RandNLA tutorial [Wan15] covers a wide range of algorithms for low-rank approximation and hosts some MATLAB implementations on GitHub.

Section 5

Further Possibilities for Drivers

This section covers multi-purpose matrix decompositions, the solution of unstructured linear systems, and trace estimation. These are the last problems we cover that might be handled by "drivers" in a high-level RandNLA library. We emphasize that this monograph does not exhaust the set of prominent linear algebra problems that are amenable to randomization. We make no effort to cover randomized algorithms for general eigenvalue problems, nor do we cover randomized algorithms for computing the action of matrices produced from matrix functions (i.e., computing $f(\mathbf{A})\boldsymbol{b}$ for an analytic matrix function f), even though there are effective algorithms for both of these problems [NT21; GS22; CKN22].

We have chosen the topics of this section because they require comparatively little background material to state, and we believe our summary of the relevant algorithms has some contribution to the literature. For example, the key contribution from Section 5.1 is a novel algorithm for QR with column pivoting based on Cholesky QR. The algorithm is notable for its ability to handle ill-conditioned or even outright rank-deficient matrices. The contributions from Section 5.2 include detailed introductions to recently-developed iterative methods for solving general linear systems. Finally, our coverage of trace estimation in Section 5.3, includes state-of-the-art algorithms and implementations that were not available when earlier RandNLA surveys were published.

5.1 Multi-purpose matrix decompositions

Early in the year 2000, the IEEE publication *Computing in Science & Engineering* published a list of the top ten algorithms of the twentieth century. Among this list was *the decompositional approach to matrix computation*, on which G. W. Stewart gave the following remark.

> The underlying principle of the decompositional approach of matrix computation is that it is not the business of matrix algorithmists to solve particular problems but to construct computational platforms from which a variety of problems can be solved. This approach, which was in full swing by the mid-1960s, has revolutionized matrix computation.

This section covers three decompositions that provide broad platforms for problem solving. They are addressed in an order where randomization offers increasing benefits over purely deterministic algorithms. We note in advance that these randomized algorithms do not aim for an asymptotic speedup over deterministic methods. Rather, the aim is to significantly reduce time-to-solution by taking better advantage of modern computing hardware.

5.1.1 QR decomposition of tall matrices

Algorithms for computing unpivoted QR decompositions are true workhorses of numerical linear algebra. They are the foundation for the preferred algorithms for solving least squares problems with full-rank data matrices. They are also an important ingredient in preprocessing for more expensive algorithms.

For example, suppose we want to decompose a very tall $m \times n$ matrix $\mathbf{A}$ via QR with column pivoting. The instinctive thing to do here is to reach for the LAPACK function `GEQP3`. However, on modern machines, it is much faster to compute an unpivoted decomposition $\mathbf{A} = \mathbf{QR}$, and then run `GEQP3` on $\mathbf{R}$. The final decomposition would be mathematically equivalent to calling `GEQP3` directly on $\mathbf{A}$, just represented in a different format.

With this significance of unpivoted QR in mind, we briefly cover two types of randomized algorithms for computing such decompositions.

Orthogonality in the standard inner product

Cholesky QR is a method for computing unpivoted QR decompositions of matrices with linearly independent columns. It is based on the following elementary observation: given a QR decomposition $\mathbf{A} = \mathbf{QR}$ of a full-column-rank matrix $\mathbf{A}$, the factor $\mathbf{R}$ is simply the upper-triangular Cholesky factor of the Gram matrix $\mathbf{A}^*\mathbf{A}$. Therefore in principle one can compute a QR decomposition as follows.

1. Compute a Cholesky decomposition of the Gram matrix $\mathbf{A}^*\mathbf{A} = \mathbf{R}^*\mathbf{R}$.
2. Perform a matrix-matrix triangular solve to obtain $\mathbf{Q} = \mathbf{A}\mathbf{R}^{-1}$.

Implementing Cholesky QR only requires three functions: `syrk` from BLAS, `potrf` from LAPACK, and `trsm` from BLAS. Standard implementations of these functions parallelize extremely well. As a result, Cholesky QR can offer substantial speedups over Householder QR (and even Tall-and-Skinny QR [DGG+15]) for very tall matrices on modern machines.

Despite the speed advantage of Cholesky QR, it is rarely used in practice, since it is unsuitable for even moderately ill-conditioned matrices. Recently it has been shown that randomization can overcome this limitation by preconditioning Cholesky QR to ensure stability [FGL21]. For detailed analysis of this method we refer the reader to the results in [Bal22b] on the algorithm called "`RCholeskyQR2`."

In Section 5.1.2 we extend this methodology to rank-deficient matrices, and we connect it to an existing randomized algorithm for QRCP of general matrices.

Orthogonality in a sketched inner product

In [BG22], Balabanov and Grigori propose a randomized Gram–Schmidt (RGS) process that orthogonalizes n vectors in $\mathbb{R}^m$ with respect to a sketched inner product

$$\langle \boldsymbol{u}, \boldsymbol{v} \rangle_{\mathbf{S}} = (\mathbf{S}\boldsymbol{u})^*(\mathbf{S}\boldsymbol{v}). \tag{5.1}$$

We call such vectors $\mathbf{S}$-*orthogonal* or *sketch-orthogonal*. When [BG22, Algorithm 2] is run on the columns of a matrix $\mathbf{A}$, values computed during sketched projections are assembled in an upper-triangular matrix $\mathbf{R}$ so that $\mathbf{A} = \mathbf{QR}$ and $(\mathbf{SQ})^*(\mathbf{SQ}) = \mathbf{I}_n$. One can choose the distribution from which $\mathbf{S}$ is drawn so that $\mathbf{Q}$ will be nearly-orthonormal with respect to the standard inner product, with high probability. Empirical and theoretical results show RGS is faster than classic Gram–Schmidt but as stable as modified Gram–Schmidt.

The idea of computing QR decompositions where $\mathbf{Q}$ is sketch-orthogonal can be taken in several directions. For example, a block version of RGS is proposed and analyzed in [BG21]. Taking this approach to the extreme where the block size is the number of columns in the matrix, one can compute the factor $\mathbf{R}$ by Householder QR on $\mathbf{SA}$ and then represent $\mathbf{Q} = \mathbf{AR}^{-1}$ as a linear operator. Note that this procedure is the essence of sketch-and-precondition for least squares, as proposed in [RT08]. A detailed numerical analysis of this last method can be found in [Bal22b], where the algorithm is called `RCholeskyQR`.

5.1.2 QR decomposition with column pivoting

We recall the following reformulation of QR with column pivoting (QRCP) for the reader's convenience.

> Given a matrix $\mathbf{A}$, produce a column-orthogonal matrix $\mathbf{Q}$, an upper-triangular matrix $\mathbf{R}$, and a permutation vector J so that
>
> $$\mathbf{A}[:, J] = \mathbf{QR}.$$

The diagonal entries of $\mathbf{R}$ should approximate $\mathbf{A}$'s singular values, and the columns of $\mathbf{Q}$ should approximate $\mathbf{A}$'s left singular vectors. These stipulations reflect QRCP's main use cases: in low-rank approximation and in solving ill-conditioned least squares problems. As usual, we say that our matrix $\mathbf{A}$ is $m \times n$.

It's all in the pivots. We note that if $m \geq n$, then for any permutation vector J, the economic QR decomposition of $\mathbf{A}[:, J]$ is unique.[1] Therefore J completely

[1] Technically, it is only unique up to sign flips on the columns of $\mathbf{Q}$ and rows of $\mathbf{R}$. But it is clear how signs must be chosen if the diagonal of $\mathbf{R}$ is to approximate the singular values of $\mathbf{A}$.

determines how well the columns of **Q** (resp., diagonal entries of **R**) approximate the left singular vectors of **A** (resp., singular values of **A**).

The method of choosing pivots that sees the widest use today (a simple method based on column norms) was first described in [BG65]. The straightforward implementation of this method can have subtle failure cases in finite-precision arithmetic, however, this can be resolved by carefully restructuring norm calculations [DB08].

An established randomized algorithm for general matrices

Here, we outline a remarkable algorithm first developed by Martinsson [Mar15] and Duersch and Gu [DG17], and then refined by Martinsson, Quintana-Ortí, Heavner, and van de Geijn [MHG17]. This refined algorithm was introduced with the name *Householder QR with Randomization for Pivoting* or *HQRRP*. As this name implies, the factor **Q** from HQRRP is an $m \times m$ operator defined by n Householder reflectors. The algorithm can run much faster than standard QRCP methods by processing the matrix in column blocks, which makes it possible to cast the overwhelming majority of its operations in terms of BLAS 3, instead of about half BLAS 2.

While a full description of HQRRP is beyond our scope, we can outline its structure. As input, it requires that the user provide a block size parameter b and an oversampling parameter s. Typical values for these parameters are $b = 64$ and $s = 10$. HQRRP starts by forming a thin $(b + s) \times n$ sketch $\mathbf{Y} = \mathbf{SA}$, and then it enters the following iterative loop.

1. Use any QRCP method to find P_{block}: the first b pivots for **Y**.
2. Process the panel $\mathbf{A}[:, P_{\text{block}}]$ by QRCP.
3. Suitably update $(\mathbf{A}, \mathbf{Y})$ and return to Step 1.

The update to **A** at Step 3 can be handled by standard methods, such as those used in blocked unpivoted Householder QR. The update to **Y** is more subtle. If done appropriately (particularly, by Duersch and Gu's method [DG17]) then the leading term in the FLOP count for HQRRP is identical to that of unpivoted Householder QR. The one downside of this algorithm is that the diagonal entries of **R** are not guaranteed to decrease across block boundaries.

Implementation notes. We adapted the C implementation from [MHG17] into C++ code at

https://github.com/rileyjmurray/hqrrp.

Our main change was to access BLAS and LAPACK through BLAS++ and LAPACK++. The modified code also allows for matrix dimensions to be specified with either 32-bit or 64-bit integers and includes a small test suite.

We briefly point out two opportunities to improve the performance of this algorithm. The first is to use mixed-precision arithmetic. Specifically, both the sketch of **A** and the call to deterministic QRCP on that sketch could use reduced precision. Given that the real purpose of QRCP on the sketch is to select the block pivot indices for **A**, it might be that loss of accuracy in that phase does not compromise the accuracy of the larger algorithm. The second opportunity is to call unpivoted QR on the very matrix $\mathbf{A}_{\text{panel}}$ in the second phase of processing a block; if pivoting is used in the second phase then the pivots can be determined by deterministic QRCP on the **R** factor from the unpivoted QR of $\mathbf{A}_{\text{panel}}$.

A novel randomized algorithm for very tall matrices

The following algorithm overcomes the limitation of the preconditioned Cholesky QR methodology from [FGL21] of requiring full-rank data matrices. It does so by using a randomized preconditioner based on QRCP.

Algorithm 7 QRCP via sketch-and-precondition and Cholesky QR.

1: **function** $[\mathbf{Q}, \mathbf{R}, J] = \texttt{sap\_chol\_qrcp}(\mathbf{A}, d)$

Inputs:
A matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, an integer d satisfying $n \leq d \ll m$

Output:
Column-orthonormal $\mathbf{Q} \in \mathbb{R}^{m \times k}$, upper-triangular $\mathbf{R} \in \mathbb{R}^{k \times n}$, and a permutation vector J of length n.

Abstract subroutines:
`SketchOpGen` generates an oblivious sketching operator

2: $\mathbf{S} = \texttt{SketchOpGen}(d, m)$ # $\mathbf{S}$ is $d \times m$
3: $[\mathbf{Q}^{\text{sk}}, \mathbf{R}^{\text{sk}}, J] = \texttt{qrcp}(\mathbf{SA})$ # $\mathbf{SA}[:, J] = \mathbf{Q}^{\text{sk}}\mathbf{R}^{\text{sk}}$
4: $k = \text{rank}(\mathbf{R}^{\text{sk}})$
5: $\mathbf{A}^{\text{pre}} = \mathbf{A}[:, J[:k]](\mathbf{R}^{\text{sk}}[:k, :k])^{-1}$
6: $[\mathbf{Q}, \mathbf{R}^{\text{pre}}] = \texttt{chol\_qr}(\mathbf{A}^{\text{pre}})$
7: $\mathbf{R} = \mathbf{R}^{\text{pre}}\mathbf{R}^{\text{sk}}[:k, :]$
8: **return** $\mathbf{Q}$, $\mathbf{R}$, J

Remark 5.1.1. This monograph was released as a technical report in November 2022. It has come to our attention that Algorithm 7 was discovered slightly earlier by Balabanov; it is termed `RRRCholesyQR2` in arXiv:2210.09953:v2 [Bal22b].

The following proposition states that Algorithm 7 produces correct output in exact arithmetic, under mild assumptions on $(\mathbf{S}, \mathbf{A})$. We prove the proposition in Appendix D.

Proposition 5.1.2. *Consider the context of Algorithm 7. If* $\text{rank}(\mathbf{SA}) = \text{rank}(\mathbf{A})$ *then* $\mathbf{A}[:, J] = \mathbf{QR}$.

A practical implementation of Algorithm 7 would need to consider aspects of finite-precision arithmetic. One such aspect is that we cannot use the exact rank for $\mathbf{R}^{\text{sk}}$ on Line 4. Instead, some tolerance-based scheme would be needed.

In analyzing the behavior of this algorithm our main concern is the condition number of $\mathbf{A}^{\text{pre}}$. Indeed, if that matrix is not well-conditioned, then the factor $\mathbf{Q}$ from Cholesky QR may not be orthonormal to machine precision. More generally, if $\text{cond}(\mathbf{A}^{\text{pre}}) \geq \epsilon^{-1/2}$ (where ϵ is the working precision), then it is possible for Cholesky QR to fail outright.

Our next proposition says that if $\mathbf{A}^{\text{pre}}$ is formed in exact arithmetic then its condition number depends on neither the conditioning of $\mathbf{A}$ nor that of $\mathbf{A}^{\text{sk}}$. Therefore if the distribution of the sketching operator is chosen judiciously, then the algorithm should return an accurate decomposition with extremely high probability.

Proposition 5.1.3. *Consider the context of Algorithm 7 and let* $\mathbf{U}$ *be an orthonormal basis for the range of* $\mathbf{A}$*. If* $\operatorname{rank}(\mathbf{SA}) = \operatorname{rank}(\mathbf{A})$*, then the singular values of* $\mathbf{A}^{\mathrm{pre}}$ *are the reciprocals of the singular values of* $\mathbf{SU}$*.*

Proposition 5.1.3 follows easily from Proposition 3.3.1; we omit a formal proof.

Application to matrices with any aspect ratio. Although Cholesky QR only applies to very tall matrices, one could apply it to any $m \times n$ matrix $\mathbf{A}$ (with $m \geq n$) by processing the matrix in blocks.

In fact, it would be natural to use Cholesky QR as the subroutine for processing a block of columns of $\mathbf{A}$ in HQRRP. Since each iteration of HQRRP performs QRCP on a sketch of $\mathbf{A}$, the triangular factor from that run of QRCP can be used as the preconditioner in processing the subsequent panel of $\mathbf{A}$. However, there is a complication in this approach.

> HQRRP's update rule for $\mathbf{A}$ requires that each panel's orthogonal factor is represented as a composition of b Householder reflectors, where each reflector is $m \times m$. By contrast, Cholesky QR only returns an explicit $m \times b$ column-orthonormal matrix $\mathbf{Q}$.

This issue can be resolved by using a method to restore the full Householder representation of the explicit column-orthonormal matrix $\mathbf{Q}$. In LAPACK, this is done with `sorhr_col`, which amounts to unpivoted LU factorization. While pairing Cholesky QR with `sorhr_col` will reduce its speed benefit, it may still be faster than Householder QR (`GEQRF`) and Tall-and-Skinny QR (`GEQR`) in certain settings. Detailed analysis of and benchmarks for this method are forthcoming.

5.1.3 UTV, URV, and QLP decompositions

If QRCP cannot be relied upon to provide an adequate surrogate for the SVD, then one can consider decompositions of the form

$$\mathbf{A} = \mathbf{UTV}^*,$$

where $\mathbf{U}, \mathbf{V}$ are column-orthogonal and $\mathbf{T}$ is square and triangular. This recovers the SVD when $\mathbf{T}$ is the diagonal matrix of singular values of $\mathbf{A}$. It also recovers QRCP when $\mathbf{V}$ is a permutation matrix. These decompositions were first meaningfully studied by Stewart [Ste92; Ste93; Ste99]. They are known by various names, including *UTV*, *URV*, and *QLP*. We have a slight preference for the name "UTV" for aesthetic reasons.

Deterministic algorithms

Stewart's best-known algorithm for UTV (see [Ste99]) is as follows.

1. Run QRCP on the original matrix: $\mathbf{A} = \mathbf{Q}_1\mathbf{R}_1(\mathbf{P}_1)^*$.
2. Run QRCP on $(\mathbf{R}_1)^*$, to obtain $\mathbf{R}_1 = \mathbf{P}_2(\mathbf{R}_2)^*(\mathbf{Q}_2)^*$.
3. Grouping terms, we find the factors

$$\mathbf{A} = \big(\underbrace{\mathbf{Q}_1\mathbf{P}_2}_{\mathbf{U}}\big)\,\underbrace{(\mathbf{R}_2)^*}_{\mathbf{T}}\,\big(\underbrace{\mathbf{P}_1\mathbf{Q}_2}_{\mathbf{V}}\big)^*.$$

   Note in particular that $\mathbf{T}$ is *lower* triangular.

Assuming the standard pivoting scheme is used in the second call to QRCP, one can be certain that the diagonal entries of $\mathbf{T}$ are in decreasing order: $T_{ii} \geq T_{jj}$ for $j \leq i$. Numerical experiments show that the diagonal of $\mathbf{T}$ can track the singular values of $\mathbf{A}$ much better than the diagonal of $\mathbf{R}_1$ (see, e.g., [Ste99, §3]). One can find intuition for this by considering the similarities between the successive calls to QRCP with the successive calls to QR in the well-known *QR iteration*. In [FHH99], Stewart's UTV algorithm is even described as "half a QR iteration." Remarkably, this algorithm can be modified to interleave the computation of $\mathbf{R}_1$ with factoring $\mathbf{R}_1$ [Ste99, §5]. The resulting method, like QRCP, can be stopped early at a specified rank or once some accuracy metric is satisfied.

Complete Orthogonal Decomposition There is a notion of a UTV decomposition that is not the SVD, not QRCP, and yet predates Stewart's UTV by several decades. It is called the *complete orthogonal decomposition* (COD), and it is computed by one call to QRCP followed by one call to unpivoted QR [HL69]. The main use of a COD is to facilitate the application of a pseudoinverse $\mathbf{A}^\dagger$ when $\mathbf{A}$ is rank-deficient. We note that this is only modestly in line with the "spirit" of UTV, which asks for a decomposition that can be used as a surrogate for the SVD more generally. Still, the COD does have some historical importance in the development of randomized UTV algorithms.

Randomized algorithms

The first randomized algorithm for UTV was described in [DDH07, §5]. It used a random orthogonal transformation as a preconditioner for computing a COD, which made it safe to replace the usual call to QRCP with a call to unpivoted QR. This approach does not produce good surrogates for the SVD on its own, however, it has since been extended with power iteration ideas through the *PowerURV* algorithm [GM18, §3].[2] PowerURV is able to obtain better approximations of the SVD than Stewart's UTV without using any pivoted QR decompositions.

Much of the value in Stewart's algorithm for UTV is its ability to compute the decomposition incrementally. The earliest randomized algorithm for UTV that enjoys this capability is given in [MQH19, Figure 4]. Qualitatively, this algorithm can be thought of as extending the ideas of HQRRP without relying on HQRRP as a black box. In a historical context, it is notable because it is the first full-rank UTV algorithm to use sketching (i.e., random dimension reduction) rather than random rotations.

As we wrap up the discussion on this topic, we note that one can trivially incorporate randomization into Stewart's UTV by using HQRRP for the requisite QRCP calls. There would be a downside to this approach in that the diagonal entries of $\mathbf{T}$ would not be guaranteed to decrease across block boundaries. However, that downside could be circumvented by using HQRRP for the initial QRCP of $\mathbf{A}$ and then using a standard QRCP algorithm (e.g., LAPACK's `GEQP3`) for the QRCP of $(\mathbf{R}_1)^*$. The speedup of such an approach over Stewart's UTV would be fundamentally limited, but it should still be observable for $n \times n$ matrices even when n is as small as a few thousand.

[2] We note that the authors of [DDH07] were not trying to develop a randomized algorithm for its own sake. Rather, they used randomization as a tool to reduce many linear algebra problems to a format amenable to recursive unpivoted QR, which can be accelerated by black-box fast matrix multiplication methods.

5.2 Solving unstructured linear systems

Two broad methodologies have emerged for incorporating randomization into general linear solvers. The first aims to ameliorate the cost of common safeguards that are applied to fast but potentially unreliable direct methods. The second aims to restructure computations in existing general-purpose iterative methods. There is generally more excitement in the community for methods of this second kind, but methods of the first kind remain a subject of practical interest.

5.2.1 Direct methods

Direct methods for solving systems of linear equations center on finding a factored representation of the system matrix. Most famously, we have the *LU decomposition* of a general $n \times n$ matrix, which takes the form

$$\mathbf{A} = \mathbf{LU}$$

for a lower-triangular matrix $\mathbf{L}$ with unit diagonal ($L_{ii} = 1$ for all i) and an upper-triangular matrix $\mathbf{U}$. For Hermitian matrices, there is the *LDL decomposition*

$$\mathbf{A} = \mathbf{LDL}^*,$$

where $\mathbf{L}$ is unit lower-triangular and $\mathbf{D}$ is block diagonal with blocks of size one and two.

These are some of the most fundamental matrix decompositions. Once in hand, they can be used to solve linear systems involving $\mathbf{A}$ in $O(n^2)$ operations. The standard methods for their computation exhibit good data locality and are naturally adapted to parallel processing environments. However, these decompositions should be used cautiously; there are some nonsingular matrices for which they do not exist, or for which they cannot be computed stably in finite precision arithmetic. Therefore these decompositions need to be carefully modified to ensure reliability without sacrificing too much speed.

Stability through randomized pivoting

Pivoting is the standard paradigm to modify LU and LDL for improved numerical stability. For LU, we have partial pivoting and complete pivoting, which look like

$$\mathbf{PA} = \mathbf{LU} \qquad \text{and} \qquad \mathbf{P}_1\mathbf{AP}_2 = \mathbf{LU} \tag{5.2}$$

respectively, where $\mathbf{P}, \mathbf{P}_1, \mathbf{P}_2$ are permutation matrices.

The standard algorithms for computing these decompositions are Gaussian elimination with partial pivoting (GEPP) and Gaussian elimination with complete pivoting (GECP). While GEPP is substantially faster than GECP, it has weaker theoretical guarantees than GECP when it comes to numerical behavior. In [MG15], Melgaard and Gu propose a randomized algorithm for partially pivoted LU that makes pivoting decisions in a manner similar to HQRRP (see page 96). The randomized algorithm achieves efficiency comparable to that of GEPP, while also satisfying GECP-like element-growth bounds with high probability.

For LDL, pivoted decompositions take the form

$$\mathbf{A} = (\mathbf{PL})\mathbf{D}(\mathbf{PL})^*, \tag{5.3}$$

where (again) $\mathbf{D}$ is block-diagonal with blocks of size one and two. There are a variety of ways to introduce pivoting into LDL decompositions. The most notable are Bunch–Kaufman [BK77] and bounded Bunch–Kaufman (which uses rook pivoting) [AGL98], both of which are available in LAPACK. In [FXG18], Feng, Xiao, and Gu propose a randomized algorithm for pivoted LDL that is as stable as GECP and yet only slightly slower than Bunch–Kaufman and bounded Bunch–Kaufman.

Stability through randomized rotations

In Section 5.1.3, we mentioned how the first randomized algorithm for UTV used randomized preconditioning to compute a COD-like factorization using only *unpivoted* QR decompositions. This was not the first use of randomization to remove the need for pivoting in matrix decompositions. In fact, this idea was explored by Parker in 1995 to remove the need for pivoting in Gaussian elimination [Par95]. Here we summarize Parker's approach.

We begin by introducing some terms. For an integer $d \geq 1$, a *butterfly matrix* of size $2d \times 2d$ is a two-by-two block matrix, with $d \times d$ diagonal matrices in each of the four blocks. Speaking loosely, a *recursive butterfly transformation* (RBT) is a product of a chain of matrices, each with butterfly matrices as diagonal blocks. RBTs of order n (i.e., RBTs of size $n \times n$) are usually analyzed when n is a power of two for the sake of simplicity. The recursive structure in RBTs makes it possible to apply them with FFT-like methods. In particular, an RBT of order $n = 2^\ell$ can be applied to an n-vector in $O(n\ell)$ time. Detailed discussion on RBTs of general order can be found in [Pec21].

We are interested in RBTs that are *orthogonal* and *random*. The orthogonality is useful since it means the same FFT-like algorithms used to apply an RBT can be used to apply its inverse. The randomness in orthogonal RBT stems from how one chooses the entries in the diagonal matrices. While there are a variety of ways that this can be done [Par95], we simply speak in terms of a distribution $\mathcal{D}_n$ over orthogonal RBTs of order n.

One of the major contributions of [Par95] was to prove that for any nonsingular matrix $\mathbf{A}$ of order n, one can sample $\mathbf{B}_1, \mathbf{B}_2$ iid from a certain distribution $\mathcal{D}_n$, so that matrix $\mathbf{B}_1\mathbf{A}\mathbf{B}_2$ has an unpivoted LU decomposition with high probability. Put another way, the decomposition

$$\mathbf{A} = (\mathbf{B}_1)^*\mathbf{L}\mathbf{U}(\mathbf{B}_2)^*$$

exists with high probability.

The high speed at which RBTs can be applied and the excellent data locality properties of unpivoted matrix decompositions have led to substantial interest in RBTs from the HPC community. For example, implementation considerations for hybrid CPU/GPU machines were studied in [BDH+13] (in the single-node setting) and [LLD20] (in the distributed setting).

The idea of using RBTs to precondition an "unsafe" unpivoted method naturally applies to LDL. In this case, one obtains factorizations of the form

$$\mathbf{A} = (\mathbf{B}\mathbf{L})\mathbf{D}(\mathbf{B}\mathbf{L})^*$$

where $\mathbf{B}$ is the random RBT. Again, this methodology has received recent attention from the HPC community; see [BBB+14] for work in the multi-core distributed-memory setting [BBB+14] and [BDR+17] for work in the setting of a single machine with a hybrid CPU/GPU architecture.

Remarkably, although the idea of RBTs seems predicated on destroying sparsity structure present in the matrix $\mathbf{A}$, the random RBT methodology can be applied to sparse matrices without catastrophic fill-in. See [BLR14] for work on this topic for both general matrices and symmetric/Hermitian indefinite matrices.

5.2.2 Iterative methods

Background on GMRES

GMRES is a well-known iterative method for solving linear systems of the form $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$ where $\mathbf{A}$ is $n \times n$ and nonsingular. The trajectory $(\boldsymbol{x}_p)_{p\geq 1}$ it generates has a simple variational characterization. Specifically, $\boldsymbol{x}_p$ minimizes $L(\boldsymbol{x}) = \|\mathbf{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2$ over all vectors $\boldsymbol{x}$ in the p-dimensional *Krylov subspace*

$$K_p = \operatorname{span}\left\{\boldsymbol{b}, \mathbf{A}\boldsymbol{b}, \ldots, \mathbf{A}^{p-1}\boldsymbol{b}\right\}. \tag{5.4}$$

The standard implementation of GMRES uses the Arnoldi process. This can be seen as a specialization of (modified) Gram–Schmidt to orthogonalize implicitly-defined matrices of the form $\mathbf{K}_p = [\boldsymbol{b}, \mathbf{A}\boldsymbol{b}, \ldots, \mathbf{A}^{p-1}\boldsymbol{b}]$. In particular, as iterations proceed, the Arnoldi process maintains a column-orthonormal matrix $\mathbf{V}_p$ where $\operatorname{range}(\mathbf{V}_p) = K_p$. Optionally, it can also maintain an *Arnoldi decomposition*, which represents $\mathbf{A}\mathbf{V}_p = \mathbf{V}_{p+1}\mathbf{H}_p$ in terms of an $n \times (p+1)$ column-orthonormal matrix $\mathbf{V}_{p+1}$ and a $(p+1) \times p$ upper-Hessenberg matrix $\mathbf{H}_p$.[3]

Letting $T_{\mathrm{mv}}(\mathbf{A})$ denote the cost of a matrix-vector multiply with $\mathbf{A}$, the Arnoldi decomposition up to step p can be computed in time

$$O(pT_{\mathrm{mv}}(\mathbf{A}) + np^2).$$

If we are given this decomposition, then the least squares problem defining $\boldsymbol{x}_p$ can be solved in $O(np)$ time by applying a suitable direct method. Strictly speaking, one does not need to compute $\boldsymbol{x}_{p-1}$ to compute $\boldsymbol{x}_p$.

We summarize some ways to introduce randomness into GMRES below. They all work by relaxing the requirement that $\mathbf{V}_p$ be column-orthonormal while retaining the requirement that $\operatorname{range}(\mathbf{V}_p) = K_p$. Some of them work by changing the loss function $L(\boldsymbol{x})$ to be minimized by $\boldsymbol{x}_p$. These methods are of interest when the cost of the matrix-vector multiplies is dwarfed by the complexity of maintaining the Arnoldi decomposition. We note that this situation can only arise when $\mathbf{A}$ is a sparse or otherwise structured operator.

Randomized GMRES: Arnoldi decompositions in a sketch-orthogonal basis

The method from [BG22, § 4.2] can be interpreted as using a "sketched Arnoldi process" based on sketched Gram–Schmidt. It works by building up $\mathbf{V}_p$ so that its columns are $\mathbf{S}$-orthogonal in the sense of (5.1), where $\mathbf{S}$ is a $d \times n$ sketching operator ($p \lesssim d \ll n$). Along the way, it maintains an *Arnoldi-like decomposition* $\mathbf{A}\mathbf{V}_p = \mathbf{V}_{p+1}\mathbf{H}_p$, where $\mathbf{V}_{p+1}$ is likewise $\mathbf{S}$-orthogonal. Access to this decomposition at step p makes it possible to minimize the loss function $\|\mathbf{S}(\mathbf{A}\boldsymbol{x} - \boldsymbol{b})\|_2^2$ over all $\boldsymbol{x}$ in K_p in only $O(np)$ added time.

[3] A matrix is called upper-Hessenberg if all entries below the first subdiagonal are zero.

To understand the quality of the solution obtained by this method it is helpful to consider the unconstrained formulation

$$\min_{\boldsymbol{z}} \|\mathbf{A}\mathbf{V}_p\boldsymbol{z} - \boldsymbol{b}\|_2^2. \tag{5.5}$$

GMRES would return $\boldsymbol{x}_\star = \mathbf{V}_p\boldsymbol{z}_\star$ where $\boldsymbol{z}_\star$ solves (5.5) exactly. The sketched Arnoldi approach effectively approximates this solution by applying sketch-and-solve to (5.5). This puts us in a position to draw from our coverage of sketch-and-solve in Section 3.2.1. If δ is the effective distortion of $\mathbf{S}$ for the subspace K_{p+1}, then the solution $\boldsymbol{x}_{\text{sk}}$ obtained by the sketched Arnoldi approach will satisfy $\|\mathbf{A}\boldsymbol{x}_{\text{sk}} - \boldsymbol{b}\|_2 \leq (1+\delta)\|\mathbf{A}\boldsymbol{x}_\star - \boldsymbol{b}\|_2$.

The big-O time complexity of the sketched Arnoldi process is unchanged relative to the classic Arnoldi process. However, the flop count for the sketched process can be up to a factor of two smaller. The sketched process also makes better use of BLAS 2 over BLAS 1, and it has fewer synchronization points compared to the classic Arnoldi process based on modified Gram–Schmidt. Taken together, using the sketched process can significantly reduce the wallclock time needed to obtain the decomposition of $\mathbf{A}\mathbf{V}_p$ while retaining the reliability of classic Arnoldi.

We note that a block version of this algorithm (for linear systems with multiple right-hand sides) is presented in the preprint [BG21]. For MATLAB implementations of these methods, see [Bal22a].

Randomized GMRES: handling general non-orthogonal bases

Both classic GMRES and the randomized variant given above maintain Arnoldi-like decompositions of matrices $\mathbf{A}\mathbf{V}_p$ at a cost of $O(np^2)$ time complexity. Interestingly, this cost cannot be asymptotically reduced by forgoing the decomposition of $\mathbf{A}\mathbf{V}_p$. The trouble is that building $\mathbf{V}_p$ with full orthogonalization – in the standard sense or the $\mathbf{S}$-orthogonal sense – already takes $O(np^2)$ time.

In [NT21], Nakatsukasa and Tropp identified that (5.5) has precisely the form needed to benefit from randomized algorithms, independent from how $\mathbf{V}_p$ and $\mathbf{A}\mathbf{V}_p$ are generated. Based on this observation they called attention to longstanding classical methods for computing non-orthogonal bases of Krylov subspaces. For example, one can compute $\mathbf{V}_p$ by a truncated k-step Arnoldi process for some $k \ll p$. This can be done in $O(npk)$ time and can easily be implemented to provide a dense representation of $\mathbf{A}\mathbf{V}_p$ at no added cost. Alternatively, it may be practical to use the Chebyshev method if one has knowledge of the spectrum of $\mathbf{A}$.

[NT21] primarily advocates for approximately solving (5.5) via sketch-and-solve, where the sketched subproblem is handled by factoring $\mathbf{S}\mathbf{A}\mathbf{V}_p$. Note that in exact arithmetic the solutions obtained from this method would coincide with those of GMRES based on sketched Arnoldi. On the one hand this is very appealing, since the cost of running this method for p iterations can *easily* undercut the $O(np^2)$ cost of sketched Arnoldi. On the other hand, the behavior of these methods can differ in finite-precision arithmetic. If one is too lax in building the basis matrix $\mathbf{V}_p$ then the condition number of $\mathbf{A}\mathbf{V}_p$ can explode as p increases.

All in all, the design space for this methodology is large and worth navigating with care. Valuable advice in this regard is given throughout [NT21, §3 – §5]. One particularly compelling comment is that one could simply solve (5.5) to high accuracy via a sketch-and-precondition method, such as Algorithm 1. The resulting solution in this case would be very close to that produced by GMRES.

Nested randomization in block-projection and block-descent methods

Having discussed GMRES at length, we now speak to a family of iterative solvers that do not use the Krylov subspace approach.

This family came into focus with the development of *sketch-and-project* – a template iterative algorithm for solving linear systems of the form $\mathbf{F}\boldsymbol{z} = \boldsymbol{g}$, where $\mathbf{F} \in \mathbb{R}^{M\times m}$ has at least as many rows as columns ($M \geq m$) [GR15]. Its special cases include randomized Kaczmarz [SV08] and randomized block Kaczmarz [NT14]. It also has variants that are specifically designed for overdetermined least squares problems [GIG21].

Without getting into the mechanics of sketch-and-project in detail, we note that these methods share a significant weakness: their convergence rates worsen as one considers larger and larger problems. We think they are most likely to be useful when one cannot fit an $m \times m$ matrix in memory. While such situations fall outside our primary data model, the *subproblems* encountered in sketch-and-project are amenable to methods we have covered. Indeed, the subproblems are equivalent to problems of the form

$$\min_{\boldsymbol{y}\in\mathbb{R}^m} \{\|\boldsymbol{y} - \boldsymbol{b}\|_2^2 \,:\, \mathbf{A}^*\boldsymbol{y} = \boldsymbol{c}\}, \qquad \text{((3.4), revisited)}$$

where the number of columns n in $\mathbf{A}$ is a user-selected tuning parameter $n \ll m \leq M$. Such problems are clearly amenable to Algorithm 2.

Recently, a general analysis framework for randomized linear system solvers based on block projection or block descent has been proposed [PJM22]. We refer the reader to Table 3 of [PJM22] (and appendices A.15 – A.26) for an extensive list of new and old randomized linear system solvers that are amenable to their proposed analysis framework. Some of these methods are distinguished in their applicability to underdetermined problems. As with sketch-and-project, the subproblems encountered in essentially all of these methods can be chosen to have a structure amenable to Algorithm 2.

5.3 Trace estimation

Many scientific computing and machine learning applications require estimating the trace of a square linear operator $\mathbf{A}$ that is represented implicitly. Randomized methods are especially effective for such problems.

5.3.1 Trace estimation by sampling

Let $\mathbf{A}$ be $n \times n$ and $\{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n\}$ be the standard basis vectors in $\mathbb{R}^n$. Clearly, one can compute the trace of $\mathbf{A}$ with n matrix-vector products by using the identity

$$\operatorname{tr}(\mathbf{A}) = \sum_{i=1}^{n} \boldsymbol{e}_i^* \mathbf{A} \boldsymbol{e}_i.$$

Randomization creates opportunities to estimate this quantity using $m \ll n$ matrix-vector multiplications. The most basic method uses the fact that if $\boldsymbol{\omega} \sim \mathcal{D}$ is a random vector satisfying $\mathbb{E}[\boldsymbol{\omega}\boldsymbol{\omega}^*] = \mathbf{I}_n$, then

$$\operatorname{tr}(\mathbf{A}) = \mathbb{E}\left[\boldsymbol{\omega}^* \mathbf{A} \boldsymbol{\omega}\right].$$

It is natural to approximate the expected value by the empirical mean. That is, upon drawing m independent vectors $\boldsymbol{\omega}_i \sim \mathcal{D}$, we estimate

$$\operatorname{tr}(\mathbf{A}) \approx \frac{1}{m} \sum_{i=1}^{m} \boldsymbol{\omega}_i^* \mathbf{A} \boldsymbol{\omega}_i. \tag{5.6}$$

The idea for this method goes back to 1987 with work by Girard [Gir87], who proposed that $\mathcal{D}$ be the uniform distribution over the ℓ_2 hypersphere with radius $\sqrt{n}$. Shortly thereafter, Hutchinson proposed that one take $\mathcal{D}$ as a distribution over Rademacher random vectors [Hut90]. Hutchinson's choice of $\mathcal{D}$ minimizes the variance of the estimator when $\mathbf{A}$ is fixed, while Girard's choice minimizes the worst-case variance over sets of matrices that are closed under conjugation by unitary matrices; see [Epp23] for an explanation of this point.

We call the right-hand side of (5.6) a *Girard–Hutchinson estimator*. Such estimators require $m \in \Omega(1/\epsilon^2)$ samples to approximate $\operatorname{tr}(\mathbf{A})$ to within ϵ error for some constant failure probability.

5.3.2 Trace estimation with help from low-rank approximation

Compress and trace

In [SAI17], Saibaba, Alexanderian, and Ipsen propose two randomized algorithms for estimating the trace of a psd linear operator $\mathbf{A}$.

When $\mathbf{A}$ is accessible by matrix-vector products, the proposed method begins with a rangefinder step to find a column-orthonormal $n \times m$ matrix $\mathbf{Q}$ where $\mathbf{Q}\mathbf{Q}^*\mathbf{A}\mathbf{Q}\mathbf{Q}^* \approx \mathbf{A}$. The method then approximates

$$\operatorname{tr}(\mathbf{A}) \approx \operatorname{tr}(\mathbf{Q}\mathbf{Q}^*\mathbf{A}\mathbf{Q}\mathbf{Q}^*) = \operatorname{tr}(\mathbf{Q}^*\mathbf{A}\mathbf{Q}).$$

Whether or not this bound is accurate depends on the rate of $\mathbf{A}$'s spectral decay and on how well-aligned $\mathbf{Q}$ is with the dominant eigenvectors of $\mathbf{A}$. This method can provide for better relative error bounds than a Girard–Hutchinson estimator if $\mathbf{A}$'s spectral decay is sufficiently fast and $\mathbf{Q}$ is obtained by power iteration.

Trace estimation is especially challenging when matrix-vector products with $\mathbf{A}$ are expensive. This often happens when $\mathbf{A}$ is the image of another matrix $\mathbf{B}$ under a matrix function, in the sense of [Hig08]. Saibaba *et al.* consider the case where

$$\mathbf{A} = \log(\mathbf{I} + \mathbf{B})$$

for a psd matrix $\mathbf{B}$.[4] The idea here is again to find a tall $n \times m$ column-orthonormal $\mathbf{Q}$ so that $\mathbf{Q}\mathbf{Q}^*\mathbf{B}\mathbf{Q}\mathbf{Q}^*$ is a good low-rank approximation of $\mathbf{B}$. Then we approximate

$$\operatorname{tr}(\mathbf{A}) \approx \sum_{i=1}^{m} \log\left(1 + \lambda_i(\mathbf{Q}^*\mathbf{B}\mathbf{Q})\right)$$

where $\lambda_i(\cdot)$ returns the i^{th}-largest eigenvalue of the given matrix. Error bounds can be obtained for this estimate under suitable assumptions on the spectral decay of $\mathbf{B}$. We note that some of the techniques used to prove these bounds extend to any matrix function that is operator-monotone, such as the matrix square-root.

[4] For any positive definite matrix $\mathbf{M}$, we use $\log(\mathbf{M})$ to denote the Hermitian matrix with the same eigenvectors as $\mathbf{M}$ and whose eigenvalues are the logs of the eigenvalues of $\mathbf{M}$. One can verify that $\operatorname{tr}(\log(\mathbf{M})) = \log\det\mathbf{M}$ holds for any positive definite $\mathbf{M}$.

Split, trace, and approximate

In [MMM+21], Meyer et al. combined ideas from low-rank approximation with the Girard–Hutchinson estimator to obtain `Hutch++`. This estimator starts by sampling a matrix $\mathbf{Q}$ uniformly at random from the set of $n \times m$ column-orthonormal matrices. It then defines the low-rank approximation $\hat{\mathbf{A}} = \mathbf{Q}\mathbf{Q}^*\mathbf{A}\mathbf{Q}\mathbf{Q}^*$ and computes the trace of this approximation by the formula

$$\operatorname{tr}(\hat{\mathbf{A}}) = \operatorname{tr}(\mathbf{Q}^*\mathbf{A}\mathbf{Q}).$$

The last phase of `Hutch++` applies Girard–Hutchinson to the deflated matrix

$$\mathbf{\Delta} = (\mathbf{I} - \mathbf{Q}\mathbf{Q}^*)\mathbf{A}(\mathbf{I} - \mathbf{Q}\mathbf{Q}^*),$$

and adds this estimate to $\operatorname{tr}(\hat{\mathbf{A}})$.

The basic validity of `Hutch++` follows by splitting the trace of $\mathbf{A}$ into two parts:

$$\operatorname{tr}(\mathbf{A}) = \operatorname{tr}(\hat{\mathbf{A}}) + \operatorname{tr}(\mathbf{A} - \hat{\mathbf{A}})$$

and verifying that $\operatorname{tr}(\mathbf{A} - \hat{\mathbf{A}}) = \operatorname{tr}(\mathbf{\Delta})$. As a splitting and deflation approach, this method is very effective in reducing the variance of the Girard–Hutchinson estimator. Early results along these lines can be found in [GSO17], which investigated the use of deflation in estimating the trace of an inverted matrix.

The initial results proven for `Hutch++` applied only to psd matrices. In that context, `Hutch++` can (with some small *fixed* failure probability) compute $\operatorname{tr}(\mathbf{A})$ to within ϵ relative error using only $O(1/\epsilon)$ matrix-vector products. This is a substantial improvement upon the $O(1/\epsilon^2)$ matrix-vector products that are required by plain Girard–Hutchinson estimators. In fact, the sample complexity of `Hutch++` cannot be improved when considering a large class of algorithms [MMM+21, Theorems 4.1 and 4.2].

Persson, Cortinovis, and Kressner have since extended `Hutch++` so that it can proceed adaptively, only terminating once some error tolerance has been achieved (up to a *controllable* failure probability) [PCK21]. The analysis of their modified `Hutch++` method notably accommodates symmetric indefinite matrices $\mathbf{A}$. We note that the accuracy guarantees of trace estimators for indefinite matrices cannot be as strong as those for positive definite matrices. Indeed, relative error guarantees are essentially impossible when $\operatorname{tr}(\mathbf{A}) = 0$. Persson et al., therefore, provide additive error guarantees in this setting.

Leveraging the exchangability principle

In [ETW23], Epperly, Tropp, and Webber develop a trace estimator based on the *exchangability principle*. In the context of trace estimation, this principle stipulates that if an algorithm computes its estimate based on m pairs $\{(\boldsymbol{\omega}_i, \mathbf{A}\boldsymbol{\omega}_i)\}_{i=1}^m$ where $\boldsymbol{\omega}_i$ are iid random vectors, then the minimum-variance unbiased estimator for $\operatorname{tr}(\mathbf{A})$ must be invariant under relabelings $\{\boldsymbol{\omega}_i\}_{i=1}^m \leftarrow \{\boldsymbol{\omega}_{\sigma(i)}\}_{i=1}^m$ for permutations σ.

`Hutch++` does not respect the exchangability principle, since it uses randomness in two distinct stages: first to compute the matrix $\mathbf{Q}$ and then to estimate the trace of the $\mathbf{\Delta}$ by a a Girard–Hutchinson estimator.

The `XTrace` algorithm proposed in [ETW23] can be thought of as a symmetrized version of `Hutch++`. Given m samples $\{(\boldsymbol{\omega}_i, \mathbf{A}\boldsymbol{\omega}_i)\}_{i=1}^m$, its estimate is an average of m runs of `Hutch++`, where the j^{th} run uses $\mathbf{Q}_j = \texttt{orth}([\mathbf{A}\boldsymbol{\omega}_i]_{i \neq j})$ and estimates $\operatorname{tr}(\mathbf{\Delta}_j)$

by $\boldsymbol{\omega}_j^* \boldsymbol{\Delta}_j \boldsymbol{\omega}_j$. Implementing `XTrace` naively would be very expensive. However, as explained in [ETW23, §2.1], a careful implementation can achieve the same asymptotic complexity as `Hutch++`. `XTrace` also comes with adaptive-stopping and variance estimation methods analogous to those developed in [PCK21].

5.3.3 Estimating the trace of $f(\mathrm{B})$ via integral quadrature

Section 5.3.2 touched on a method for estimating the trace of $\mathbf{A} = \log(\mathbf{B} + \mathbf{I})$, where $\mathbf{B}$ is psd and $\log(\cdot)$ is the matrix logarithm. This section covers powerful methods for a broader class of trace estimation problems. The original method, now known as *stochastic Lanczos quadrature* (SLQ), was introduced to the linear algebra community in [BFG96], was popularized by [UCS17], and has since extended in a few different ways [CH22; PK22; CTU22].

To begin, consider how any function $f : \mathbb{R} \to \mathbb{R}$ can canonically be extended to act on a Hermitian matrix by acting separately on the eigenvalues of the matrix. That is, if we expand $\mathbf{B}$ in its eigenbasis

$$\mathbf{B} = \sum_{i=1}^{n} \lambda_i \boldsymbol{u}_i \boldsymbol{u}_i^*,$$

then we can define

$$f(\mathbf{B}) = \sum_{i=1}^{n} f(\lambda_i) \boldsymbol{u}_i \boldsymbol{u}_i^*.$$

Here we cover quadrature-based methods for approximating the trace of such matrices. The concepts behind these methods apply whenever f is sufficiently smooth and $\mathbf{B}$ is Hermitian. Theoretical guarantees for these methods are usually obtained under stronger assumptions, such as f being analytic on $[\lambda_n, \lambda_1]$, or $\mathbf{B}$ being psd.

Technical background

The concepts we summarize below are detailed in the book [GM10].

Riemann-Stieltjes integrals. Let μ be a real-valued function on $\mathbb{R}$. The expression

$$\int_{\mathbb{R}} f(t) \mathrm{d}\mu(t) \tag{5.7}$$

is called the *Riemann-Stieltjes integral* of f against μ. We do not provide a formal definition of this integral. Rather, we offer two footholds for understanding it. First, if μ is continuously differentiable, then (5.7) is simply the Riemann integral of $t \mapsto f(t)(\frac{\mathrm{d}}{\mathrm{d}t}\mu(t))$. Second, recall the interpretation of Riemann integration in which one identifies $\mathrm{d}t \approx t_{\ell+1} - t_\ell$, where $t_\ell < t_{\ell+1}$ are consecutive points in a partition of the region of integration. If the analogous interpretation is applied to (5.7), then we would say that $\mathrm{d}\mu(t) \approx \mu(t_{\ell+1}) - \mu(t_\ell)$.

For our purposes we can assume that μ is nondecreasing. We also assume that there are constants L and U where $\mu(t) = \mu(L)$ for all $t \le L$ and $\mu(t) = \mu(U)$ for all $t \ge U$. Under these assumptions, (5.7) is well-defined whenever f is continuous.

Quadrature and orthogonal polynomials. An *s-point quadrature rule* for (5.7) specifies s-vectors $\boldsymbol{w}$ and $\boldsymbol{\theta}$ (of weights and nodes respectively) to define an approximation

$$\int_{\mathbb{R}} f(t)\mathrm{d}\mu(t) \approx \sum_{\ell=1}^{s} w_\ell f(\theta_\ell). \tag{5.8}$$

The nodes and weights selected by any reliable quadrature method will depend on μ. One prominent approach to defining quadrature rules is to require that (5.8) holds with equality whenever f is a polynomial of degree d, where d is suitably bounded in terms of s. The idea behind this is that s increases, we should be able to accommodate polynomials of higher degree.

Gaussian quadrature achieves optimal sample complexity. This method is exact for polynomials up to degree $2s-1$, and there is no rule that can guarantee exactness for polynomials of degree higher than $2s-1$ with only s samples. There is a deep connection between Gaussian quadrature and orthogonal polynomials that make this quadrature rule viable in practice. The connection uses the fact that, under our assumptions on μ, it can be taken to define an inner product

$$\langle p, q\rangle_\mu = \int_{\mathbb{R}} p(t)q(t)\mathrm{d}\mu(t).$$

This inner product can be used to define an orthonormal basis for the set of polynomials that have at most some prescribed degree. When this orthonormal basis is sorted by degree, the resulting sequence of polynomials must satisfy a three-term recurrence relationship [GM10, §2]. The coefficients of the recurrence relationship up to step s can be assembled in a tridiagonal matrix J, called the Jacobi matrix, of size $s \times s$. The nodes and weights of Gaussian quadrature against μ can be recovered from an eigendecomposition of the Jacobi matrix [GM10, Theorem 6.2].

Stochastic Lanczos quadrature : approximating Girard–Hutchinson

A Girard–Hutchinson estimator for the trace of $f(\mathbf{B})$ takes the form

$$T = \frac{1}{m}\sum_{i=1}^{m} T_i \quad \text{where} \quad T_i = \boldsymbol{\omega}_i^* f(\mathbf{B})\boldsymbol{\omega}_i$$

for independent random vectors $\boldsymbol{\omega}_i$ drawn from a suitable distribution. The naive way to compute this estimator would be to call a black-box function that implements the action of $f(\mathbf{B})$; for each sample i one would compute $\boldsymbol{v}_i = f(\mathbf{B})\boldsymbol{\omega}_i$ and then take a dot product $T_i = \boldsymbol{\omega}_i^*\boldsymbol{v}_i$. Here we describe an alternative approach which begins with an integral representation for T_i and then approximates that integral via Gaussian quadrature [BFG96]. The resulting method for trace estimation is now known as stochastic Lanczos quadrature (SLQ) [UCS17].

Integral representation of a single sample. Let u denote the piecewise constant function that is zero for $t \le 0$ and one for $t > 0$. This function can be used to define a Riemann-Stieltjes integral that samples f at any prescribed point. Specifically, for any scalar z, we have $f(z) = \int_{\mathbb{R}} f(t)\mathrm{d}u(t-z)$. Therefore upon setting

$$\mu_i(t) = \sum_{j=1}^{n} |\boldsymbol{\omega}_i^*\boldsymbol{u}_j|^2\, u(t-\lambda_j), \tag{5.9}$$

the following identity is immediate from the definition of $f(\mathbf{B})$:

$$T_i = \int_{\mathbb{R}} f(t) \mathrm{d}\mu_i(t). \tag{5.10}$$

We note that this integral is written as being over all of $\mathbb{R}$, but it would suffice to integrate over the interval $[\lambda_n, \lambda_1]$.

Quadrature of a sample's integral representation. Integration is often described as a continuous analog of summation. As such, one usually thinks of quadrature as an act of approximating a continuous operation by a discrete operation. Quadrature of Riemann-Stieltjes integrals does not always follow this pattern. Indeed, the integral (5.10) can already be expressed as a weighted sum of $s = n$ point evaluations of f, with weights $w_{i\ell} = |\boldsymbol{\omega}_i^* \boldsymbol{u}_\ell|^2$ and nodes $\theta_{i\ell} = \lambda_\ell$. The problem with this representation is that we do not know the weights or nodes a-priori. Therefore in the setting of Riemann-Stieltjes integration it is possible that quadrature acts as a means of approximating an unknown integrator μ_i by a known integrator $\hat{\mu}_i$ for which we can efficiently compute $\int_{\mathbb{R}} f(t) \mathrm{d}\hat{\mu}_i(t)$.

Enter, *Lanczos quadrature*. This is a method for computing the Gaussian quadrature rule (or variations thereof) of Riemann-Stieltjes integrals with integrators of the form (5.9) [GM10, §7]. It uses the fact that the polynomials that are orthogonal with respect to the integrator μ_i are none other than the Lanczos polynomials associated with $(\mathbf{B}, \boldsymbol{\omega}_i)$ (see [GM10, Theorem 4.2]). Hence, the Lanczos algorithm for computing an orthonormal basis for the s-dimensional Krylov subspace

$$\operatorname{span}\{\boldsymbol{\omega}_i, \mathbf{B}\boldsymbol{\omega}_i, \ldots, \mathbf{B}^{(s-1)}\boldsymbol{\omega}_i\}$$

can be used to compute the Jacobi matrix. Given that, standard tridiagonal eigensolvers can provide us with the nodes and weights needed for Gaussian quadrature.

Implementation notes. SLQ entails approximating m samples of the form $\boldsymbol{\omega}_i^* f(\mathbf{B})\boldsymbol{\omega}_i$, where each $\boldsymbol{\omega}_i$ is an independent random vector drawn from some distribution $\mathcal{D}$. The quality of each approximate sample depends on the number of nodes allowed in the Gaussian quadrature rule, and hence on the number of steps in the Lanczos algorithm.

Taking s steps of the Lanczos algorithm will always require $s-1$ matrix-vector products with $\mathbf{B}$. The arithmetic and storage complexity needed to compute each sample in SLQ depends on whether we run Lanczos proper or a version of Lanczos that only computes the data needed for Gaussian quadrature. Indeed, in the latter case we have a substantial amount of freedom to make tradeoffs between computational complexity and numerical stability. At one end this tradeoff, s iterations of Lanczos with full orthogonalization costs $O(ns)$ storage and $O(ns^2)$ arithmetic. At the other end of the tradeoff, performing no reorthogonalization reduces the costs to only $O(n)$ storage and $O(ns)$ arithmetic. (We emphasize that these costs do not account for the $s-1$ matrix-vector products needed with $\mathbf{B}$.)

SLQ is a powerful tool, with important applications in Gaussian process regression. For an implementation of this method that scales to petascale problems by running on GPU farms, we refer the reader to the IMATE Python package [Ame22].

Beyond stochastic Lanczos quadrature

Accelerated quadrature-based methods. Let $\mathbf{A} = f(\mathbf{B})$. The convergence rate of SLQ for estimating $\operatorname{tr}(\mathbf{A})$ can only be as good as a Girard–Hutchinson estimator. As such, one needs $m \in \Omega(1/\epsilon^2)$ samples in order to estimate $\operatorname{tr}(\mathbf{A})$ to within ϵ error for some constant failure probability. This leaves substantial improvement for SLQ in the case when $\mathbf{A}$ is positive definite, where `Hutch++` could make do with $m \in \Omega(1/\epsilon)$ queries to $\mathbf{A}$. Luckily, it is possible to extend SLQ to use similar splitting techniques that `Hutch++` employs for its variance reduction; see [CH22] and [PK22] for details.

Spectral density estimation. One of SLQ's remarkable properties is that its quadrature rule for approximating (5.10) does not depend on f. As such, if the quadrature nodes and weights are computed to estimate $\operatorname{tr}(f(\mathbf{B}))$ for one function f, then one can use those same nodes and weights to compute an estimate for $\operatorname{tr}(g(\mathbf{B}))$ for another function g. This gives some motivation for directly estimating the function

$$\phi(t) = \sum_{j=1}^{n} u(t - \lambda_i) \tag{5.11}$$

which satisfies $\int_{\mathbb{R}} f(t)\mathrm{d}\phi(t) = \operatorname{tr}(f(\mathbf{B}))$ for all continuous functions $f : \mathbb{R} \to \mathbb{R}$. Note that ϕ is a nonnegative nondecreasing function with $\lim_{t\to\infty} \phi(t) = n$. As such, ϕ/n is a cumulative probability distribution function that can be uniquely identified with the spectrum of $\mathbf{B}$.

The problem of estimating a function of the form (5.11) is a particular case of *spectral density estimation*. This problem, which has broader applications in the physical sciences than trace estimation, has been approached explicitly with randomized algorithms [Lin16]. Approaching the linear algebraic problem of trace estimation with this in mind can lead to new insights on how to leverage prior knowledge on the structure of f or $\mathbf{B}$ for algorithmic purposes. In particular, [CTU22] provides a systematic treatment of quadrature-based trace estimation algorithms based on this perspective.

Section 6

Advanced Sketching: Leverage Score Sampling

Leverage scores quantify the extent to which a low-dimensional subspace aligns with coordinate subspaces. They are fundamental to RandNLA theory since they determine how well a matrix can be approximated through sketching by row or column selection, and thus indirectly how well a matrix can be approximated by sparse data-oblivious sketching methods [DM16]. They have algorithmic uses in least squares [DMM06; DMM+12] and low-rank approximation [DMM08; BMD09; MD16] among other topics. More broadly, they play a key role in statistical regression diagnostics [CH88; MMY15].

The computational value of leverage scores stems from how they induce data-aware probability distributions over the rows or columns of a matrix. *Leverage score sampling* refers to sketching by row or column sampling according to a leverage score distribution (or an approximation thereof). The quality of sketches produced by leverage score sampling is relatively insensitive to numerical properties of the matrix to be sketched. This can be contrasted with sketching by uniform row or column sampling, which can perform very poorly on certain families of matrices.

Leverage score distributions can be computed exactly with standard deterministic algorithms. However, exact computation is expensive except in very specific cases (see Section 7). Therefore in practice it is necessary to use randomized algorithms to *approximate* leverage score distributions. On the one hand, this point is significant since the costs of the approximation algorithms undermine the efficiency gains obtained from sketching by simple row or column selection, making the cost comparable to implementing data-oblivious random projection methods. On the other hand, uniform sampling is clearly suboptimal in many cases, e.g., in that it can miss important nonuniformity structures needed to obtain data-aware subspace embeddings. In general, the practical utility of leverage scores derives from when row or column selection of a matrix is *required* by a particular application. Leverage scores, therefore, compete with both uniform sampling and other methods for column (or row) selection as discussed in Section 4.3.4.

We emphasize that we have made no concrete plans regarding RandLAPACK's support for leverage score sampling methods. We review them here since they are prominent and sophisticated sketching methods, and they *might be* appropriate to support in RandLAPACK via a suite of computational routines.

In what follows we introduce three flavors of leverage scores (§6.1) and methods for approximately computing them (§6.2). We also cover three special topics: Section 6.3.1 explains how leverage scores can be used to define long-axis-sparse sketching operators (in the sense of Section 2.4.2), and Sections 6.3.2 and 6.3.3 discuss generalizations of leverage scores.

6.1 Definitions and background

Here we cover three types of leverage scores and corresponding approaches to leverage score sampling. The first type of leverage score (which we mean by default) is applicable to sketching in the embedding regime. As such, it is applicable primarily to highly overdetermined least squares problems or other saddle point problems with tall data matrices. We spend more time on this first type of leverage score since it has theoretical value in understanding the behavior of RandNLA algorithms. The second type is used for sketching in the sampling regime and has applications in a variety of low-rank approximation problems. The third type is specifically for approximating psd matrices (typically kernel matrices) in the presence of explicit regularization.

6.1.1 Standard leverage scores

Let U be an n-dimensional linear subspace of $\mathbb{R}^m$ and $\mathbf{P}_U$ be the orthogonal projector from $\mathbb{R}^m$ to U. The i^{th} *leverage score* of U is

$$\ell_i(U) = \|\mathbf{P}_U \boldsymbol{\delta}_i\|_2^2 = \mathbf{P}_U[i,i]. \tag{6.1}$$

where $\boldsymbol{\delta}_i$ is the i^{th} standard basis vector. Collectively, leverage scores describe how well the subspace U aligns with the standard basis in $\mathbb{R}^m$. They have algorithmic implications when we consider induced *leverage score distributions*, defined by

$$p_i(U) = \frac{\ell_i(U)}{\sum_{j=1}^m \ell_j(U)} = \frac{\ell_i(U)}{n}. \tag{6.2}$$

Given a matrix $\mathbf{A}$, one can associate as many sets of leverage scores to that matrix $\mathbf{A}$, as one can associate subspaces to $\mathbf{A}$. Two of the most important such subspaces are $U = \text{range}(\mathbf{A})$ and $V = \text{range}(\mathbf{A}^*)$. In these contexts we say that the leverage score for the i^{th} row of $\mathbf{A}$ is $\ell_i(U)$, while the leverage score for the j^{th} column is $\ell_j(V)$. Such leverage scores provide leverage score distributions over the rows and columns of $\mathbf{A}$, respectively. Note that only one of these distributions can be nonuniform if $\mathbf{A}$ is full-rank. Therefore when speaking of leverage scores we typically assume the $m \times n$ matrix $\mathbf{A}$ is tall, which allows for the possibility that $p(U)$ is nonuniform.

Moving forward, we routinely replace U by $\mathbf{A}$ in (6.1) and (6.2), with the understanding that $U = \text{range}(\mathbf{A})$.

Probabilistic guarantees of sketching via row sampling

Suppose $\mathbf{S}$ is a wide $d \times m$ sketching operator that implements row sampling according to a probability distribution $\boldsymbol{q}$. We are interested in evaluating the statistical quality of $\mathbf{S}$ as a row sampling operator for an $m \times n$ matrix $\mathbf{A}$. Here, our measure of sketch quality the smallest $\epsilon \in (0, 1)$ where $\boldsymbol{y} \in \text{range}(\mathbf{A})$ implies

$$(1 - \epsilon)\|\boldsymbol{y}\|_2^2 \leq \|\mathbf{S}\boldsymbol{y}\|_2^2 \leq (1 + \epsilon)\|\boldsymbol{y}\|_2^2. \tag{6.3}$$

Note that this metric is very similar to subspace embedding distortion. In this monograph we have generally advocated for measuring sketch quality by a scale-invariant metric called *effective distortion*. Despite this, we care about (6.3) since it provides for the following standard result (which we prove in Appendix A.3).

Proposition 6.1.1. *Suppose* $\mathbf{A}$ *is an* $m \times n$ *matrix of rank* n*. If*

$$r := \min_{j \in [\![m]\!]} \frac{q_j}{p_j(\mathbf{A})}$$

then for all $0 < \epsilon < 1$*, we have*

$$\Pr\{(6.3) \text{ fails for } (\mathbf{S}, \mathbf{A}, \epsilon)\} \leq 2n \left(\frac{\exp(\epsilon)}{(1 + \epsilon)^{(1+\epsilon)}} \right)^{rd/n} \tag{6.4}$$

and $\exp(\epsilon) < (1 + \epsilon)^{(1+\epsilon)}$.

The proposition's basic message is that the probability of $\mathbf{SA}$ being a good sketch improves as $\boldsymbol{q}$ gets closer to the leverage score distribution $p(\mathbf{A})$, where "closer" means that the value r becomes larger. This makes it desirable for $\boldsymbol{q}$ to approximate the leverage score distribution. In practice, such approximations would be obtained by first estimating leverage scores (e.g., via the method described in Section 6.2.1) and then normalizing according to the estimates. That is, we compute $\hat{\ell}$ as an estimate of $\ell(\mathbf{A})$, then set

$$q_i = \frac{\hat{\ell}_i}{\sum_{j=1}^m \hat{\ell}_j}.$$

With this in mind, we turn to our next question: how large should d be so that the failure probability (6.4) tends to zero as n tends to infinity?

As a short answer, it can be shown that taking $d \in O\left(n \log n / r\epsilon^2\right)$ is sufficient for (6.4) to tend to zero as n tends to infinity.[1] With (exact) leverage score sampling we are fortunate to have $r = 1$, and so it suffices for the embedding dimension to satisfy

$$d_{\text{lev}} \in O\left(\frac{n \log n}{\epsilon^2}\right). \tag{6.5}$$

To describe the bound with uniform sampling, we introduce the *coherence of* $\mathbf{A}$ as

$$\mathscr{C}(\mathbf{A}) := m \max_{i \in [\![m]\!]} \ell_i(\mathbf{A}).$$

It is easily be shown that coherence is bounded by $n \leq \mathscr{C}(\mathbf{A}) \leq m$ and that uniform sampling leads to $r = n/\mathscr{C}(\mathbf{A})$. In view of these facts, the embedding dimension for uniform sampling should be on the order of

$$d_{\text{unif}} \in O\left(\frac{\mathscr{C}(\mathbf{A}) \log n}{\epsilon^2}\right).$$

This is no better than leverage score sampling, and it can be *much* worse.

As a final point on the effectiveness of sketching by row selection methods, consider the situation of using *approximate* leverage scores where we have a bound $q_j \geq \beta p_j(\mathbf{A})$ for all j. In such a situation we would have $\beta \leq r$ and setting $d = d_{\text{lev}}/\beta$ would suffice to achieve the same guarantees as leverage score sampling.

Preconditioned leverage score sampling, hidden in plain sight

Many data-oblivious sketching operators can be described as applying a "rotation" and then performing coordinate subsampling. Here are two such examples.

- A wide $d \times m$ Haar sketching operator $\mathbf{S}$ can be viewed as a composition of an $m \times m$ orthogonal matrix followed by a coordinate sampling operator.
- The diagonal sign flip and the fast trig transform in an SRFT amounts to a rotation, and the full action of the SRFT is just applying coordinate sampling to the rotated input.

In both cases, the rotation acts as a type of preconditioner for sampling, i.e., as a transformation that converts a given problem into a related form that is more suitable for sampling methods [DM16]. The example of SRFTs is especially informative, since using an embedding dimension $d \in O(n \log n)$ suffices for a $d \times m$ SRFT to be a subspace embedding with constant distortion (say, distortion $1/2$) with high probability [AMT10].

Formulas for leverage scores

There are many concrete ways to express the leverage scores of a tall $m \times n$ matrix $\mathbf{A}$. Here is an expression that emphasizes the matrix itself, without making explicit reference to its range:

$$\ell_j(\mathbf{A}) = \mathbf{A}[j,:] \left(\mathbf{A}^* \mathbf{A}\right)^\dagger \mathbf{A}[j,:]^*. \tag{6.6}$$

[1] Technically, this choice of d also gives an explicit rate at which the probability tends to zero, but we do not dwell on that here.

We can obtain other concrete expressions for the leverage scores by considering *any* matrix $\mathbf{U}$ whose columns form an orthonormal basis for $U = \text{range}(\mathbf{A})$. For example, this matrix $\mathbf{U}$ could be the $\mathbf{Q}$ from a QR decomposition or the $\mathbf{U}$ from the SVD or any other such matrix. Any such matrix suffices since $\mathbf{P} = \mathbf{U}\mathbf{U}^*$, as the orthogonal projector onto U, satisfies

$$\ell_j(\mathbf{A}) = \|\mathbf{U}[j,:]\|_2^2 = (\mathbf{U}\mathbf{U}^*)[j,j].$$

The subspace perspective is useful since it shows that leverage scores are unchanged if $\mathbf{A}$ is replaced by $\mathbf{A}\mathbf{A}^*$. More generally, if $\mathbf{A} = \mathbf{E}\mathbf{F}$ and $\mathbf{F}$ has full row-rank then the leverage scores of $\mathbf{E}$ match those of $\mathbf{A}$.

6.1.2 Subspace leverage scores

The standard leverage scores described in Section 6.1.1 are not suitable for low-rank approximation. The first problem is that it is perfectly reasonable to ask for a low-rank approximation of a matrix that is invertible but has many small singular values. In such situations both the row and column leverage scores will be uniform, and hence contain no information. The second problem is that the map from a matrix to its leverage scores is not locally continuous at $\mathbf{A}$ whenever $\mathbf{A}$ is rank-deficient. (As a general rule, it is difficult to solve linear algebra problems where the map from problem data to the solution is discontinuous.)

These shortcomings can partially be addressed with the concept of *subspace leverage scores*, which are also called *rank-k leverage scores* and *leverage scores relative to the best rank-k approximation*; see [DMM+12, §5] along with [DMM08] as an earlier conference version of the same.

Expressing the $m \times n$ matrix $\mathbf{A}$ by its compact SVD, $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*$, the *rank-k leverage scores* for its range are

$$\ell_j^k(\mathbf{A}) = \|\mathbf{U}[j,:k]\|_2^2.$$

Note that the rank-k leverage scores can be nonuniform regardless of the aspect ratio of the matrix. Indeed, so long as $k < \text{rank}(\mathbf{A})$, the rank-$k$ leverage scores of both range($\mathbf{A}$) and range($\mathbf{A}^*$) can be nonuniform. The problem of discontinuity of the map from a matrix to its rank-k subspace leverage scores can still persist. More generally, there is a problem that a matrix may admit multiple distinct "best rank-k approximations" for a given value of k. These problems are less troublesome if one assumes that the k^{th} spectral gap $\sigma_k(\mathbf{A}) - \sigma_{k+1}(\mathbf{A})$ is bounded away from zero. (This assumption is perhaps more often made than well-justified.) Alternatively, one can consider how well the computed scores approximate the leverage scores for some "nearby" rank-k space [DMM+12].

Let us turn to how subspace leverage scores are *used*. Continuing to focus on the case of row sampling, we are interested in the rank-k leverage score distribution

$$p_j^k(\mathbf{A}) = \frac{\ell_j^k(\mathbf{A})}{\sum_{i=1}^m \ell_i^k(\mathbf{A})}.$$

If $\mathbf{S}$ denotes a $d \times m$ row-sampling operator induced by $\boldsymbol{p}^k(\mathbf{A})$, then the sketch $\mathbf{Y} = \mathbf{S}\mathbf{A}$ leads naturally to the approximation $\hat{\mathbf{A}} = \mathbf{A}\mathbf{Y}^\dagger\mathbf{Y}$. Letting $\mathbf{A}_k$ denote some best-rank-k approximation of $\mathbf{A}$ in a unitarily invariant matrix norm "$\|\cdot\|$," it is possible to choose d sufficiently large so that

$$\|\mathbf{A} - \hat{\mathbf{A}}\| \lesssim \|\mathbf{A} - \mathbf{A}_k\| \tag{6.7}$$

holds with high probability. Note that if $\mathbf{Y}$ were an arbitrary matrix then it would be possible to choose $\mathbf{Y}$ so that the projection $\hat{\mathbf{A}} = \mathbf{A}\mathbf{Y}^{\dagger}\mathbf{Y}$ was equal to some best-rank-k approximation of $\mathbf{A}$. However, the restriction that the rows of $\mathbf{Y}$ are scaled rows of $\mathbf{A}$ significantly limits the projectors that could be used to define $\hat{\mathbf{A}}$. Because of this limitation, one may need $d \gg k$ to have any chance that (6.7) holds.

Remark 6.1.2. One rarely samples according to an exact rank-k leverage score distribution in practice. Rather, one uses randomized algorithms to approximate them. The key fact that enables this approximation is that leverage scores ("standard" or "subspace") are preserved if we replace $\mathbf{A}$ by $\mathbf{A}\mathbf{A}^*$. Moreover, as leverage scores quantify a notion of eigenvector localization, we should note that in many applications one has domain knowledge that eigenvalues should be localized [SCS10], and this could be used to construct approximations.

6.1.3 Ridge leverage scores

Ridge leverage scores are used to approximate matrices in the presence of explicit regularization. That is, we are given an $m \times m$ psd matrix $\mathbf{K}$ and a positive regularization parameter λ, and we approximate $\mathbf{K} + \lambda\mathbf{I}$ by $\hat{\mathbf{K}} + \lambda\mathbf{I}$ where $\mathbf{K}$ is a psd matrix of rank $n \ll m$. The low-rank structure in these approximations makes it much cheaper to apply $(\hat{\mathbf{K}} + \lambda\mathbf{I})^{-1}$ compared to $(\mathbf{K} + \lambda\mathbf{I})^{-1}$. This motivates the following question.

> What rank n is needed for $(\hat{\mathbf{K}} + \lambda\mathbf{I})^{-1}$ to approximate $(\mathbf{K} + \lambda\mathbf{I})^{-1}$ up to some fixed accuracy?

It turns out that this is determined by quantity $\operatorname{tr}(\mathbf{K}(\mathbf{K} + \lambda\mathbf{I})^{-1})$, which is called the *effective rank of* $\mathbf{K}$. Using μ_i to denote the i^{th}-largest eigenvalue of $\mathbf{K}$, we can express the effective rank as

$$\operatorname{tr}(\mathbf{K}(\mathbf{K} + \lambda\mathbf{I})^{-1}) = \sum_{i=1}^{m} \frac{\mu_i}{\mu_i + \lambda}.$$

Since we are working with psd matrices it is natural to define $\hat{\mathbf{K}}$ as a Nyström approximation of $\mathbf{K}$ with respect to some sketching operator $\mathbf{S}$ (see Section 4.2.2). Taking that as given, this leaves the question of how to choose the distribution for $\mathbf{S}$. Here it is worth considering how many numerically-low-rank psd matrices arising in applications are defined implicitly through pairwise evaluations of a *kernel function* on a given dataset. Taking $\mathbf{S}$ as a column-selection operator is especially appealing in these settings.

[AM15] introduced ridge leverage scores as a framework for data-aware column sampling in this context. Formally, the *ridge leverage scores of* $(\mathbf{K}, \lambda)$ are

$$\ell_i(\mathbf{K}; \lambda) = \left(\mathbf{K}\left(\mathbf{K} + \lambda\mathbf{I}\right)^{-1}\right)[i, i]. \tag{6.8}$$

In certain cases – particularly for estimating ridge leverage scores – it can be convenient to express these quantities in terms of a matrix $\mathbf{B}$ that satisfies $\mathbf{K} = \mathbf{B}\mathbf{B}^*$ and that has at least as many rows as columns. Specifically, by expressing $\mathbf{B}$ in terms of its compact SVD, one can show that

$$\ell_i(\mathbf{K}; \lambda) = \boldsymbol{b}_i^* \left(\mathbf{B}^*\mathbf{B} + \lambda\mathbf{I}\right)^{-1} \boldsymbol{b}_i \tag{6.9}$$

where $\boldsymbol{b}_i^*$ is the i^{th} row of $\mathbf{B}$. We note that the identity matrix appearing in (6.9) will be smaller than that from (6.8) if $\mathbf{B}$ is not square.

6.2 Approximation schemes

Computing leverage scores exactly is an expensive proposition. If $\mathbf{A}$ is a tall $m \times n$ matrix, then it takes $O(mn^2)$ time to compute the standard leverage scores exactly.[2] If one is interested in subspace leverage scores and k is small, then one can in principle use Krylov methods to approximate the dominant k singular vectors in far less than $O(mn^2)$ time. Such methods are not very reliable for producing good approximations of the truncated SVD, but they might suffice for estimating leverage scores. If we want to compute the ridge leverage scores of an $m \times m$ matrix $\mathbf{K}$ exactly, then the straightforward implementation takes $O(m^3)$ time.

These facts necessitate the development of efficient and reliable methods for *leverage score estimation*, which we discuss below. While these methods are generally too sophisticated for the RandBLAS, they may be appropriate for higher-level libraries such as RandLAPACK.

6.2.1 Standard leverage scores

Suppose the $m \times n$ matrix $\mathbf{A}$ is very tall, i.e., $m \gg n$. Here we summarize a method by Drineas et al. that can compute approximate leverage scores, to within a constant multiplicative error factor, in $O(mn \log m)$ time, i.e., in roughly the time it takes to implement a random projection, with some constant failure probability bounded away from one [DMM+12]. This can offer improved efficiency over straightforward $O(mn^2)$ approaches when $m \gg n$ and yet $m \in o(2^n)$.

We set the stage for this method by expressing leverage scores as follows

$$\ell_j(\mathbf{A}) = \|\boldsymbol{\delta}_j^* \mathbf{U}\|_2^2 = \|\boldsymbol{\delta}_j^* \mathbf{U}\mathbf{U}^*\|_2^2 = \|\boldsymbol{\delta}_j^* \mathbf{A}\mathbf{A}^\dagger\|_2^2 \tag{6.10}$$

where we note that the second equality in the above display follows from unitary invariance of the spectral norm. The method proceeds by approximating two operations in the right-most expression in (6.10). First we approximate the pseudoinverse of $\mathbf{A}$ and then we approximate the matrix-matrix product $\mathbf{A}\mathbf{A}^\dagger$. It is important to note that using approximations in both steps is essential for asymptotic complexity improvements, since traditional methods would take $O(mn^2)$ for the first step and $O(m^2 n)$ time for the second step. (In extreme situations, depending on the hardware that would be used, it may be worth performing the matrix-matrix product of the second step explicitly.)

The pseudoinverse computation is approximated by applying a wide $d_1 \times m$ SRFT $\mathbf{S}_1$ to the left of $\mathbf{A}$. Letting $\mathbf{U}_1 \mathbf{\Sigma}_1 \mathbf{V}_1^*$ be an SVD of this $d_1 \times n$ sketched matrix $\mathbf{S}_1\mathbf{A}$, we approximate

$$\begin{aligned} \ell_j(\mathbf{A}) \approx \hat{\ell}_j(\mathbf{A}) &= \|\boldsymbol{\delta}_j^* \mathbf{A}(\mathbf{S}_1\mathbf{A})^\dagger\|_2^2 \\ &= \|\boldsymbol{\delta}_j^* \mathbf{A}\mathbf{V}_1 \mathbf{\Sigma}_1^{-1} \mathbf{U}_1^*\|_2^2 \\ &= \|\boldsymbol{\delta}_j^* \mathbf{A}\mathbf{V}_1 \mathbf{\Sigma}_1^{-1}\|_2^2 \end{aligned}$$

at a cost of $O(d_1 n^2)$. However, we are not out of the woods yet, since multiplying $\mathbf{A}$ with $\mathbf{V}_1 \mathbf{\Sigma}_1^{-1}$ would still cost $O(mn^2)$. This is addressed by applying a tall sketching

[2] The preferred way to do this would be to take the row norms of the factor $\mathbf{Q}$ from a thin QR decomposition of $\mathbf{A}$.

operator $\mathbf{S}_2$ of size $n \times d_2$ to the right of $\mathbf{V}_1\mathbf{\Sigma}_1^{-1}$ before multiplying it by $\mathbf{A}$. That is, we further approximate

$$\hat{\ell}_j(\mathbf{A}) \approx \hat{\hat{\ell}}_j(\mathbf{A}) = \|\boldsymbol{\delta}_j^* \mathbf{A}(\mathbf{V}_1\mathbf{\Sigma}_1^{-1}\mathbf{S}_2)\|_2^2. \tag{6.11}$$

This reduces the cost of the matrix multiplication to $O(mnd_2)$ and hence the cost of the overall procedure to $O(d_1n^2 + d_2mn)$. [DMM+12] gives details on how large d_1 and d_2 must be to ensure useful accuracy guarantees for the approximate leverage scores; see also [MMY15, §5.2] for a related evaluation.

This estimation method can be adapted to efficiently compute "cross-leverage scores," as well as subspace leverage scores; see [DMM+12] for details. It also has natural adjustments to make it faster. For example, [CW17] suggest replacing the SRFT $\mathbf{S}_1$ by $\tilde{\mathbf{S}}_1 = \mathbf{FC}$ where $\mathbf{C}$ is a CountSketch and $\mathbf{F}$ is an SRFT that further compresses the output of $\mathbf{C}$; [NN13] propose replacing $\mathbf{S}_1$ by a SASO (recall from Section 2.4.1 that a SASO is generalized CountSketch), which yields a similar speed-up as that achieved in [CW17].

6.2.2 Subspace leverage scores

There is a wide range of possibilities for estimating subspace leverage scores. We describe two such methods here (slightly adapted) from [DMM+12]. Let us say that we want to estimate the rank-k leverage scores of $\mathbf{A}$ for some $k \ll \min\{m, n\}$. Both of the algorithms below work by finding the *exact* leverage scores of an implicit rank-k matrix $\hat{\mathbf{A}}$, for which a distance $\|\hat{\mathbf{A}} - \mathbf{A}\|$ is near-optimal among all rank-k approximations.

An adaptation of [DMM+12, Algorithm 5]

The original goal of this algorithm was to return the leverage scores of a rank-k approximation of $\mathbf{A}$ that was near-optimal in Frobenius norm. Framing things more abstractly, the approach requires that the user specify an oversampling parameter $s \in O(k)$. Its first step is to compute a rank-$(k+s)$ QB decomposition of $\mathbf{A}$ (e.g., by some method from Section 4.3.2) $\mathbf{A} \approx \mathbf{QB}$. Next, it computes the top k left singular vectors of $\mathbf{B}$ by some traditional method. Letting $\mathbf{U}_k$ denote the $(k+s) \times k$ matrix of such leading left singular vectors, the algorithm takes the columns of $\mathbf{QU}_k$ to define approximations of the leading k left singular vectors of $\mathbf{A}$. The row-norms of this matrix define the approximate rank-k leverage scores.

In context, [DMM+12, Algorithm 5] used an elementary QB decomposition with $\mathbf{Q} = \texttt{orth}(\mathbf{AS})$ for an $n \times (k+s)$ Gaussian operator $\mathbf{S}$. The analysis of this algorithm presumed that $s \geq \lceil k/\epsilon + 1 \rceil$ for some tolerance parameter ϵ. The meaning of ϵ was as follows: when viewed as random variables, the returned leverage scores coincide with those of a rank-k approximation $\hat{\mathbf{A}}$ where

$$\mathbb{E}\|\hat{\mathbf{A}} - \mathbf{A}\|_{\mathrm{F}}^2 \leq (1+\epsilon) \sum_{j>k} \sigma_j(\mathbf{A})^2.$$

Looking back at this error bound from our present perspective, it is clear that a huge variety of similar bounds can be obtained by using different methods for the QB decomposition. One possibility on this front would be to use adaptive QB algorithms that approximate $\mathbf{A}$ to some prescribed accuracy. Subspace leverage scores obtained in this way may be well-suited for approximating $\mathbf{A}$ by a low-rank submatrix-oriented decomposition up to prescribed accuracy.

A description of [DMM+12, Algorithm 4]

This is a two-stage method to find the leverage scores of a rank-k approximation to $\mathbf{A}$ that is near-optimal in spectral norm.

To understand the first stage, recall that some of the simplest QB algorithms make use of power iteration as described in Section 4.3.1. That is, rather than setting $\mathbf{Q} = \texttt{orth}(\mathbf{AS})$ for Gaussian $\mathbf{S}$, they set $\mathbf{S} = (\mathbf{A}^*\mathbf{A})^q\mathbf{S}_0$ for Gaussian $\mathbf{S}_0$. Practical implementations of QB based on power iteration introduce stabilization between successive applications of $\mathbf{A}$ and $\mathbf{A}^*$. Such stabilization preserves the range of $\mathbf{AS}$, but it may change its singular vectors. If such stabilization is *not* used, then the left singular vectors of $\mathbf{A}(\mathbf{A}^*\mathbf{A})^q\mathbf{S}_0$ for Gaussian $\mathbf{S}_0$ would be reasonable approximations to the leading left singular vectors of $\mathbf{A}$ (modulo numerical problems that are sure to arise for moderate q).

The observation above is the basis for [DMM+12, Algorithm 4]. In context, its first stage is to compute $\mathbf{S}_{q+1} = (\mathbf{AA}^*)^q\mathbf{AS}_0$ from an $n \times 2k$ Gaussian operator $\mathbf{S}_0$. In a second stage, approximate leverage scores of $\mathbf{S}_{q+1}$ – call them $\hat{\ell}_i$ – are obtained from any method that ensures

$$|\hat{\ell}_i - \ell_i(\mathbf{S}_{q+1})| \leq \epsilon\ \ell_i(\mathbf{S}_{q+1}).$$

These approximations are the estimates for the rank-k leverage scores of $\mathbf{A}$.

[DMM+12, Lemma 15 and Theorem 16] prescribe a value for q (as a function of m, n, k, and ϵ) that ensures an approximation guarantee for the leverage score estimates given above. Specifically, for the prescribed q, the estimated leverage scores are within a factor $\frac{1-\epsilon}{2(1+\epsilon)}$ of the leverage scores of a rank-k matrix $\hat{\mathbf{A}}$ that satisfies

$$\mathbb{E}\|\hat{\mathbf{A}} - \mathbf{A}\|_2 \leq (1 + \epsilon/10)\sigma_{k+1}(\mathbf{A}).$$

As before, the randomness in this expectation is over the randomness used to estimate the leverage scores.

6.2.3 Ridge leverage scores

A wide variety of algorithms have been devised to estimate ridge leverage scores or carry out approximate ridge leverage score sampling. The simplest such algorithm, proposed in [AM15] alongside the definition of ridge leverage scores, proceeds as follows:

- Start with a distribution $\boldsymbol{p} = (p_i)_{i \in [\![m]\!]}$ over the column index set of $\mathbf{K}$.
- Construct a column selection operator $\mathbf{S}$ with n columns, where each column is independently set to $\boldsymbol{\delta}_i \in \mathbb{R}^m$ with probability p_i.
- Compute the Nyström approximation of $\mathbf{K}$ with respect to $\mathbf{S}$. Suppose the approximation is represented as $\hat{\mathbf{K}} = \mathbf{BB}^*$ for an $m \times n$ matrix $\mathbf{B}$.
- Using $\boldsymbol{b}_i \in \mathbb{R}^n$ for the i^{th} row of $\mathbf{B}$, take $\tilde{\ell}_i := \boldsymbol{b}_i^*(\mathbf{B}^*\mathbf{B} + \lambda\mathbf{I})^{-1}\boldsymbol{b}_i$ as an approximation for the i^{th} ridge leverage score of $\mathbf{K}$ with regularization λ.

One can of course start with $\boldsymbol{p} = (1/m)_{i \in [\![m]\!]}$ as the uniform distribution over columns of $\mathbf{K}$. An alternative starting point is the distribution $\boldsymbol{p} = \operatorname{diag}(\mathbf{K})/\operatorname{tr}(\mathbf{K})$. While the latter distribution can lead to useful theoretical guarantees (see [AM15, Theorem 4]) it is not suitable for computing very accurate approximations.

Iterative methods should be used if accurate approximations to ridge leverage scores are desired. Notably, most of the iterative methods in the literature *simultaneously* estimate the ridge leverage scores and sample columns from $\mathbf{K}$ according to the estimates [MM17; CLV17; RCC+18]. This algorithmic structure blurs the distinction between approximating ridge leverage scores and producing a Nyström approximation of $\mathbf{K}$ via column selection. This precise nature of the blurring can also vary substantially from one algorithm to another. For example, [MM17, Algorithms 2 and 3] are very different from [CLV17, Algorithm 1], which in turn is materially different from [RCC+18, Algorithms 1 and 2].

The abundance and sophistication of these methods make it impractical for us to summarize them here. We instead settle for stating their general qualitative conclusions. Letting $d = \operatorname{tr}(\mathbf{K}(\mathbf{K} + \lambda\mathbf{I})^{-1})$ denote the effective rank of $\mathbf{K}$, one can construct an approximation $\hat{\mathbf{K}}$ of rank $n \in O(d \log d)$ for which $\|\mathbf{K} - \hat{\mathbf{K}}\|_2 \leq \lambda$ holds with high probability. Furthermore, these approximations can be constructed in time $O(mn^2)$ using only n column samples from $\mathbf{K}$. We refer the reader to the cited works above for details on specific algorithms.

6.3 Special topics and further reading

Here, we mention a handful of generalizations and variants of leverage score sampling that, while not part of our immediate plans, may be of longer-term interest. The interested reader should consult the source material for details of what we describe below. In addition to those source materials, the interested reader is referred to Sobczyk and Gallopoulos' paper [SG21], which is accompanied by a carefully developed C++ and Python library called `pylspack` [SG22]. We also recommend Larsen and Kolda's recent work [LK20, §4 and Appendix A] – which includes practical advice on leverage score sampling and theoretical results with explicit constant factors.

6.3.1 Leverage score sparsified embeddings

Our concept of long-axis-sparse operators from Section 2.4.2 is based on the *Leverage Score Sparsified* or *LESS* embeddings of Dereziński et al. [DLD+21]. Here we explain the role of leverage scores when using these sketching operators.

Let $\mathbf{S}$ be a a random $d \times m$ long-axis-sparse operator ($d \ll m$) with sparsity parameter k and sampling distribution $\boldsymbol{p} = (p_1, \ldots, p_m)$. The idea of LESS embeddings is that varying k should provide a way to interpolate between the low cost of sketching by row sampling and the high cost of sketching by Gaussian operators, while still obtaining a sketch that is meaningfully Gaussian-like. Indeed, if $k \approx n = \operatorname{rank}(\mathbf{A})$, then [Der22a] showed that, with high probability, the resulting sketching operator is nearly indistinguishable from a dense sub-gaussian operator (such as Gaussian or Rademacher), despite the reduction from $O(dmn)$ time to $O(dn^2)$. This performance comparison was demonstrated for several estimation tasks involving the inverse covariance matrix $\mathbf{A}^*\mathbf{A}$ [DLD+21], as well for the Newton Sketch optimization method [DLP+21].

As with other uses of leverage scores, approximate leverage scores suffice for LESS embeddings; and the computational cost of a LESS embedding is typically dominated by the cost of estimating the leverage scores of $\mathbf{A}$. The use of leverage scores in the sparsification pattern is essential for theoretically showing that a LESS

embedding exhibits nearly identical performance to a Gaussian operator for all matrices $\mathbf{A}$. Good empirical performance observed in practice, to a varying degree, also when $\boldsymbol{p}$ is the uniform distribution and $k \ll \operatorname{rank}(\mathbf{A})$ [DLP+21, §5].

6.3.2 Determinantal point processes

In many data analysis applications, submatrix-oriented decompositions such as Nyström approximation via column selection are desirable for their interpretability. In this context, we may wish to produce a very small but high-quality sketch of the matrix $\mathbf{A}$, using a method more refined (albeit slower) than leverage score sampling. Here we discuss Determinantal Point Processes (DPPs; [DM21a]) as one of many such methods from the literature.

Let $\mathbf{A}$ be an $m \times m$ psd matrix. A *Determinantal Point Process* is a distribution over index subsets $J \subseteq [\![m]\!]$ such that:

$$\mathbb{P}(J = S) = \frac{\det(\mathbf{A}[S, S])}{\det(\mathbf{A} + \mathbf{I})}.$$

The above DPP formulation is known as an L-ensemble, and it is also sometimes called volume sampling [DRV+06; DM10]. Unlike leverage score sampling, individual indices sampled in a DPP are not drawn independently, but rather jointly, to minimize redundancies in the sampling process. In fact, a DPP can be viewed as an extension of leverage score sampling that incorporates dependencies between the samples, inducing diversity in the selected subset [KT12].

DPP sampling can be used to construct improved Nyström approximations $\hat{\mathbf{A}} = (\mathbf{AS})(\mathbf{S}^*\mathbf{AS})^\dagger(\mathbf{AS})^*$ where the selection matrix $\mathbf{S}$ corresponds to the random subset J. In particular, [DRV+06; GS12; DKM20] established strong guarantees for this approach in terms of the nuclear norm error relative to the best rank k approximation: $\|\hat{\mathbf{A}} - \mathbf{A}\|_* \leq (1 + \epsilon)\|\mathbf{A}_k - \mathbf{A}\|_*$, where k is the target rank and the subset size $|J|$ is chosen to be equal or slightly larger than k. DPPs have also found applications in machine learning [KT12; DKM20; DM21a] as a method for constructing diverse and interpretable data representations.

It is challenging to implement efficient methods for sampling from a DPP, and this is an area of ongoing work. One promising method has recently been proposed by Poulson [Pou20]. Two other classes of methods can be obtained by exploiting the connection between DPPs and ridge leverage scores.

1. One can use intermediate sampling with ridge leverage scores to produce a larger index set T, which is then trimmed down to produce a smaller DPP sample $J \subseteq T$ [Der19; DCV19; CDV20].

2. One can use iterative refinement on a Markov chain, where we start with an initial subset J_1, and then we gradually update it by swapping out one index at a time, producing a sequence of subsets $J_1, J_2, J_3, ...$, which rapidly converges to a DPP distribution [AGR16; AD20; ADV+22].

The computational cost of these procedures is usually dominated by the cost of estimating the ridge leverage scores (recall that there are methods for doing this that do not need to access all of $\mathbf{A}$). However, these procedures carry additional overhead since some of the ridge leverage score samples must be discarded to produce the final DPP sample.

6.3.3 Further variations on leverage scores

In the case of tall data matrices $\mathbf{A}$, leverage scores are useful for finding data-aware sketching operators $\mathbf{S}$ so that the Euclidean norms of vectors in the range of $\mathbf{SA}$ are comparable to the Euclidean norms of vectors in the range of $\mathbf{A}$. A related concept called *Lewis weights* can be used for matrix approximation where we want $\mathbf{S}$ to approximately preserve the p-norm of vectors in the range of $\mathbf{A}$ for some $p \neq 2$ [CP15]. These are improved versions of leverage-like scores used by Dasgupta et al. for ℓ_p regression [DDH+09]. A more recently proposed concept samples according to the probabilities

$$p_i = \frac{\left\| (\mathbf{A}^*\mathbf{A})^{-1} \mathbf{A}[i,:] \right\|_2}{\sum_{j=1}^{m} \left\| (\mathbf{A}^*\mathbf{A})^{-1} \mathbf{A}[j,:] \right\|_2}$$

in order to estimate the *variability* of sketch-and-solve solutions to overdetermined least squares problems [MZX+22]; see also [MMY15] for related importance sampling probabilities that come with useful statistical properties. Similar probabilities (where all norms in the above expression were squared) were studied in [DCM+19].

Section 7

Advanced Sketching: Tensor Product Structures

This section considers efficient sketching of data with tensor product structure. We specifically focus on implicit matrices with Kronecker and Khatri–Rao product structure. These structures are of interest in RandNLA due to their prominent role in certain randomized algorithms for tensor decomposition. A secondary point of interest is that the operators discussed in this section can also be used for sketching unstructured matrices. They may, for example, be used as alternatives to unstructured test vectors in norm and trace estimation [BK21]. In this case, the main benefit would not be improved speed but reduced storage requirements for storing the sketching operator.

In Section 7.1 we define the Kronecker and Khatri–Rao matrix products. Section 7.2 presents four families of sketching operators that can be applied efficiently to matrices that are stored implicitly with these product structures. Section 7.3 discusses implementation considerations for the structured sketching operators in this section, with a focus on how they can be used in tensor decomposition algorithms.

A note on scope

We should emphasize that algorithms for general tensor computations are out-of-scope for RandLAPACK. The functionality described here would only be made available as utility functions (i.e., computational routines) for *facilitating* certain tensor computations. This is part of a broader idea that RandLAPACK should facilitate advanced sketching operations of interest in RandNLA that are outside the scope of the RandBLAS.

7.1 The Kronecker and Khatri–Rao products

Suppose that $\mathbf{B}$ is an $m \times n$ matrix and $\mathbf{C}$ is a $p \times q$ matrix. The Kronecker product of $\mathbf{B}$ and $\mathbf{C}$ is the $mp \times nq$ matrix

$$\mathbf{B} \otimes \mathbf{C} = \begin{bmatrix} \mathbf{B}[1,1] \cdot \mathbf{C} & \mathbf{B}[1,2] \cdot \mathbf{C} & \cdots & \mathbf{B}[1,n] \cdot \mathbf{C} \\ \mathbf{B}[2,1] \cdot \mathbf{C} & \mathbf{B}[2,2] \cdot \mathbf{C} & \cdots & \mathbf{B}[2,n] \cdot \mathbf{C} \\ \vdots & \vdots & & \vdots \\ \mathbf{B}[m,1] \cdot \mathbf{C} & \mathbf{B}[m,2] \cdot \mathbf{C} & \cdots & \mathbf{B}[m,n] \cdot \mathbf{C} \end{bmatrix}.$$

If $\mathbf{B}$ and $\mathbf{C}$ have the same number of columns (i.e., if $n = q$), then their Khatri–Rao product is the $mp \times n$ matrix

$$\mathbf{B} \odot \mathbf{C} = \begin{bmatrix} \mathbf{B}[:,1] \otimes \mathbf{C}[:,1] & \mathbf{B}[:,2] \otimes \mathbf{C}[:,2] & \cdots & \mathbf{B}[:,n] \otimes \mathbf{C}[:,n] \end{bmatrix}.$$

The Khatri–Rao product is sometimes also referred to as the *matching columnwise Kronecker product* for transparent reasons. The Kronecker and Khatri–Rao products for more than two matrices are defined in the obvious way. Note that for two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ we have that

$$\boldsymbol{x} \otimes \boldsymbol{y} = \boldsymbol{x} \odot \boldsymbol{y} = \mathrm{vec}(\boldsymbol{x} \circ \boldsymbol{y})$$

where $\circ$ denotes the outer product and $\mathrm{vec}(\cdot)$ is an operator that turns a matrix into a vector by vertically concatenating its columns. We also use $\circledast$ to denote the elementwise (Hadamard) product.

Matrices with Kronecker and Khatri–Rao product structure tend to be *very* large. For example, consider matrices $\mathbf{B}_1, \ldots, \mathbf{B}_L$, all of size $m \times n$. Their Kronecker product $\mathbf{B}_1 \otimes \cdots \otimes \mathbf{B}_L$ is an $m^L \times n^L$ matrix and their Khatri–Rao product $\mathbf{B}_1 \odot \cdots \odot \mathbf{B}_L$ is an $m^L \times n$ matrix. The exponential dependence on L means that these products can become very large even if the matrices $\mathbf{B}_1, \ldots, \mathbf{B}_L$ are not especially large. Even just forming and storing these products may therefore be prohibitively expensive.

Kronecker and Khatri–Rao product matrices feature prominently in algorithms for tensor decomposition (i.e., decomposition of multidimensional arrays into sums and products of more elementary objects, see Section 7.3). They also appear in a variety of other contexts when sketching techniques are helpful, such as for representation of polynomial kernels [PP13; ANW14; WZ20; WZ22], when fitting polynomial chaos expansion models in surrogate modeling [TNX15; SNM17; CMX+22], multi-dimensional spline fitting [DSS+18], and in PDE inverse problems [CLN+20].

7.2 Sketching operators

Section 7.2.1 introduces sketching operators that are distinguished by having rows with particular structures. Section 7.2.2 discusses a variant of the SRFT with an additional tensor-produce structure. Section 7.2.3 discusses TensorSketch operators, which are analogous to CountSketch operators from Section 2.4.1. In Section 7.2.4 we describe sketching operators that are recursive and have multi-stage structure. These incorporate some of the sketching operators discussed in the previous subsections as stepping stones. Section 7.2.5 covers row sampling methods for tall matrices with tensor product structure.

We note that the sketching operators in Sections 7.2.1–7.2.4 are all oblivious, whereas the sampling-based methods in Section 7.2.5 are not. We also note that all of the oblivious sketching operators we discuss could be applied to *unstructured* matrices. This would yield no speed benefit compared to using their unstructured counterparts, but it would reduce the storage requirement compared to traditional dense sketching operators of the kind supported by the RandBLAS.

7.2.1 Row-structured tensor sketching operators

Here we describe three types of sketching operators whose rows can be applied to Kronecker and Khatri–Rao product matrices very efficiently. The second of these methods requires notions of tensor representations such as the *CP format*, which we will revisit in Section 7.3.

Khatri–Rao products of elementary sketching operators

The most basic row-structured sketching operator takes the form

$$\mathbf{S} = (\mathbf{S}_1 \odot \mathbf{S}_2 \odot \cdots \odot \mathbf{S}_L)^*, \tag{7.1}$$

where each $\mathbf{S}_k$ is an appropriate random matrix of size $m_k \times d$ for $k \in [\![L]\!]$. Such an operator maps $(m_1 \cdots m_L)$-vectors to d-vectors. It can be efficiently applied to Kronecker product vectors, which in turn means that it can be applied efficiently (column-wise) to both Kronecker and Khatri–Rao product matrices. Consider vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_L$ where $\boldsymbol{x}_k$ is a length-m_k vector. The operator in (7.1) is then applied to a vector $\boldsymbol{v} = \boldsymbol{x}_1 \otimes \cdots \otimes \boldsymbol{x}_L$ via the formula

$$\mathbf{S}\boldsymbol{v} = (\mathbf{S}_1^* \boldsymbol{x}_1) \circledast (\mathbf{S}_2^* \boldsymbol{x}_2) \circledast \cdots \circledast (\mathbf{S}_L^* \boldsymbol{x}_L).$$

To the best of our knowledge, [BBB15] were the first to use random matrices of the form (7.1) to accelerate tensor computations in the spirit of RandNLA.[1] They suggest drawing the entries of each $\mathbf{S}_k$ independently from a distribution with mean zero and unit variance, but they do not provide any theoretical guarantees for the performance of such sketching operators. Sun et al. [SGT+18] independently propose using operators of the form (7.1) where the submatrices $\mathbf{S}_k$ are chosen to be either Gaussian or sparse operators. They also propose a variance-reduced modification which is an appropriate rescaling of the sum of several maps of the form (7.1). They provide theoretical guarantees for sketching operators in (7.1) with $L = 2$ (and its variance-reduced modification) when $\mathbf{S}_1$ and $\mathbf{S}_2$ have entries that are drawn independently from an appropriately scaled mean-zero sub-Gaussian distribution, leaving analysis for the case when $L > 2$ open for future work.

[1] Similar ideas were used earlier for applications in differential privacy; see [KRS+10; Rud12].

Row-wise vectorized tensors

Rakhshan and Rabusseau [RR20] propose a distribution of sketching operators for which the i^{th} row is given by $\mathbf{S}[i,:] = \text{vec}(\mathcal{X}_i)^*$, where $\mathcal{X}_i$ is a tensor in some factorized format and vec is a function that returns a vectorized version of its input as a column vector ($\text{vec}(\mathcal{X}) = \mathcal{X}(:)$ in Matlab notation). More specifically, they consider two cases: In the first case, $\mathcal{X}_i$ is in *CP format* and defined elementwise via

$$\mathcal{X}_i[j_1, j_2, \ldots, j_L] = \sum_{r=1}^{R} \boldsymbol{a}_r^{(i,1)}[j_1] \cdot \boldsymbol{a}_r^{(i,2)}[j_2] \cdots \boldsymbol{a}_r^{(i,L)}[j_L] \tag{7.2}$$

where the vector entries $\boldsymbol{a}_r^{(i,n)}[j_n]$ are drawn independently from an appropriately scaled Gaussian distribution. In the second case, $\mathcal{X}_i$ is in so-called *tensor train format* and defined elementwise via

$$\mathcal{X}_i[j_1, j_2, \ldots, j_L] = \mathbf{A}_{j_1}^{(i,1)} \mathbf{A}_{j_2}^{(i,2)} \cdots \mathbf{A}_{j_L}^{(i,L)}, \tag{7.3}$$

where L is the number of tensor modes, and each matrix $\mathbf{A}_{j_n}^{(i,n)}$ is of size $R_n \times R_{n+1}$ where $R_1 = R_{L+1} = 1$ to ensure that the product is a scalar. The entries of $\mathbf{A}_{j_n}^{(i,n)}$ are drawn independently from an appropriately scaled Gaussian distribution.

For both of the constructs described above, the inner product of $\text{vec}(\mathcal{X}_i)$ and Kronecker product vectors can be computed efficiently due to the special structure of the CP and tensor train formats. This makes efficient application of the operator to Kronecker and Khatri–Rao product matrices possible. Theoretical guarantees are provided for these vectorized tensor sketching operators in [RR20]. The follow-up work [RR21] shows that the results for the tensor train-based sketching operators also extend to the case when the cores are drawn from a Rademacher distribution.

Two-stage operators

Iwen et al. [INR+20] propose a two-stage sketching procedure for mapping $(m_1 \cdots m_L)$-vectors to d-vectors. The first step consists of applying a row-structured matrix $(\mathbf{S}_1 \otimes \cdots \otimes \mathbf{S}_L)$, where each $\mathbf{S}_k$ is a sketching operator of size $p_k \times m_k$. This maps the $(m_1 \cdots m_L)$-vector to an intermediate embedding space of dimension $(p_1 \cdots p_L)$. This is then followed by another sketching operator $\mathbf{T}$ of size $d \times (p_1 \cdots p_L)$ which maps the intermediate representation to the final d-dimensional space.

7.2.2 The Kronecker SRFT

Kronecker SRFTs are a variant of the SRFTs discussed in Section 2.5. They can be applied very efficiently to a Kronecker product vector without forming the vector explicitly. They were first proposed by [BBK18] for efficient sketching of the Khatri–Rao product matrices that arise in tensor CP decomposition. Theoretical analysis of these sketching operators can be found in [JKW20; MB20; BKW21].

The Kronecker SRFT that maps $(m_1 \cdots m_L)$-vectors to d-vectors takes the form

$$\mathbf{S} = \sqrt{\frac{m_1 \cdots m_L}{d}} \mathbf{R} \Big(\bigotimes_{k=1}^{L} \mathbf{F}_k \Big) \Big(\bigotimes_{k=1}^{L} \mathbf{D}_k \Big), \tag{7.4}$$

where each $\mathbf{D}_k$ is a diagonal $m_k \times m_k$ matrix of independent Rademachers, each $\mathbf{F}_k$ is an $m_k \times m_k$ fast trigonometric transform, and $\mathbf{R}$ randomly samples d components

from an $(m_1 \cdots m_L)$-vector. The Kronecker SRFT replaces the $\mathbf{F}$ and $\mathbf{D}$ operators in the standard SRFT by Kronecker products of smaller operators of the same form. With $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_L$ defined as in Section 7.2.1, the operator in (7.4) can be applied efficiently to $\boldsymbol{x}_1 \otimes \cdots \otimes \boldsymbol{x}_L$ via the formula

$$\sqrt{\frac{m_1 \cdots m_L}{d}} \mathbf{R} \Big(\bigotimes_{k=1}^{L} \mathbf{F}_k \Big) \Big(\bigotimes_{k=1}^{L} \mathbf{D}_k \Big) \Big(\bigotimes_{k=1}^{L} \boldsymbol{x}_k \Big) = \sqrt{\frac{m_1 \cdots m_L}{d}} \mathbf{R} \Big(\bigotimes_{k=1}^{L} \mathbf{F}_k \mathbf{D}_k \boldsymbol{x}_k \Big).$$

The formula shows that only those entries in $\bigotimes_k \mathbf{F}_k \mathbf{D}_k \boldsymbol{x}_k$ that are sampled by $\mathbf{R}$ need to be computed. From this, we can back out the indices of each vector $\mathbf{F}_k \mathbf{D}_k \boldsymbol{x}_k$ that need to be computed. Given these indices one could compute the relevant entries of these vectors using subsampled FFT methods of the kind alluded to in Section 2.5. We note that this formula is straightforwardly extended to Kronecker and Khatri–Rao product matrices.

7.2.3 TensorSketch

A TensorSketch operator is a kind of structured CountSketch that can be applied very efficiently to Kronecker product matrices.[2] The improved computational efficiency of TensorSketch comes at the cost of needing a larger embedding dimension than CountSketch. TensorSketch was first proposed in [Pag13] for fast approximate matrix multiplication. It was further developed in [PP13; ANW14; DSS+18] where it is used for low-rank approximation, regression, and other tasks.

Let $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_L$ be defined as in Section 7.2.1. A TensorSketch, which we denote by $\mathbf{S}$ below, maps an $(m_1 \cdots m_L)$-vector $\boldsymbol{v} = \boldsymbol{x}_1 \otimes \cdots \otimes \boldsymbol{x}_L$ to a d-vector via the formula

$$\mathbf{S}\boldsymbol{v} = \mathrm{DFT}^{-1} \Big(\bigoplus_{k=1}^{L} \mathrm{DFT}(\mathbf{S}_k \boldsymbol{x}_k) \Big), \tag{7.5}$$

where each $\mathbf{S}_k$ is an independent CountSketch that maps m_k-vectors to d-vectors. Here, DFT denotes the discrete Fourier transform which can be efficiently applied using fast Fourier transform (FFT) methods. TensorSketches use the fact that polynomials can be multiplied using the DFT, which is why DFT and its inverse appear in the formula above; see [Pag13] for details.

Remark 7.2.1. We have not investigated whether fast trig transforms other than the discrete Fourier transform (e.g., the discrete cosine transform) can be used for this type of sketching operator.

7.2.4 Recursive sketching

In order to achieve theoretical guarantees, the sketching operators discussed so far require an embedding dimension d which scales *exponentially* with L when embedding a vector of the form $\boldsymbol{x}_1 \otimes \cdots \otimes \boldsymbol{x}_L$. Ahle et al. [AKK+20] propose sketching operators that are computed recursively and have the remarkable property that their requisite embedding dimensions scale *polynomially* with L. Since [AKK+20] are concerned with oblivious subspace embedding of polynomial kernels, they consider the case when all $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_L$ are of the same length. However, their results

[2] Recall that a CountSketch is a SASO in the sense of Section 2.4.1. Each short-axis vector in a CountSketch has a single nonzero entry, sampled from the Rademacher distribution.

should extend to the general case when the vectors are of different lengths (for example, see [Mal22, Corollary 18]).

Suppose $\boldsymbol{x}_1, \dots, \boldsymbol{x}_L$ are m-vectors and that $L = 2^q$ for a positive integer q. The recursive sketching operator first computes

$$\boldsymbol{y}_k^{(0)} = \mathbf{T}_k \boldsymbol{x}_k \quad \text{for} \quad k \in [\![L]\!]$$

where $\mathbf{T}_1, \dots, \mathbf{T}_L$ are independent SASOs (e.g., CountSketches, see Section 2.4.1) that map m-vectors to d-vectors. The d-vectors $\boldsymbol{y}_1^{(0)}, \dots, \boldsymbol{y}_L^{(0)}$ are now combined pairwise into $L/2 = 2^{q-1}$ vectors. This is done by computing

$$\boldsymbol{y}_k^{(1)} = \mathbf{S}_k(\boldsymbol{y}_{2k-1}^{(0)} \otimes \boldsymbol{y}_{2k}^{(0)}) \quad \text{for} \quad k \in [\![L/2]\!]$$

where $\mathbf{S}_1, \dots, \mathbf{S}_{L/2}$ are independent sketching operators that map d^2-vectors to d-vectors. If the initial $\mathbf{T}_1, \dots, \mathbf{T}_L$ are CountSketches then the $\mathbf{S}_i$ are canonically TensorSketches. If instead $\mathbf{T}_1, \dots, \mathbf{T}_L$ are more general SASOs then the $\mathbf{S}_i$ are canonically Kronecker SRFTs. Regardless of which configuration we use, the pairwise combination of vectors is repeated for a total of $q = \log_2(L)$ steps until a single d-vector remains, which is the embedding of $\boldsymbol{x}_1 \otimes \cdots \otimes \boldsymbol{x}_L$. The case when L is not a power of two is handled by adding additional vectors $\boldsymbol{x}_k = \boldsymbol{e}_1$ for $k = L+1, \dots, 2^{\lceil \log_2(L) \rceil}$ where $\boldsymbol{e}_1$ is the first standard basis vector in $\mathbb{R}^m$. Recursive sketching operators are linear despite their somewhat complicated description.

Song et al. [SWY+21] develop a similar recursive sketching operator which takes inspiration from the one discussed above and applies it to the sketching of polynomial kernels. For the degree-L polynomial kernel, this involves sketching of matrices of the form $\mathbf{A}^{\odot L} = \mathbf{A} \odot \cdots \odot \mathbf{A}$, where the matrix $\mathbf{A}$ appears L times in the right-hand side.

The recursive sketching operator by [AKK+20] can be described by a binary tree, with each node corresponding to an appropriate sketching operator. Ma and Solomonik [MS22] generalize this idea by allowing for other graph structures, but limit nodes in these graphs to be associated with Gaussian sketching operators. Under this framework, they develop a structured sketching operator whose embedding dimension only scales *linearly* with L. These operators can be adapted for efficient application to vectors with general tensor network structure which includes Kronecker products of vectors as a special case.

7.2.5 Leverage score sampling for implicit matrices with tensor product structures

Consider the problem of sketching and solving a least squares problem

$$\min_{\mathbf{x}} \|\mathbf{A}\mathbf{X} - \mathbf{Y}\|_{\mathrm{F}} \tag{7.6}$$

when the columns of $\mathbf{A}$ have tensor product structure and $\mathbf{Y}$ is a thin unstructured matrix. The sketching operators discussed so far in this section can be efficiently applied to $\mathbf{A}$. However, since $\mathbf{Y}$ lacks structure, these sketching operators require accessing all nonzero elements of $\mathbf{Y}$. This can be prohibitively expensive in applications such as the following.

- In iterative methods for tensor decomposition, one typically solves a sequence of least squares problems for which $\mathbf{A}$ is structured and $\mathbf{Y}$ contains all the

entries of the tensor being decomposed [KB09]. When $\mathbf{Y}$ has a fixed proportion of nonzero entries, the cost will therefore scale exponentially with the number of tensor indices—a manifestation of the *curse-of-dimensionality*.

- When fitting polynomial chaos expansion functions in surrogate modeling [TNX15; SNM17; CMX+22], $\mathbf{A}$ contains evaluations of a multivariate polynomial on a structured quadrature grid and $\mathbf{Y}$ (which will now be a column vector) contains the outputs of an expensive data generation process (e.g., an experiment or high-fidelity PDE simulation).

In both example applications, it is clearly desirable to avoid using all entries of $\mathbf{Y}$ when solving (7.6). As discussed in Section 6, leverage score sampling can be used to sketch-and-solve least squares problems without accessing all entries of the right-hand side $\mathbf{Y}$ while still providing performance guarantees. Here we discuss how to take advantage of the structure of $\mathbf{A}$ to speed up leverage score sampling.

Kronecker product structure

Consider a Kronecker product $\mathbf{A} = \mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_L$ of $m_k \times n_k$ matrices $\mathbf{A}_k$. It is possible to perform *exact* leverage score sampling on $\mathbf{A}$ without even forming it. Cheng et al. [CPL+16] used this fact to approximately solve least squares problems with Kronecker product design matrices, which has applications in algorithms for Tucker tensor decomposition. Formal statements and proofs of these results later appeared in [DJS+19].

To see how the sampling works, let (p_i) be the leverage score sampling distribution of $\mathbf{A}$ and let (p_{i_k}) be the leverage score sampling distribution of $\mathbf{A}_k$ for $k \in [\![L]\!]$. For any $i \in [\![\prod_{k=1}^{L} m_k]\!]$ and corresponding multi-index $(i_1, \ldots, i_L)$ satisfying

$$\mathbf{A}[i, :] = \mathbf{A}_1[i_1, :] \otimes \cdots \otimes \mathbf{A}_L[i_L, :], \tag{7.7}$$

it holds that

$$p_i = p_{i_1}^{(1)} p_{i_2}^{(2)} \cdots p_{i_L}^{(L)}. \tag{7.8}$$

Therefore, instead of drawing an index i according to (p_i), one can draw the index i_k according to $(p_{i_k}^{(k)})$ for each $k \in [\![L]\!]$. Due to (7.7), the row corresponding to the drawn index can be computed and rescaled without constructing $\mathbf{A}$. This process can be easily adapted to drawing multiple samples.

Fahrbach et al. [FFG22] discuss how the sampling approach above can be adapted for use in ridge regression when the design matrix is a Kronecker product. Malik et al. [MXC+22] show an approach for efficient sampling according to the exact leverage scores of matrices of the form $\mathbf{A}[:, J]$ when $\mathbf{A}$ is a Kronecker product and J is an index vector that satisfies certain monotonicity properties.

Khatri–Rao product structure

Sampling according to the leverage scores of a Khatri–Rao product matrix $\mathbf{A} = \mathbf{A}_1 \odot \cdots \odot \mathbf{A}_L$ is more challenging than it is for a Kronecker product matrix. Still, several approaches for doing so have been proposed. We divide them into two categories. The methods in the first category sample according to the leverage scores of the *Kronecker product* of $\mathbf{A}_1, \ldots, \mathbf{A}_L$ instead of the Khatri–Rao product since this allows for simple and efficient sampling. This can be viewed as sampling from a coarse approximation of the Khatri–Rao product leverage scores. The methods in

the second category sample according to exact or high-accuracy approximations of the Khatri–Rao product leverage score distribution.

Sampling according to Kronecker product leverage scores As noted by [CPL+16; BBK18], the leverage scores of $\mathbf{A}$ can be upper bounded by

$$\ell_i(\mathbf{A}) \leq \prod_{k=1}^{L} \ell_{i_k}(\mathbf{A}_k), \tag{7.9}$$

where $(i_1, \ldots, i_L)$ is the multi-index corresponding to i. The two papers [CPL+16; LK20] use the expression on the right-hand side of (7.9) as an approximation to the exact leverage scores on the left-hand side. By using the bound (7.9), they are able to prove theoretical performance guarantees when this approach is used for sketch-and-solve in least squares problems. More precisely, Cheng et al. [CPL+16] sample according to a mixture of the distribution in (7.8) and a distribution which depends on the magnitude of the dependent variables (i.e., the entries in the "right-hand sides" in the least squares problem). Larsen and Kolda [LK20] sample with respect to only the distribution in (7.8). Bharadwaj et al. [BMM+22] extend the work by [CPL+16; LK20] to a distributed-memory setting and provide high-performance parallel implementations. Ideas similar to those in [LK20] are developed for the more complicated design matrices that arise in algorithms for tensor ring decomposition in [MB21]. Those matrices have columns that are sums of vectors with Kronecker product structure.

Sampling according to exact or high-quality approximations of leverage scores Malik [Mal22] proposes a different approach for the Khatri–Rao product least squares problem. By combining some of the ideas for fast leverage score estimation (see §6.2) and recursive sketching (see §7.2.4) with a sequential sampling approach, he improves the sampling and computational complexities of [LK20]. Malik et al. [MBM22] simplify and generalize the method by [Mal22] to a wider family of structured matrices. An upshot of this work is a method for efficiently sampling a Khatri–Rao product matrix according to its *exact* leverage score distribution without forming the matrix.

Motivated by applications in kernel methods, [WZ20] develop a recursive leverage score sampling method for sketching of matrices of the form $\mathbf{A}^{\odot L} = \mathbf{A} \odot \cdots \odot \mathbf{A}$. Their method starts by sampling from a coarse approximation to the leverage score sampling distribution and then iteratively refining it. These ideas are further refined in [WZ22] where the method is also extended to general Khatri–Rao products of matrices that can all be distinct.

7.3 Partial updates to Kronecker product sketches

The structured sketching operators discussed in Section 7.2 are notable in that they are defined in terms of multiple smaller sketching operators. Here we discuss situations when it is advantageous to reuse some of these smaller sketches across multiple calls to the structured sketching operator. The examples we discuss come from works that use sketching in tensor decomposition algorithms. Our goal with this discussion is to bring to light some of the functionality we think is important

for structured sketches to have in order to best support potential usage in the tensor community.

By tensor, we mean multi-index arrays containing real numbers. A tensor with L indices is called an *L-way tensor*. Vectors and matrices are one- and two-way tensors, respectively. Calligraphic capital letters (e.g., $\mathcal{X}$) are used to denote tensors with three or more indices. Much like matrix decomposition, the purpose of tensor decomposition is to decompose a tensor into some number of simpler components. We only give minimal background material on tensor decomposition here; see the review papers [KB09; CMD+15; SDF+17] for further details.

7.3.1 Background on the CP decomposition

We first consider the CANDECOMP/PARAFAC (CP) decomposition which is also known as the canonical polyadic decomposition [KB09, §3]. It decomposes an L-way tensor $\mathcal{X}$ of size $m_1 \times m_2 \times \cdots \times m_L$ into a sum of R rank-1 tensors:

$$\mathcal{X} = \sum_{r=1}^{R} \boldsymbol{a}_r^{(1)} \circ \boldsymbol{a}_r^{(2)} \circ \cdots \circ \boldsymbol{a}_r^{(L)}, \tag{7.10}$$

where $\circ$ denotes the outer product. The $m_n \times R$ matrices $\mathbf{A}^{(n)} = \begin{bmatrix} \boldsymbol{a}_1^{(n)} & \cdots & \boldsymbol{a}_R^{(n)} \end{bmatrix}$ for $n \in [\![L]\!]$ are called factor matrices. When R is sufficiently large, we can express the factor matrices as the solution to

$$\underset{\mathbf{A}^{(1)},\ldots,\mathbf{A}^{(L)}}{\arg\min} \left\| \mathcal{X} - \sum_{r=1}^{R} \boldsymbol{a}_r^{(1)} \circ \boldsymbol{a}_r^{(2)} \circ \cdots \circ \boldsymbol{a}_r^{(L)} \right\|_{\mathrm{F}}, \tag{7.11}$$

where $\|\cdot\|_{\mathrm{F}}$ denotes the Frobenius norm as generalized to tensors in the obvious way. The broader problem of tensor decomposition is concerned with approximately solving (7.11). In particular, it is common to seek locally optimal solutions to this problem even when R is too small for an identity of the form (7.10) to hold for $\mathcal{X}$.

It is computationally intractable to solve (7.11) in the general case. However, the problem admits several heuristics that are effective in practice. One of the most popular heuristics is alternating minimization, wherein one solves for only one factor matrix at a time while keeping the others fixed. That is, one solves a sequence of problems of the form

$$\mathbf{A}^{(n)} = \underset{\mathbf{A}}{\arg\min} \left\| \mathcal{X} - \sum_{r=1}^{R} \boldsymbol{a}_r^{(1)} \circ \cdots \circ \boldsymbol{a}_r^{(n-1)} \circ \boldsymbol{a}_r \circ \boldsymbol{a}_r^{(n+1)} \circ \cdots \circ \boldsymbol{a}_r^{(L)} \right\|_{\mathrm{F}} \tag{7.12}$$

for $n \in [\![L]\!]$. If we adopt appropriate notation then (7.12) can be stated as a familiar linear least squares problem. Accordingly, this alternating minimization approach is called *alternating least squares* (ALS). The ALS approach cycles through the indices $n \in [\![L]\!]$ multiple times until some termination criteria is met. Typical termination criteria include reaching a maximum number of iterations or seeing that the improvement in the objective falls below some threshold.

Formulating and solving the least squares problem

We begin by introducing flattened representations of $\mathcal{X}$. Specifically, for $n \in [\![L]\!]$, the $m_n \times \left(\prod_{j \neq n} m_j\right)$ matrix $\mathbf{X}_{(n)}$ is given by horizontally concatenating the mode-n

fibers $\mathcal{X}[i_1, \ldots, i_{n-1}, :, i_{n+1}, \ldots, i_N]$ as columns. Such a matrix can be expressed in Matlab notation as follows

$$\mathbf{X}_{(n)} = \texttt{reshape}\left(\texttt{permute}(\mathcal{X}, [n, 1, \ldots, n-1, n+1, \ldots, L]), m_n, \prod_{j \neq n} m_j\right). \tag{7.13}$$

Next, we introduce the following flattened tensorizations of the factor matrices:

$$\mathbf{A}^{\neq n} := \mathbf{A}^{(L)} \odot \cdots \odot \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \odot \cdots \odot \mathbf{A}^{(1)} =: \bigodot_{\substack{j=L \\ j \neq n}}^{1} \mathbf{A}^{(j)}. \tag{7.14}$$

Where we emphasize that the order of the matrices in the above product is important; our notation for the Khatri–Rao product at right reflects how the order proceeds from $j = L$ to $j = 1$, skipping $j = n$.

In terms of these matrices, the ALS update rule for the n^{th} factor matrix is

$$\mathbf{A}^{(n)} = \arg\min_{\mathbf{A}} \left\| \mathbf{A}^{\neq n} \mathbf{A}^* - \mathbf{X}_{(n)}^* \right\|_{\text{F}}. \tag{7.15}$$

We note that the Gram matrix for this least squares problem can be computed efficiently by the formula

$$\begin{aligned} \mathbf{A}^{\neq n *} \mathbf{A}^{\neq n} = (\mathbf{A}^{(L)*} \mathbf{A}^{(L)}) \circledast \cdots \circledast (\mathbf{A}^{(n+1)*} \mathbf{A}^{(n+1)}) \\ \circledast (\mathbf{A}^{(n-1)*} \mathbf{A}^{(n-1)}) \circledast \cdots \circledast (\mathbf{A}^{(1)*} \mathbf{A}^{(1)}). \end{aligned} \tag{7.16}$$

Therefore solving the least squares problem in (7.15) via the normal equations can be very efficient [KB09, §3.4]. Indeed, the ALS update rule for the n^{th} factor matrix becomes

$$\mathbf{A}^{(n)} = \mathbf{X}_{(n)} \mathbf{A}^{\neq n} (\mathbf{A}^{\neq n *} \mathbf{A}^{\neq n})^{\dagger}. \tag{7.17}$$

The most expensive part of this update is actually computing $\mathbf{X}_{(n)} \mathbf{A}^{\neq n}$ [BBK18, §3.1.1], which is analogous to the vector $\mathbf{F}^* \boldsymbol{h}$ in the normal equations for an overdetermined least squares problem $\min_{\boldsymbol{z}} \|\mathbf{F}\boldsymbol{z} - \boldsymbol{h}\|_2^2$. Therefore, the fact that computing this matrix is the computational bottleneck in solving (7.15) is the opposite of what one would expect when not working with tensors. This phenomenon is why row-sampling sketching operators have been successful in ALS algorithms that use sketch-and-solve for the least squares subproblems [LK20].

Remark 7.3.1. Although it is cheap to form the Gram matrix (7.16), there is potential for *very* bad conditioning even when L is small. We do not know how seriously the poor conditioning affects ALS approaches to CP decomposition in practice.

7.3.2 Sketching for the CP decomposition

Battaglino et al. [BBK18] apply the Kronecker SRFT from Section 7.2.2 to the least squares problem in (7.15). Letting $\mathbf{T}_j$ and $\mathbf{F}_j$ be of size $m_j \times m_j$, the sketching operator used before solving for the n^{th} factor matrix is

$$\mathbf{S}_n = \sqrt{\frac{\prod_{\substack{j=1 \\ j \neq n}}^{L} m_j}{d}} \mathbf{R} \Big(\bigotimes_{\substack{j=L \\ j \neq n}}^{1} \mathbf{T}_j \Big) \Big(\bigotimes_{\substack{j=L \\ j \neq n}}^{1} \mathbf{F}_j \Big). \tag{7.18}$$

Our notation for the Kronecker product operator indexes from $j = L$ to $j = 1$ so as to mimic our earlier notation for the Khatri–Rao product (see (7.14)).

A by-the-book application of this operator would require drawing new $\mathbf{R}$ and $(\mathbf{F}_j)_{j\neq n}$ every time it is applied in (7.15), i.e., L times for every execution of the for loop. Battaglino et al. [BBK18, Alg. 4] instead propose drawing $\mathbf{F}_1, \dots, \mathbf{F}_L$ once and then reusing them throughout the algorithm, only drawing $\mathbf{R}$ anew for each least squares problem. This reduces the computational cost considerably since it allows for greater reuse of various computed quantities. More specifically, the expensive application of the full Kronecker SRFT to the unstructured matrix $\mathbf{X}^*_{(n)}$ does not have to be computed for every least squares problem.

Larsen and Kolda [LK20] also sketch the least squares problems in (7.15). They use the efficient leverage score sampling scheme for Khatri–Rao products discussed in Section 7.2.5. This approach also allows for some reuse between subsequent sketches. When solving for $\mathbf{A}^{(n)}$ in (7.15), a row with multi-index $(i_1, \dots, i_{n-1}, i_{n+1}, \dots, i_L)$ is sampled with probability $p^{(1)}_{i_1} \cdots p^{(n-1)}_{i_{n-1}} p^{(n+1)}_{i_{n+1}} \cdots p^{(L)}_{i_L}$, where $(p^{(k)}_{i_k})$ is the leverage score sampling distribution for $\mathbf{A}^{(k)}$. Since each $\mathbf{A}^{(k)}$ only change for every L^{th} least squares problem, the probability distribution $(p^{(k)}_{i_k})$ can be used in $L-1$ least squares problems before it needs to be recomputed.

7.3.3 Background on the Tucker decomposition

The Tucker decomposition [KB09, §4] is another popular method that decomposes an L-way tensor $\mathcal{X}$ of size $m_1 \times m_2 \times \cdots \times m_L$ into

$$\sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \cdots \sum_{r_L=1}^{R_L} \mathcal{G}[r_1, r_2, \dots, r_L]\, \boldsymbol{a}^{(1)}_{r_1} \circ \boldsymbol{a}^{(2)}_{r_2} \circ \cdots \circ \boldsymbol{a}^{(L)}_{r_L}, \tag{7.19}$$

where the so-called *core tensor* $\mathcal{G}$ is of size $R_1 \times R_2 \times \cdots \times R_L$. The $m_n \times R_n$ matrices $\mathbf{A}^{(n)} = \begin{bmatrix} \boldsymbol{a}^{(n)}_1 & \cdots & \boldsymbol{a}^{(n)}_{R_n} \end{bmatrix}$ for $n \in [\![L]\!]$ are called factor matrices. Similarly to the CP decomposition, the core tensor and factor matrices can be initialized randomly and then updated iteratively via ALS:[3]

$$\text{For } n \text{ in } [\![L]\!]: \quad \mathbf{A}^{(n)} = \arg\min_{\mathbf{A}} \|\mathbf{B}^{\neq n} \mathbf{G}^*_{(n)} \mathbf{A}^* - \mathbf{X}^*_{(n)}\|_{\mathrm{F}}, \tag{7.20}$$

$$\mathcal{G} = \arg\min_{\mathcal{Z}} \|\mathbf{B} \operatorname{vec}(\mathcal{Z}) - \operatorname{vec}(\mathcal{X})\|_{\mathrm{F}}, \tag{7.21}$$

where

$$\mathbf{B}^{\neq n} = \mathbf{A}^{(L)} \otimes \cdots \otimes \mathbf{A}^{(n+1)} \otimes \mathbf{A}^{(n-1)} \otimes \cdots \otimes \mathbf{A}^{(1)},$$
$$\mathbf{B} = \mathbf{A}^{(L)} \otimes \cdots \otimes \mathbf{A}^{(1)},$$

and the unfolding $\mathbf{G}_{(n)}$ is defined analogously to $\mathbf{X}_{(n)}$ in (7.13). The steps in (7.20) and (7.21) are then repeated until some convergence criterion is met. We note that the least squares problems (7.20) and (7.21) are highly overdetermined when $(R_n)_{n\in[\![L]\!]}$ are small compared to $(m_n)_{n\in[\![L]\!]}$.

[3] The update rules in (7.20) and (7.21) have been formulated as least squares problems in order to show where sketching can be applied in the ALS algorithm. A more standard formulation of the update rules can be found in [KB09, §4.2].

7.3.4 Sketching for the Tucker decomposition

Malik and Becker [MB18] apply the TensorSketch discussed in Section 7.2.3 to these problems. From a straightforward adaption of (7.5) to matrix Kronecker products, we have that the TensorSketch of the design matrix $\mathbf{B}^{\neq n}$ is computed via the formula

$$\mathrm{DFT}^{-1}\left(\left(\bigodot_{\substack{j=L\\ j\neq n}}^{1}\left(\mathrm{DFT}(\mathbf{S}_j\mathbf{A}^{(j)})\right)^{\top}\right)^{\top}\right),$$

where $\mathbf{S}_j$ is a $d\times m_j$ CountSketch, and where $\top$ denotes transpose without complex conjugation. The formula for sketching $\mathbf{B}$ is the same except for that it also includes the n^{th} term in the Khatri–Rao product.

Instead of drawing new CountSketches for every application of TensorSketch, [MB18, Alg. 2] draw two sets of independent CountSketches at the start of the algorithm: $(\mathbf{S}_j^{(1)})_{j=1}^{L}$ where $\mathbf{S}_j^{(1)}$ is of size $d_1\times m_j$, and $(\mathbf{S}_j^{(2)})_{j=1}^{L}$ where $\mathbf{S}_j^{(2)}$ is of size $d_2\times m_j$. These two sets of sketches are then reused throughout the algorithm: $(\mathbf{S}_j^{(1)})$ are used for sketching (7.20) and $(\mathbf{S}_j^{(2)})$ are used for sketching (7.21). The latter least squares problems are much larger than the former. Using two sets of sketching operators makes it possible to choose a larger embedding dimension for the latter problem, i.e., choosing $d_2 > d_1$. By reusing CountSketches in this fashion, considerable improvement in run time is achieved. Moreover, it is possible to compute all relevant sketches of unfoldings of $\mathcal{X}$ at the start of the algorithm, leading to an algorithm that requires only a single pass of $\mathcal{X}$ in order to decompose it.

7.3.5 Implementation considerations

We deem it most appropriate to implement the structured sketches discussed in Section 7.2 in RandLAPACK rather than RandBLAS. In order to facilitate the applications discussed in Section 7.3, it should be possible to update or redraw specific components of the sketching operator after it has been created. For example, when applying the operator in (7.18) in an ALS algorithm for CP decomposition as in [BBK18, Alg. 4], we want to keep the random diagonal matrices $\mathbf{F}_1,\ldots,\mathbf{F}_L$ fixed but draw a new sampling operator $\mathbf{R}$ before each application of $\mathbf{S}_n$.

In the applications above, components are shared across the L different sketching operators that are used when updating the L different factor matrices. Rather than defining L different sketching operators with shared components, it is better to define a single operator that contains all components and which allows "leaving one component out" when applied to a matrix or vector. For example, consider a Kronecker SRFT from (7.4) but with reversed order in the Kronecker products. It contains the components $\mathbf{R}$ and $\mathbf{F}_1,\ldots,\mathbf{F}_L$. A user should be able to supply a parameter n which indicates that the n^{th} term in the Kronecker products should be left out when computing the sketch, resulting in a sketch of the form (7.18).

Appendix A

Details on Basic Sketching

This appendix covers sketching theory and implementation of sketching operators. Its contents are relevant to Sections 2, 3 and 6.

A.1 Subspace embeddings and effective distortion

Let $\mathbf{S}$ be a wide $d \times m$ sketching operator and L be a linear subspace of $\mathbb{R}^m$. Recall from Section 2.2.2 that $\mathbf{S}$ *embeds* L *into* $\mathbb{R}^d$ *with distortion* $\delta \in [0, 1]$ if

$$(1 - \delta)\|\boldsymbol{x}\|_2 \leq \|\mathbf{S}\boldsymbol{x}\|_2 \leq (1 + \delta)\|\boldsymbol{x}\|_2$$

holds for all $\boldsymbol{x}$ in L. We often use the term δ-*embedding* for such an operator. Note that if $\mathbf{S}$ is a δ-embedding and δ' is greater than δ, then $\mathbf{S}$ is also a δ'-embedding. It can be useful to speak of the smallest distortion δ for which $\mathbf{S}$ is a δ-embedding for L; we call this *the distortion of* $\mathbf{S}$ *for* L, and denote it by

$$\mathscr{D}(\mathbf{S}; L) = \inf\{\, \delta \;: 0 \leq \delta \leq 1 \\ \mathbf{S} \text{ is a } \delta\text{-embedding for } L\}.$$

In this notation, we have $\mathscr{D}(\mathbf{S}; L) \geq 1$ when $\ker \mathbf{S} \cap L$ is nontrivial. If there is a unit vector $\boldsymbol{x}$ in L where $\|\mathbf{S}\boldsymbol{x}\| > 2$, then $\mathscr{D}(\mathbf{S}; L) = +\infty$.

Subspace embedding distortion has a significant limitation in that it depends on the scale of $\mathbf{S}$, while many RandNLA algorithms have no such dependence. This shortcoming leads us to define the *effective distortion of* $\mathbf{S}$ *for* L as

$$\mathscr{D}_{\mathrm{e}}(\mathbf{S}; L) = \inf_{t>0} \mathscr{D}(t\mathbf{S}; L). \tag{A.1}$$

Here, the infimum is over $t > 0$ rather than $t \neq 0$ since $\mathscr{D}(\mathbf{S}; L) = \mathscr{D}(-\mathbf{S}; L)$.

There is a convenient formula for effective distortion using concepts of *restricted singular values* and *restricted condition numbers*. Restricted singular values are a fairly general concept of use in random matrix theory; see, e.g., [OT17]. They are measures an operator's "size" when considered from different vantage points within a set of interest. Formally, we define the largest and smallest restricted singular values of a sketching operator $\mathbf{S}$ for a subspace L as

$$\sigma_{\max}(\mathbf{S}; L) = \max_{\boldsymbol{x} \in L} \{\|\mathbf{S}\boldsymbol{x}\|_2 \ : \ \|\boldsymbol{x}\|_2 = 1\}$$

and

$$\sigma_{\min}(\mathbf{S}; L) = \min_{\boldsymbol{x} \in L} \{\|\mathbf{S}\boldsymbol{x}\|_2 \ : \ \|\boldsymbol{x}\|_2 = 1\}$$

Given these concepts, we define the restricted condition number of $\mathbf{S}$ on L as

$$\operatorname{cond}(\mathbf{S}; L) = \frac{\sigma_{\max}(\mathbf{S}; L)}{\sigma_{\min}(\mathbf{S}; L)},$$

where we take $c/0 = +\infty$ for any $c \geq 0$.

We have formulated the concepts of restricted singular values and condition numbers in a way that reflects their geometric meaning. More concrete descriptions can be obtained by considering any matrix $\mathbf{U}$ whose columns provide an orthonormal basis for L. With this one can see that $\sigma_{\min}(\mathbf{S}; L)$ and $\sigma_{\max}(\mathbf{S}; L)$ coincide with the extreme singular values of $\mathbf{SU}$, and that $\operatorname{cond}(\mathbf{S}; L) = \operatorname{cond}(\mathbf{SU})$.

Next, we provide the connection between restricted condition numbers and effective distortion. Appendix B.1.1 explores this connection more in the context of sketch-and-precondition algorithms for saddle point problems.

Proposition A.1.1. *Let L be a linear subspace and $\mathbf{S}$ be a sketching operator on L. The effective distortion of $\mathbf{S}$ for L is*

$$\mathscr{D}_{\mathrm{e}}(\mathbf{S}; L) = \frac{\kappa - 1}{\kappa + 1}$$

where we take $(\infty - 1)/(\infty + 1) = 1$.

Proof. The scaled sketching operator $t\mathbf{S}$ is a δ-embedding for L if and only if

$$(1 - \delta)\|\boldsymbol{x}\|_2 \leq t\|\mathbf{S}\boldsymbol{x}\|_2 \leq (1 + \delta)\|\boldsymbol{x}\|_2 \quad \text{for all } \boldsymbol{x} \text{ in } L.$$

This is equivalent to

$$\frac{1 - \delta}{t} \leq \frac{\|\mathbf{S}\boldsymbol{x}\|_2}{\|\boldsymbol{x}\|_2} \quad \text{and} \quad \frac{\|\mathbf{S}\boldsymbol{x}\|_2}{\|\boldsymbol{x}\|_2} \leq \frac{1 + \delta}{t} \quad \text{for all } \boldsymbol{x} \text{ in } L \setminus \{\mathbf{0}\}.$$

To simplify these bounds, we optimize over $\boldsymbol{x}$. Abbreviating $\sigma_1 := \sigma_{\max}(\mathbf{S}; L)$ and $\sigma_n := \sigma_{\max}(\mathbf{S}; L)$ for $n = \dim(L)$, we find that $t\mathbf{S}$ is a δ-embedding if and only if

$$\frac{1 - \delta}{t} \leq \sigma_n \quad \text{and} \quad \frac{1 + \delta}{t} \leq \sigma_1.$$

These identities can be rearranged to find the following constraints on δ:

$$1 - \sigma_1 t \leq \delta \quad \text{and} \quad t\sigma_n - 1 \leq \delta.$$

The value of t which permits minimum δ is that which makes the lower bounds coincide. That is, the optimal t is $t_\star = 2/(\sigma_1 + \sigma_n)$. Plugging this into the bounds above, our constraints on δ reduce to

$$\delta \geq 1 - \sigma_1 t_\star = t_\star \sigma_n - 1 = \frac{\sigma_1 - \sigma_n}{\sigma_1 + \sigma_n} = \frac{\kappa - 1}{\kappa + 1},$$

as desired. □

A.1.1 Effective distortion of Gaussian operators

It is informative to consider the concepts of restricted condition numbers and effective distortion for Gaussian sketching operators. Therefore, let us suppose that our $d \times m$ sketching operator $\mathbf{S}$ has iid mean-zero Gaussian entries, and consider an n-dimensional subspace L in $\mathbb{R}^m$. By rotational invariance of Gaussian distribution, we can infer that the distribution of $\operatorname{cond}(\mathbf{S}; L)$ coincides with that of $\operatorname{cond}(\tilde{\mathbf{S}})$ for a $d \times n$ Gaussian matrix $\tilde{\mathbf{S}}$. Strong concentration results are available to understand the distribution of $\operatorname{cond}(\tilde{\mathbf{S}})$.

Specifically, letting $d = sn$ for a constant $s > 1$, results by Silverstein [Sil85] and Geman [Gem80] imply

$$\operatorname{cond}(\mathbf{S}; L) \to \frac{\sqrt{s} + 1}{\sqrt{s} - 1} \quad \text{almost surely as} \quad n \to \infty. \tag{A.2}$$

This can be turned around using Proposition A.1.1 to obtain

$$\mathscr{D}_{\mathrm{e}}(\mathbf{S}; L) \to \frac{1}{\sqrt{s}} \quad \text{almost surely as} \quad n \to \infty. \tag{A.3}$$

We emphasize that (A.2) and (A.3) hold for any fixed n-dimensional subspace L. These facts justify aggressively small choices of embedding dimension when using Gaussian sketching operators in randomized algorithms for least squares. Meng, Saunders, and Mahoney come to the same conclusion in their work on LSRN [MSM14, Theorem 4.4].

A.2 Short-axis-sparse sketching operators

In this appendix we make liberal use of the abbreviation *SASO* (for "short-axis-sparse sketching operator") introduced in Section 2.4.1. Without loss of generality, we describe SASOs in the wide case, i.e., when $\mathbf{S}$ is $d \times m$ with $d \ll m$.

A.2.1 Implementation notes

Constructing SASOs column-wise

It is extremely cheap to construct and store a wide SASO. The precise storage format depends on how one wants to apply the SASO later on, which can vary depending on context. However, the construction is embarrassingly parallel across columns provided one uses CBRNGs (counter-based random number generators; see §2.1.1), and this structure leads to canonical methods for generating SASOs.

We first consider the SASO construction where row indices are partitioned into index sets $I_1, \dots, I_k$ of roughly equal size. Given such a partition, the indices of

nonzeros for a given column are chosen by taking one element (independently and uniformly) from each of the index sets I_j. The naive implementation can sample these row indices with k parallel calls to the CBRNG.

Now consider the construction where the row indices for a column are chosen by selecting k elements from $[\![d]\!]$ uniformly without replacement. This can be done in $O(km)$ time by using Fisher-Yates sampling and carefully re-using workspace. For a concrete implementation, we refer the reader to

> https://github.com/BallisticLA/RandBLAS/blob/19sept22/src/sjlts.cc#L14-L78.[1]

While the implementation above is sequential, it is easy to parallelize. Given T threads, the natural modification to the algorithm takes $O(mk/T)$ time and requires $O(Td)$ workspace. The constants in the big-O notation are small.

Remarks on storage formats

It is reasonable for a standard library to restrict SASOs to only the most common sparse matrix formats. We believe both compressed sparse row (CSR) and compressed sparse column (CSC) are worth considering. CSC is the more natural of the two since (wide) SASOs are constructed columnwise. If CSR format is preferred for some reason, then we recommend constructing $\mathbf{S}$ columnwise while retaining extra data to facilitate conversion to CSR.

In principle, if the nonzero entries of $\mathbf{S}$ are ± 1 *and* CSC is used as the storage format, then one could do away with storing the nonzero values explicitly; one could instead store the sign information using signed integers for the row indices. We do not favor this approach since it precludes working with SASOs with more than two distinct nonzero values.

For the matrices $\mathbf{A}$ and $\mathbf{A}_{\mathrm{sk}}$, we must consider whether they are in column-major or row-major format. Indeed, both formats need to be supported since Section 3 framed all least squares problems with tall data matrices. While this was without loss of generality from a mathematical perspective, a user with an underdetermined least squares problem involving a wide column-major data matrix $\mathbf{B}$ is effectively needing to sketch the tall row-major matrix $\mathbf{A} = \mathbf{B}^*$.

Applying a wide SASO

There are four combinations of storage formats we need to consider for $(\mathbf{S}, \mathbf{A})$.

$\mathbf{S}$ is CSC, $\mathbf{A}$ is row-major. Suppose we have P processors. Our suggested approach is to partition the row index set $[\![d]\!]$ into $I_1, \ldots, I_P$ and to have each processor be responsible for its own block of rows. The p^{th} processor computes its row block by streaming over the columns of $\mathbf{S}$ and rows of $\mathbf{A}$, accumulating outer products as indicated below

$$\mathbf{A}_{\mathrm{sk}}[I_p, :] = \sum_{\ell \in [\![m]\!]} \mathbf{S}[I_p, \ell]\mathbf{A}[\ell, :].$$

An individual term $\mathbf{S}[I_p, \ell]\mathbf{A}[\ell, :]$ can cheaply be accumulated into $\mathbf{A}_{\mathrm{sk}}[I_p, :]$ by using the fact that $\mathbf{S}[I_p, \ell]$ is extremely sparse. If R denotes the number of nonzeros in $\mathbf{S}[I_p, \ell]$, then the outer-product accumulation can be computed with R `axpy`

[1] This code was written when we used the term "SJLT" for what we now call a "SASO."

operations involving the row $\mathbf{A}[\ell, :]$. Note that since $\mathbf{S}$ has k nonzeros per column (with row indices distributed uniformly at random), this value R is a sum of $|I_p|$ iid Bernoulli random variables with mean k/d. Therefore the expected number of `axpy`'s performed by processor p for term ℓ is $|I_p|k/d$.

$\mathbf{S}$ *is CSR,* $\mathbf{A}$ *is row-major.* Here, we suggest that the d rows of $\mathbf{A}_{\text{sk}}$ be computed separately from one another. An individual row is given by $\mathbf{A}_{\text{sk}}[i, :] = \mathbf{S}[i, :]\mathbf{A}$. Evaluating the product of this sparse vector and dense matrix can be done by taking a linear combination of a small number of rows of $\mathbf{A}$. Specifically, if R is the number of nonzeros in $\mathbf{S}[i, :]$ then computing $\mathbf{A}_{\text{sk}}[i, :]$ only requires R rows from $\mathbf{A}$. Since the columns of $\mathbf{S}$ are independent, R is a sum of m iid Bernoulli random variables with mean k/d. Therefore we expect to access mk/d rows of $\mathbf{A}$ in order to compute $\mathbf{A}_{\text{sk}}[i, :]$.

$\mathbf{S}$ *is CSC,* $\mathbf{A}$ *is column-major.* Here, we suggest that the n columns of $\mathbf{A}_{\text{sk}}$ be computed separately from one another. An individual column is given by $\mathbf{A}_{\text{sk}}[:, j] = \mathbf{SA}[:, j]$. We evaluate this product by taking a linear combination of the columns of $\mathbf{S}$, according to

$$\mathbf{A}_{\text{sk}}[:, j] = \sum_{\ell \in [\![m]\!]} \mathbf{S}[:, \ell]\mathbf{A}[\ell, j].$$

Note that each of the ℓ terms in this sum is a sparse vector with k nonzero entries. Based on our preliminary experiments, this method has mediocre single-thread performance, but it has excellent scaling properties for many-core machines.

$\mathbf{S}$ *is CSR,* $\mathbf{A}$ *is column-major.* We were unable to determine a method that parallelizes well for this pair of data formats. The most efficient algorithm may be to convert $\mathbf{S}$ to CSC and then to apply the preferred method when $\mathbf{S}$ is CSC and $\mathbf{A}$ is column-major.

A.2.2 Theory and practical usage

SASO theory

A precursor to the SASOs we consider is described in [DKS10], which sampled row indices for nonzero entries from $[\![d]\!]$ with replacement. The first theoretical analysis of the SASOs we consider was conducted in [KN12] and concerned the distributional Johnson-Lindenstrauss property. Shortly thereafter, [CW13] and [MM13] studied OSE properties for SASOs with a single nonzero per column; the latter referred to the construction as "CountSketch."

Theoretical analyses for OSE properties of general SASOs (i.e., those with more than one nonzero per column) were first carried out by [NN13; KN14] and subsequently improved by [BDN15; Coh16]. Much of the SASO analysis has been through the lens of "hashing functions," and does not require that the columns of the sketching operator are fully independent. We do not know of any practical advantage to SASOs with partial dependence across the columns.

Remark A.2.1 (Navigating the literature). [CW17] is a longer journal version of [CW13]. [KN14] and [KN12] have the same title, and the former is considered a more developed journal version of the latter.

Selecting parameters for SASOs

We are in the process of developing recommendations for how to set the parameters d and k for a distribution over SASOs. So far we have observed that when d is fixed the sketch quality increases rapidly with k before reaching a plateau. As one point of reference, we have observed that there is no real benefit in k being larger than eight when embedding the range of a $100,000 \times 2,000$ matrix into a space with ambient dimension $d = 6,000$. Furthermore, for the data matrices we tested, the restricted condition numbers of those sketching operators were tightly concentrated at $O(1)$. Extensive experiments with parameter selection for SASOs in a least squares context are given in [Ura13].

A.3 Theory for sketching by row selection

Here we prove Proposition 6.1.1. Our proof is inspired by [Tro20, Problem 5.13], which begins with the following adaptation of [Tro15, Theorem 5.1.1].

Theorem A.3.1. *Consider an independent family $\{\mathbf{X}_1, \dots, \mathbf{X}_s\} \subset \mathbb{H}^n$ of random psd matrices that satisfy $\lambda_{\max}(\mathbf{X}_i) \le L$ almost surely. Let $\mathbf{Y} = \sum_{i=1}^{s} \mathbf{X}_i$, and define the mean parameters*

$$\mu_{\max} = \lambda_{\max}(\mathbb{E}\mathbf{Y}) \quad \text{and} \quad \mu_{\min} = \lambda_{\min}(\mathbb{E}\mathbf{Y}).$$

One has that

$$\Pr\{\lambda_{\max}(\mathbf{Y} - (1+t)\mathbb{E}\mathbf{Y}) \ge 0\} \le n \left[\frac{\exp(t)}{(1+t)^{(1+t)}}\right]^{\mu_{\max}/L} \quad \text{for } t > 0, \quad \text{and}$$

$$\Pr\{\lambda_{\max}((1-t)\mathbb{E}\mathbf{Y} - \mathbf{Y}) \ge 0\} \le n \left[\frac{\exp(-t)}{(1-t)^{(1-t)}}\right]^{\mu_{\min}/L} \quad \text{for } t \in (0,1).$$

Here, we restate the result we aim to prove.

Proposition A.3.2 (Adaptation of Proposition 6.1.1)**.** *Suppose $\mathbf{A}$ is an $m \times n$ matrix of rank n, $\boldsymbol{q}$ is a distribution over $[\![m]\!]$, and t is in $(0,1)$. Let $\mathbf{S}$ be a $d \times m$ sketching operator with rows that are distributed iid as*

$$\mathbf{S}[i,:] = \frac{\boldsymbol{\delta}_j}{\sqrt{dq_j}} \text{ with probability } q_j,$$

and let $E(t, \mathbf{S})$ denote the event that

$$(1-t)\|\boldsymbol{y}\|_2^2 \le \|\mathbf{S}\boldsymbol{y}\|_2^2 \le (1+t)\|\boldsymbol{y}\|_2^2 \quad \forall\, \boldsymbol{y} \in \operatorname{range} \mathbf{A}.$$

Using $r := \min_{j \in [\![m]\!]} \frac{q_j}{p_j(\mathbf{A})}$, we have

$$\Pr\{E(t, \mathbf{S}) \text{ fails}\} \le 2n \left(\frac{\exp(t)}{(1+t)^{(1+t)}}\right)^{rd/n}.$$

Proof. The way that we use Theorem A.3.1 is along the lines of the hint in [Tro20, Problem 5.13, Part 3]. We begin by considering the Gram matrices $\mathbf{G} = \mathbf{A}^*\mathbf{A}$ and $\mathbf{G}_{\mathrm{sk}} = \mathbf{A}^*\mathbf{S}^*\mathbf{S}\mathbf{A}$. The event $E(t, \mathbf{S})$ is equivalent to

$$(1-t)\mathbf{I}_n \preceq \mathbf{G}^{-1/2}\mathbf{G}_{\mathrm{sk}}\mathbf{G}^{-1/2} \preceq (1+t)\mathbf{I}_n.$$

The sketched Gram matrix can be expressed as a sum of d outer products of rows of $\mathbf{SA}$. Each of the d outer products is conjugated by $\mathbf{G}^{-1/2}$ to obtain our matrices $\{\mathbf{X}_1, \ldots, \mathbf{X}_d\}$. That is, we set

$$\mathbf{X}_i = \mathbf{G}^{-1/2} \left((\mathbf{SA})\,[i,:] \right)^* \left((\mathbf{SA})\,[i,:] \right) \mathbf{G}^{-1/2} \tag{A.4}$$

so that $\mathbf{Y} = \sum_{i=1}^{d} \mathbf{X}_i$ satisfies $\mathbb{E}\mathbf{Y} = \mathbf{I}_n$. A union bound provides

$$\Pr\{E(t, \mathbf{S}) \text{ fails}\} \leq \Pr\{\lambda_{\max}(\mathbf{Y}) \geq 1 + t\} + \Pr\{1 - t \geq \lambda_{\min}(\mathbf{Y})\}.$$

Note that the claim of this proposition only invokes Theorem A.3.1 in the special case when t is between zero and one. Moreover, our particular choice of $\mathbf{Y}$ leads to $\mu_{\min} = \mu_{\max} = 1$. Given these restrictions, it can be shown that the larger of the two probability bounds in the theorem is that involving the term $\exp(t)/(1+t)^{(1+t)}$. Therefore we have

$$\Pr\{E(t, \mathbf{S}) \text{ fails}\} \leq 2n \left(\exp(t)/(1+t)^{(1+t)} \right)^{1/L}.$$

Next, we turn to finding the smallest possible L given this construction, so as to maximize $1/L$.

Let i be an arbitrary index in $[\![d]\!]$. By the definition of $\mathbf{S}$, the following must hold for some $k \in [\![m]\!]$:

$$\mathbf{X}_i = \frac{1}{d} \left(\frac{1}{q_k} \mathbf{G}^{-1/2} \mathbf{A}[k,:]^* \mathbf{A}[k,:] \mathbf{G}^{-1/2} \right).$$

Our next step is to use the fact that the trace of a rank-1 psd matrix is equal to its largest eigenvalue. Cycling the trace shows that

$$\lambda_{\max} \left(\mathbf{G}^{-1/2} \mathbf{A}[k,:]^* \mathbf{A}[k,:] \mathbf{G}^{-1/2} \right) = \mathbf{A}[k,:] \mathbf{G}^{-1} \mathbf{A}[k,:]^* = \ell_k(\mathbf{A}),$$

and hence

$$L = \frac{1}{d} \max_{j \in [\![m]\!]} \left\{ \frac{\ell_j(\mathbf{A})}{q_j} \right\}$$

is the smallest value that guarantees $\lambda_{\max}(\mathbf{X}_i) \leq L$.

To complete the proof we use the assumption that $\mathbf{A}$ is of full rank n to express the leverage score $\ell_j(\mathbf{A})$ as $\ell_j(\mathbf{A}) = n p_j(\mathbf{A})$. This shows that

$$L = \frac{n}{d} \max_{j \in [\![m]\!]} \frac{p_j(\mathbf{A})}{q_j},$$

and the proposition's claim follows from just a little algebra. □

Appendix B

Details on Least Squares and Optimization

This appendix covers a few distinct topics. Appendix B.1 proves a novel result relevant to sketch-and-precondition algorithms for saddle point problems, and connects this result to the idea of effective distortion. In Appendix B.2, we provide background from classical NLA on what it means to compute an "accurate" solution to a least squares problem (overdetermined or underdetermined). Appendix B.3 derives limiting solutions of saddle point problems as the regularization parameter tends to zero from above. These limiting solutions are important for treating saddle point problems as linear algebra problems even when their optimization formulations are ill-posed. Finally, Appendix B.4 gives background on kernel ridge regression and details a new approach to sketch-and-solve of regularized quadratics.

B.1 Quality of preconditioners

Here we consider preconditioners of the kind described in Section 3.3.2. These are obtained by sketching a tall $m \times n$ data matrix $\mathbf{A}$ in the embedding regime, factoring the sketch, and using the factorization to construct an orthogonalizer.

Proposition B.1.1 (Adaptation of Proposition 3.3.1)**.** *Consider a sketch* $\mathbf{A}_{\text{sk}} = \mathbf{SA}$ *and a matrix* $\mathbf{U}$ *whose columns are an orthonormal basis for* $\text{range}(\mathbf{A})$*. If* $\text{rank}(\mathbf{A}_{\text{sk}}) = \text{rank}(\mathbf{A})$ *and the columns of* $\mathbf{A}_{\text{sk}}\mathbf{M}$ *are an orthonormal basis for the range of* $\mathbf{A}_{\text{sk}}$*, then singular values of* $\mathbf{AM}$ *are the reciprocals of the singular values of* $\mathbf{SU}$*.*

Observe that this proposition is a linear algebraic result, i.e., there is no randomness. When applied to randomized algorithms, the randomness enters only via the construction of the sketch.

This result can be applied to problems with ridge regularization by working with augmented matrices in the vein of Section 3.3.2. In that context it is necessary to not only replace $(\mathbf{A}, \mathbf{A}_{\text{sk}})$ by $(\hat{\mathbf{A}}, \hat{\mathbf{A}}_{\text{sk}})$, but also to replace $\mathbf{S}$ by the augmented sketching operator $\hat{\mathbf{S}}$ that takes $\hat{\mathbf{A}}$ to $\hat{\mathbf{A}}_{\text{sk}}$. The augmented sketching operator in question was already visualized in Algorithm 2.

Our proof of Proposition B.1.1 requires finding an explicit expression for $\mathbf{M}$. Towards this end, we prove the following lemma.

Lemma B.1.2. *Suppose* $\mathbf{A}_{\text{sk}}$ *is a tall* $d \times n$ *matrix and that* $\mathbf{M}$ *is a full-column-rank matrix for which the columns of* $\mathbf{A}_{\text{sk}}\mathbf{M}$ *form an orthonormal basis for* $\text{range}(\mathbf{A}_{\text{sk}})$*. If* $\mathbf{B}$ *is a full-row-rank matrix for which* $\mathbf{A}_{\text{sk}} = \mathbf{A}_{\text{sk}}\mathbf{MB}$*, then we have* $\mathbf{M} = \mathbf{B}^{\dagger}$*.*

Proof of Lemma B.1.2. Let $k = \text{rank}(\mathbf{A}_{\text{sk}}) = \text{rank}(\mathbf{A}_{\text{sk}}\mathbf{M})$. Since the columns of $\mathbf{A}_{\text{sk}}\mathbf{M}$ are orthonormal we can infer that it has dimensions $d \times k$. Similarly, since $\mathbf{M}$ is full column-rank we can infer that it is $n \times k$. We prove that $\mathbf{B} = \mathbf{M}^{\dagger}$, which amounts to showing four properties:

1. $\mathbf{MBM} = \mathbf{M}$,
2. $\mathbf{BMB} = \mathbf{B}$,
3. $\mathbf{BM}$ is an orthogonal projector, and
4. $\mathbf{MB}$ is an orthogonal projector.

By the lemma's assumption we have the identity $\mathbf{A}_{\text{sk}} = \mathbf{A}_{\text{sk}}\mathbf{MB}$. Left multiply this expression through by $(\mathbf{A}_{\text{sk}}\mathbf{M})^*$ to see that

$$\mathbf{M}^*\mathbf{A}_{\text{sk}}^*\mathbf{A}_{\text{sk}} = \mathbf{B}. \tag{B.1}$$

Next, we right multiply both sides of (B.1) by $\mathbf{M}$ and use column orthonormality of $\mathbf{A}_{\text{sk}}\mathbf{M}$ to obtain $\mathbf{BM} = \mathbf{I}_k$ — this is sufficient to show the first three conditions for the pseudoinverse. Showing the fourth and final condition takes more work. For that we left multiply (B.1) by $\mathbf{M}$ so as to express

$$\mathbf{MM}^*\mathbf{A}_{\text{sk}}^*\mathbf{A}_{\text{sk}} = \mathbf{MB}.$$

Therefore our task is to show that $\mathbf{MM}^*\mathbf{A}_{\text{sk}}^*\mathbf{A}_{\text{sk}}$ is an orthogonal projector.

Consider the compact SVD $\mathbf{A}_{\text{sk}} = \mathbf{U}_{\text{sk}}\mathbf{\Sigma}_{\text{sk}}\mathbf{V}_{\text{sk}}$. Since $\mathbf{A}_{\text{sk}}$ is rank-k we have that $\mathbf{U}_{\text{sk}}$ has k columns and $\mathbf{\Sigma}$ is a $k \times k$ invertible matrix. Since the columns of $\mathbf{A}_{\text{sk}}\mathbf{M}$ form an orthonormal basis for the range of $\mathbf{A}_{\text{sk}}$, it must be that $\mathbf{A}_{\text{sk}}\mathbf{M} = \mathbf{U}_{\text{sk}}\mathbf{W}$ for some $k \times k$ orthogonal matrix $\mathbf{W}$. Furthermore, this orthogonal matrix can be expressed as $\mathbf{\Sigma}_{\text{sk}}\mathbf{V}_{\text{sk}}^*\mathbf{M} = \mathbf{W}$, which implies

$$\mathbf{V}_{\text{sk}}\mathbf{V}_{\text{sk}}^*\mathbf{M} = \mathbf{V}_{\text{sk}}\mathbf{\Sigma}_{\text{sk}}^{-1}\mathbf{W}. \tag{B.2}$$

We have reached a checkpoint in the proof. Our next task is to obtain an expression for $\mathbf{M}$ by simplifying (B.2).

Consider the subspaces $X = \operatorname{range} \mathbf{V}_{\mathrm{sk}}$, $Y = \ker \mathbf{A}_{\mathrm{sk}}$, and $Z = \operatorname{range} \mathbf{M}$, all contained in $\mathbb{R}^n$. We know that $Y \cap Z$ is trivial since $\operatorname{rank}(\mathbf{A}_{\mathrm{sk}}\mathbf{M}) = \operatorname{rank}(\mathbf{M})$. At the same time, since Y and Z are of dimensions $n-k$ and k respectively, it must be that $Z = Y^{\perp}$. This fact can be combined with $Y = X^{\perp}$ (from the fundamental theorem of linear algebra) to obtain $Z = X$, which in turn implies $\mathbf{V}_{\mathrm{sk}}\mathbf{V}_{\mathrm{sk}}^*\mathbf{M} = \mathbf{M}$. Therefore (B.2) simplifies to

$$\mathbf{M} = \mathbf{V}_{\mathrm{sk}}\mathbf{\Sigma}_{\mathrm{sk}}^{-1}\mathbf{W}.$$

This expression is precisely what we need; when the dust settles, it tells us that

$$\mathbf{M}\mathbf{M}^*\mathbf{A}_{\mathrm{sk}}^*\mathbf{A}_{\mathrm{sk}} = \mathbf{V}_{\mathrm{sk}}\mathbf{V}_{\mathrm{sk}}^*.$$

□

Proof of Proposition B.1.1. Let $k = \operatorname{rank}(\mathbf{A})$. It suffices to prove the statement where $\mathbf{U}$ is the $m \times k$ matrix containing the left singular vectors of $\mathbf{A}$. Our proof involves working with the compact SVD $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*$, where $\mathbf{V}$ is $n \times k$ and $\mathbf{\Sigma}$ is invertible. Noting that $\mathbf{A}_{\mathrm{sk}} = \mathbf{S}\mathbf{U}\mathbf{\Sigma}\mathbf{V}^*$ holds by definition of $\mathbf{A}_{\mathrm{sk}}$, we can replace $\mathbf{S}\mathbf{U}$ by its economic QR factorization $\mathbf{S}\mathbf{U} = \mathbf{Q}\mathbf{R}$ to see

$$\mathbf{A}_{\mathrm{sk}} = \mathbf{Q}\mathbf{R}\mathbf{\Sigma}\mathbf{V}^*. \tag{B.3}$$

Furthermore, since $\operatorname{rank}(\mathbf{A}_{\mathrm{sk}}) = k$ it must be that $\operatorname{rank}(\mathbf{S}\mathbf{U}) = k$. This tells us that $\mathbf{R}$ is invertible and that $\mathbf{Q}$ provides an orthonormal basis for the range of $\mathbf{A}_{\mathrm{sk}}$.

By assumption on $\mathbf{M}$, the matrix $\mathbf{A}_{\mathrm{sk}}\mathbf{M}$ is *also* an orthonormal basis for the range of $\mathbf{A}_{\mathrm{sk}}$. Therefore there exists a $k \times k$ orthogonal matrix $\mathbf{P}$ where $\mathbf{Q}\mathbf{P} = \mathbf{A}_{\mathrm{sk}}\mathbf{M}$. We can rewrite (B.3) as

$$\mathbf{A}_{\mathrm{sk}} = (\mathbf{Q}\mathbf{P})\left(\mathbf{P}^*\mathbf{R}\mathbf{\Sigma}\mathbf{V}^*\right).$$

Since the left factor in the above display is simply $\mathbf{A}_{\mathrm{sk}}\mathbf{M}$, we have

$$\mathbf{A}_{\mathrm{sk}} = \mathbf{A}_{\mathrm{sk}}\mathbf{M}\left(\mathbf{P}^*\mathbf{R}\mathbf{\Sigma}\mathbf{V}^*\right). \tag{B.4}$$

The next step is to abbreviate $\mathbf{B} = \mathbf{P}^*\mathbf{R}\mathbf{\Sigma}\mathbf{V}^*$ and apply Lemma B.1.2 to infer that $\mathbf{B} = \mathbf{M}^{\dagger}$. Invoking the column-orthonormality of $\mathbf{V}$ and invertibility of $(\mathbf{\Sigma}, \mathbf{R}, \mathbf{P})$ we further have $\mathbf{B}^{\dagger} = \mathbf{M} = \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{R}^{-1}\mathbf{P}$. Plug in this expression for $\mathbf{M}$ to see that

$$\mathbf{A}\mathbf{M} = (\mathbf{U}\mathbf{\Sigma}\mathbf{V}^*)\left(\mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{R}^{-1}\mathbf{P}\right) = \mathbf{U}\mathbf{R}^{-1}\mathbf{P}. \tag{B.5}$$

The proof is completed by noting that the singular values of $\mathbf{R}^{-1}$ are the reciprocals of the singular values of $\mathbf{Q}\mathbf{R} = \mathbf{S}\mathbf{U}$. □

B.1.1 Effective distortion in sketch-and-precondition

Recall from Appendix A.1 that if the columns of $\mathbf{U}$ are an orthonormal basis for a linear subspace L, then the restricted condition number of $\mathbf{S}$ on L is

$$\operatorname{cond}(\mathbf{S}; L) = \operatorname{cond}(\mathbf{S}\mathbf{U}).$$

This identity combines with Proposition B.1.1 to make for a remarkable fact. Namely, if $L = \operatorname{range}(\mathbf{A})$ and $\mathbf{M}$ is an orthogonalizer of a sketch $\mathbf{S}\mathbf{A}$, then

$$\operatorname{cond}(\mathbf{S}; L) = \operatorname{cond}(\mathbf{A}\mathbf{M}). \tag{B.6}$$

Let us contextualize this fact algorithmically.

> If $\mathbf{A}$ is an $m \times n$ matrix ($m \gg n$) in a saddle point problem, and if that problem is approached by the sketch-and-precondition methodology from Section 3.2.2, then the condition number of the preconditioned matrix handed to the iterative solver is equal to the restricted condition number of $\mathbf{S}$ on range($\mathbf{A}$).

But we can take this one step further. By invoking Proposition A.1.1 and applying (B.6), we obtain the following expression for the effective distortion of $\mathbf{S}$ for L:

$$\mathscr{D}_{\mathrm{e}}(\mathbf{S}; L) = \frac{\mathrm{cond}(\mathbf{AM}) - 1}{\mathrm{cond}(\mathbf{AM}) + 1}. \tag{B.7}$$

Alarm bells should be going off in some readers' heads. The right-hand-side of (B.7) is none other than the convergence rate of LSQR (or CGLS) for a least squares problem with data matrix $\mathbf{AM}$! This shows a deep connection between our proposed concept of effective distortion and the venerated sketch-and-precondition paradigm in RandNLA.

B.2 Basic error analysis for least squares problems

When solving a computational problem numerically it is inevitable that the computed solutions deviate from the problem's exact solution. This is a simple consequence of working in finite-precision arithmetic, and it remains true even when using very reliable algorithms. Furthermore, for large-scale computations it is often of interest to trade off computational complexity with solution accuracy; this has led to algorithms that produce approximate solutions even when run in exact arithmetic.

These facts were encountered in the earliest days of NLA. Their consequence in applications has motivated the development of a vast literature on quantifying and bounding the error of approximate solutions to computational problems. Since several of the randomized algorithms from Section 3.2 purport to solve problems to any desired accuracy, it is prudent for us to summarize key points from this vast literature here. The material from Appendices B.2.1 to B.2.4 is typically covered in an introductory course on numerical analysis. Appendix B.2.5 mentions important topics which might not be covered in such a course.

Remark B.2.1. We have focused this appendix strongly on basic least squares problems (overdetermined and underdetermined) to keep it a reasonable length.

B.2.1 Concepts: forward and backward error

The *forward error* of an approximate solution to a computational problem is its distance to the problem's exact solution. Forward error is easy to interpret, but it is not without its limitations. First, it can rarely be computed in practice, since it is presumed that we do not have access to the problem's exact solution. This means that substantial effort is needed to approximate or bound forward error in different contexts. Second, even if one algorithm's behavior with respect to forward error has been analyzed, it may not be feasible to repurpose the analysis for another algorithm. These shortcomings motivate the ideas of *backward error* and *sensitivity analysis*, wherein one asks the following questions, respectively.

- How much do we need to perturb the problem data so that the computed solution exactly solves the perturbed problem?
- How does a small perturbation to a given problem change that problem's exact solution?

The connection between the two concepts is clear: any bound on backward error can be combined with sensitivity analysis to obtain an estimate of forward error. The idea of sensitivity analysis is especially powerful since it is agnostic to the source of the problem's perturbation; the perturbations might be due to rounding errors from finite-precision arithmetic, uncertainty in data (as might arise from experimental observations), or deliberate choices to only compute approximate solutions. In any of these cases one can combine knowledge of an algorithm's backward-error guarantees to obtain forward error estimates.

This reasoning can be carried further to arrive at two major benefits of the "backward error plus sensitivity analysis" approach.

- A large portion of algorithm-specific error analysis can be accomplished purely by understanding the algorithm's behavior with respect to backward error.
- For many problems one can cheaply compute upper bounds on a solution's backward error *at runtime*.

The combination of backward error and sensitivity analysis can therefore be used to establish *a priori* guarantees on algorithm numerical behavior and *a posteriori* guarantees on the quality of an approximate solution. However, we do note that sensitivity analysis results require knowledge of problem data that may not be available, such as extreme singular values of a data matrix in a least squares problem. Therefore it is still difficult to compute forward error bounds at runtime.

B.2.2 Basic sensitivity analysis for unregularized least squares problems

Here we paraphrase facts from [GV13, §5.3 and §5.6] on sensitivity analysis of basic least squares problems. Our restatements adopt the notation we used for saddle point problems, wherein both overdetermined and underdetermined problems involve a tall $m \times n$ matrix $\mathbf{A}$. The overdetermined problem induced by $\mathbf{A}$ and an m-vector $\boldsymbol{b}$ is

$$\min_{\boldsymbol{x} \in \mathbb{R}^n} \|\mathbf{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2,$$

while the underdetermined problem induced by $\mathbf{A}$ and an n-vector $\boldsymbol{c}$ is

$$\min_{\boldsymbol{y} \in \mathbb{R}^m} \left\{ \|\boldsymbol{y}\|_2^2 \,:\, \mathbf{A}^*\boldsymbol{y} = \boldsymbol{c} \right\}.$$

In the following theorem statements, the reader should bear in mind that a perturbation $\delta\mathbf{A}$ can only satisfy $\|\delta\mathbf{A}\|_2 < \sigma_n(\mathbf{A})$ if $\sigma_n(\mathbf{A})$ is positive. Therefore these theorem statements only apply when $\mathbf{A}$ is full-rank.

Theorem B.2.2. *Suppose $\boldsymbol{b}$ is neither in the range of $\mathbf{A}$ nor the kernel of $\mathbf{A}^*$, and let $\boldsymbol{x} = \mathbf{A}^\dagger \boldsymbol{b}$ be the optimal solution of the overdetermined least squares problem with data $(\mathbf{A}, \boldsymbol{b})$. Consider perturbations $\delta\boldsymbol{b}$ and $\delta\mathbf{A}$ where $\|\delta\mathbf{A}\|_2 < \sigma_n(\mathbf{A})$. Define*

$$\epsilon = \max\left\{ \frac{\|\delta\mathbf{A}\|_2}{\|\mathbf{A}\|_2}, \; \frac{\|\delta\boldsymbol{b}\|_2}{\|\boldsymbol{b}\|_2} \right\} \tag{B.8}$$

together with some auxiliary quantities

$$\sin\theta = \frac{\|\boldsymbol{b} - \mathbf{A}\boldsymbol{x}\|_2}{\|\boldsymbol{b}\|_2} \quad \textit{and} \quad \nu = \frac{\|\mathbf{A}\boldsymbol{x}\|_2}{\sigma_n(\mathbf{A})\|\boldsymbol{x}\|_2}. \tag{B.9}$$

The perturbation $\delta\boldsymbol{x}$ *necessary for* $\boldsymbol{x} + \delta\boldsymbol{x}$ *to solve the least squares problem with data* $(\mathbf{A} + \delta\mathbf{A}, \boldsymbol{b} + \delta\boldsymbol{b})$ *satisfies*

$$\frac{\|\delta\boldsymbol{x}\|_2}{\|\boldsymbol{x}\|_2} \leq \epsilon \left\{ \frac{\nu}{\cos\theta} + \kappa(\mathbf{A})(1 + \nu\tan\theta) \right\} + O(\epsilon^2). \tag{B.10}$$

Theorem B.2.2 restates part of [GV13, Theorem 5.3.1]; following the proof of this result, the source material presents some simplified estimates for these bounds. The first step in producing the simplified estimate is to note that $\nu \leq \kappa(\mathbf{A})$ holds for all nonzero $\boldsymbol{x}$. Then, under the modest geometric assumption that θ is bounded away from $\pi/2$, (B.10) suggests that

$$\frac{\|\delta\boldsymbol{x}\|_2}{\|\boldsymbol{x}\|_2} \lesssim \epsilon \left\{ \kappa(\mathbf{A}) + \frac{\|\boldsymbol{b} - \mathbf{A}\boldsymbol{x}\|_2}{\|\boldsymbol{b}\|_2} \kappa(\mathbf{A})^2 \right\}. \tag{B.11}$$

The significance of this bound is that it shows the dependence of $\|\delta\boldsymbol{x}\|_2$ on the *square* of the condition number of $\mathbf{A}$. This dependence is a fundamental obstacle to solving least squares problems to a high degree of accuracy when measured by forward error. The situation is different if we try to bound the perturbation $\|\mathbf{A}(\delta\boldsymbol{x})\|$. We provide the following result (which completes the restatement of [GV13, Theorem 5.3.1]) as a step towards explaining why.

Theorem B.2.3. *Under the hypothesis and notation of Theorem B.2.2, we have*

$$\frac{\|\mathbf{A}(\delta\boldsymbol{x})\|_2}{\|\boldsymbol{b} - \mathbf{A}\boldsymbol{x}\|_2} \leq \epsilon \left\{ \frac{1}{\sin\theta} + \kappa(\mathbf{A}) \left(\frac{1}{\nu\tan\theta} + 1 \right) \right\} + O(\epsilon^2). \tag{B.12}$$

Theorem B.2.3 can be seen as a sensitivity analysis result for a very specific class of dual saddle point problems. Specifically, since we have assumed that $\mathbf{A}$ is full rank, $\boldsymbol{y}$ solves the dual problem if and only if $\boldsymbol{y} = \boldsymbol{b} - \mathbf{A}\boldsymbol{x}$ where $\boldsymbol{x}$ solves the primal problem. In the same vein, if $\boldsymbol{x} + \delta\boldsymbol{x}$ solves a perturbed primal problem and we set $\delta\boldsymbol{y} = -\mathbf{A}(\delta\boldsymbol{x})$, then $\boldsymbol{y} + \delta\boldsymbol{y}$ solves the perturbed dual problem.

As with the bound for $\delta\boldsymbol{x}$, (B.12) can be estimated under mild geometric assumptions; [GV13, pg. 267] points out that if θ is sufficiently bounded away from 0 and $\pi/2$, then we should have

$$\frac{\|\delta\boldsymbol{y}\|_2}{\|\boldsymbol{y}\|_2} \lesssim \epsilon\,\kappa(\mathbf{A}). \tag{B.13}$$

This shows there is more hope for solving dual saddle point problems to a high degree of forward error accuracy, at least by comparison to primal saddle point problems. Indeed, the following adaptation of [GV13, Theorem 5.6.1] provides an even more favorable sensitivity analysis result for underdetermined least squares.

Theorem B.2.4. *Let* $\boldsymbol{y} = (\mathbf{A}^*)^\dagger \boldsymbol{c}$ *solve the underdetermined least squares problem with data* $(\mathbf{A}, \boldsymbol{c})$ *for a nonzero vector* $\boldsymbol{c}$*. Consider perturbations* $\delta\boldsymbol{c}$ *and* $\delta\mathbf{A}$ *where*

$$\epsilon = \max\left\{ \frac{\|\delta\boldsymbol{c}\|_2}{\|\boldsymbol{c}\|_2}, \frac{\|\delta\mathbf{A}\|_2}{\|\mathbf{A}\|_2} \right\} < \sigma_n(\mathbf{A}).$$

The perturbation $\delta\boldsymbol{y}$ *needed for* $\boldsymbol{y}+\delta\boldsymbol{y}$ *to solve the underdetermined least squares problem with data* $(\mathbf{A}+\delta\mathbf{A}, \boldsymbol{c}+\delta\boldsymbol{c})$ *satisfies*

$$\frac{\|\delta\boldsymbol{y}\|_2}{\|\boldsymbol{y}\|_2} \leq 3\,\epsilon\,\kappa(\mathbf{A}) + O(\epsilon^2). \tag{B.14}$$

B.2.3 Sharper sensitivity analysis for overdetermined problems

The analysis results in Appendix B.2.2 have notable limitations: they hide constants in $O(\epsilon^2)$ terms. Luckily there are a wealth of more precise results in the literature that work with different notions of error. One good example along these lines for overdetermined least squares is given in [Ips09, Fact 5.14], which obtains a relative error bound normalized by the solution of the *perturbed problem* rather than the original problem. We paraphrase this fact below.

Theorem B.2.5. *Consider an overdetermined least squares problem with data* $(\mathbf{A}_o, \boldsymbol{b}_o)$ *that is solved by* $\boldsymbol{x}_o = (\mathbf{A}_o)^\dagger(\boldsymbol{b}_o)$*; consider also perturbed problem data* $\mathbf{A} = \mathbf{A}_o + \delta\mathbf{A}_o$ *and* $\boldsymbol{b} = \boldsymbol{b}_o + \delta\boldsymbol{b}_o$ *together with solution* $\boldsymbol{x} = \mathbf{A}^\dagger\boldsymbol{b}$*. If we have* $\operatorname{rank}(\mathbf{A}_o) = \operatorname{rank}(\mathbf{A}) = n$ *and define*

$$\epsilon_A = \frac{\|\delta\mathbf{A}_o\|_2}{\|\mathbf{A}_o\|_2} \quad \textit{and} \quad \epsilon_b = \frac{\|\delta\boldsymbol{b}_o\|}{\|\mathbf{A}_o\|_2\|\boldsymbol{x}\|_2},$$

then we have

$$\frac{\|\boldsymbol{x}_o - \boldsymbol{x}\|_2}{\|\boldsymbol{x}\|_2} \leq (\epsilon_A + \epsilon_b)\kappa(\mathbf{A}_o) + \epsilon_A \frac{[\kappa(\mathbf{A}_o)]^2\,\|\boldsymbol{y}\|_2}{\|\mathbf{A}_o\|_2\|\boldsymbol{x}\|_2} \tag{B.15}$$

for $\boldsymbol{y} = \boldsymbol{b} - \mathbf{A}\boldsymbol{x}$.

Bounds of similar character are given for $\boldsymbol{y}$ on [Ips09, pg. 101]. These bounds are useful for understanding how solutions exhibit different sensitivity for perturbations to the data matrix compared to perturbations to the right-hand-side. Even better bounds can be obtained by assuming *structured perturbations*. For example, if $\operatorname{range}(\mathbf{A}_o) = \operatorname{range}(\mathbf{A})$, then the sensitivity of the overdetermined least squares solution depends only linearly on $\kappa(\mathbf{A}_o)$ [Ips09, Exercise 5.1].

Our discussion of sensitivity analysis has only considered *normwise* perturbations to the problem data. More informative bounds can be had by considering *componentwise* perturbations. For example, one can measure a perturbation of an initial matrix $\mathbf{A}$ by the smallest α for which $|\delta A_{ij}| \leq \alpha|A_{ij}|$ for all i, j. We refer the reader to [Hig02, §20.1] for a componentwise sensitivity analysis result on overdetermined least squares.

B.2.4 Simple constructions to bound backward error

Here we describe two methods for constructing perturbations to problem data that render an approximate solution exact. By computing the norms of these perturbations, we can obtain upper bounds on (normwise) backward error. Such bounds are useful as termination criteria for iterative solvers.

The notation here matches that of Theorem B.2.5. That is, we say our original least squares problem has data $(\mathbf{A}_o, \boldsymbol{b}_o)$ and that $\boldsymbol{x}$ is an *approximate* solution to this problem.

Remark B.2.6. For discussion on *componentwise* backward error bounds for overdetermined least squares we again refer the reader to [Hig02, §20.1].

Inconsistent overdetermined problems

Letting $\boldsymbol{r} = \boldsymbol{b}_o - \mathbf{A}_o\boldsymbol{x}$, we define

$$\delta\mathbf{A}_o = \frac{\boldsymbol{r}\boldsymbol{r}^*\mathbf{A}_o}{\|\boldsymbol{r}\|_2^2} \quad \text{and} \quad \mathbf{A} = \mathbf{A}_o + \delta\mathbf{A}_o, \tag{B.16}$$

and subsequently

$$\delta\boldsymbol{b}_o = -(\delta\mathbf{A}_o)\boldsymbol{x} \quad \text{and} \quad \boldsymbol{b} = \boldsymbol{b}_o + \delta\boldsymbol{b}_o. \tag{B.17}$$

Some simple algebra shows that $\boldsymbol{x}$ satisfies the normal equations

$$\mathbf{A}^*\left(\boldsymbol{b} - \mathbf{A}\boldsymbol{x}\right) = \mathbf{0},$$

therefore it solves the overdetermined least squares problem with data $(\mathbf{A}, \boldsymbol{b})$.

This construction was first given in [Ste77, Theorem 3.2]. It is especially nice since the perturbation is rank-1, and so its spectral norm

$$\|\delta\mathbf{A}_o\|_2 = \frac{\|\mathbf{A}_o^*\boldsymbol{r}\|_2}{\|\boldsymbol{r}\|_2}$$

is easily computed at runtime by an iterative solver for overdetermined least squares. Furthermore, if the iterative solver in question is LSQR, and if we assume exact arithmetic, then the perturbation will satisfy $\delta\mathbf{A}_o\boldsymbol{x} = \mathbf{0}$ [PS82, §6.2]. Therefore the vector $\delta\boldsymbol{b}_o$ in (B.17) is zero when running LSQR (or any equivalent method) in exact arithmetic.

Consistent overdetermined problems: a word of warning

The perturbations given in (B.16) – (B.17) are not suitable for least squares problems where the optimal residual, $(\mathbf{I} - \mathbf{A}_o\mathbf{A}_o^\dagger)\boldsymbol{b}_o$, is zero or nearly zero. In these situations one should use a perturbation designed for consistent linear systems. We describe such a construction here based on termination criteria used in LSQR.

Let $\delta\boldsymbol{b}_o$ be an arbitrary m-vector. Suppose we set $\delta\mathbf{A}_o$ as a function of $\delta\boldsymbol{b}_o$ in the following way:

$$\delta\mathbf{A}_o = \frac{\left(\delta\boldsymbol{b}_o + \boldsymbol{b}_o - \mathbf{A}_o\boldsymbol{x}\right)\boldsymbol{x}^*}{\|\boldsymbol{x}\|_2^2}.$$

It can be seen that $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$ upon taking $\boldsymbol{b} = \boldsymbol{b}_o + \delta\boldsymbol{b}_o$ and $\mathbf{A} = \mathbf{A}_o + \delta\mathbf{A}_o$, and so $\boldsymbol{x}$ trivially solves the perturbed least squares problem with data $(\mathbf{A}, \boldsymbol{b})$.

One can obtain many backward-error constructions by considering different choices for $\delta\boldsymbol{b}_o$ as a function of $\boldsymbol{x}$, the problem data $(\mathbf{A}_o, \boldsymbol{b}_o)$, and desired error tolerances. The construction for LSQR considers two tolerance parameters $\epsilon_A, \epsilon_b \in [0, 1)$, and sets $\delta\boldsymbol{b}_o$ as follows [PS82, §6.1]:

$$\delta\boldsymbol{b}_o = \left(\frac{\epsilon_b\|\boldsymbol{b}_o\|_2}{\epsilon_b\|\boldsymbol{b}_o\|_2 + \epsilon_A\|\mathbf{A}_o\|\|\boldsymbol{x}\|_2}\right)\left(\mathbf{A}_o\boldsymbol{x} - \boldsymbol{b}_o\right). \tag{B.18}$$

The parameters ϵ_A and ϵ_b indicate the (relative) sizes of perturbations to $(\mathbf{A}_o, \boldsymbol{b}_o)$ that a user deems allowable. The authors of [PS82] suggest that "allowable" be based on the extent to which $(\mathbf{A}_o, \boldsymbol{b}_o)$ are not known exactly in applications.

It is natural to want to reduce the two tolerance parameters (ϵ_A, ϵ_b) to a single tolerance parameter. For example, one might take $\epsilon_A = \epsilon_b$. Unfortunately, our

experience is that taking $\epsilon_A = \epsilon_b$ can produce unreliable algorithm behavior for overdetermined problems. Therefore we recommend that one sets $\epsilon_A = 0$ if one wants to think only in terms of a single tolerance for consistent overdetermined problems. While this may seem like a blunt solution, it ensures that $\delta\mathbf{A}_o = \mathbf{0}$, which is useful in applying Theorem B.2.5. If setting $\epsilon_A = 0$ still feels too extreme then one might consider setting $\epsilon_A = (\epsilon_b)^2 \ll \epsilon_b$.

Remark B.2.7. As a minor detail, we point out that the norm of $\mathbf{A}_o$ in (B.18) is deliberately ambiguous. While the spectral norm would probably be most natural, the formal LSQR algorithm replaces $\|\mathbf{A}_o\|$ by an *estimate* of $\|\mathbf{A}_o\|_{\mathrm{F}}$ that monotonically increases from one iteration to the next; see [PS82, §5.3].

B.2.5 More advanced concepts

Some of the earliest work on backward-error analysis for solutions to linear systems focused on componentwise backward error for direct methods [OP64]. A principal shortcoming of componentwise error metrics is that they are expensive to compute, especially as stopping criteria for iterative solvers. [ADR92] investigates metrics for componentwise backward error suitable for iterative solvers.

The "backward error plus sensitivity analysis" approach may overestimate forward error. Alternative estimates are available for some Krylov subspace methods such as PCG, wherein one uses algorithm-specific recurrences to estimate forward error in the Euclidean norm or the norm induced by $\mathbf{A}_\mu := [\mathbf{A}; \sqrt{\mu}]$. See, for example, [AK01; ST02; ST05]. These error bounds are more accurate when used with a good preconditioner, which we can generally expect to have when using the randomized algorithms described herein.

It is not easy to apply sensitivity analysis results to compute forward error bounds at runtime. A primary obstacle in this regard is the need to have accurate estimates for the extreme singular values of $\mathbf{A}_o$ or the perturbation $\mathbf{A}$ (depending on the sensitivity analysis result in question). On this topic we note that if $\mathbf{M}$ is an SVD-based preconditioner then we will have computed the singular values and right singular vectors of a sketch $\mathbf{SA}_o$. Those singular values can be used as approximations to the reciprocals of the singular values of $\mathbf{A}_o$. It is conceivable that more accurate approximations could be obtained by applying iterative preconditioned eigenvalue estimation methods for $(\mathbf{A}_o)^*\mathbf{A}_o$. Such iterative methods typically require initialization with an approximate eigenvector. On this front one can use the leading (resp. trailing) left singular vector of $\mathbf{M}$ to approximate the trailing (resp. leading) right singular vector of $\mathbf{A}_o$. One should not expect too much of such estimates, however.[1]

Finally, we note that some Krylov-subspace iterative methods can estimate condition numbers. For example, when LSQR is applied to a problem with data matrix $\mathbf{L}$, it can estimate the Frobenius condition number $\|\mathbf{L}\|_{\mathrm{F}}\|\mathbf{L}^\dagger\|_{\mathrm{F}}$. Bear in mind that in our context we call LSQR with the preconditioned augmented data matrix, $\mathbf{L} = \mathbf{A}_\mu\mathbf{M}$. It would be useful to embed estimators for *componentwise condition numbers* (which are known to be computable in polynomial time [Dem92]) into Krylov subspace solvers.

[1] Any "cheap" method for estimating the smallest singular value even of *triangular* matrices can return substantial overestimates and underestimates [DDM01].

B.3 Ill-posed saddle point problems

Our saddle point formulations of least squares problems can be problematic when $\mathbf{A}$ is rank-deficient and μ is zero, in which case our problems can actually be infeasible or unbounded below. This appendix uses a limiting analysis to define *canonical solutions* to saddle point problems in these settings.

We begin by recalling

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} \|\mathbf{A}\boldsymbol{x}-\boldsymbol{b}\|_2^2+\mu\|\boldsymbol{x}\|_2^2+2\boldsymbol{c}^*\boldsymbol{x}, \qquad ((3.2), \text{revisited})$$

$$\min_{\boldsymbol{y}\in\mathbb{R}^m} \|\mathbf{A}^*\boldsymbol{y}-\boldsymbol{c}\|_2^2+\mu\|\boldsymbol{y}-\boldsymbol{b}\|_2^2, \qquad ((3.3), \text{revisited})$$

$$\text{and} \quad \min_{\boldsymbol{y}\in\mathbb{R}^m} \{\|\boldsymbol{y}-\boldsymbol{b}\|_2^2 \,:\, \mathbf{A}^*\boldsymbol{y}=\boldsymbol{c}\}. \qquad ((3.4), \text{revisited})$$

We also note the following form of solutions to (3.3), when μ is positive

$$\boldsymbol{y}(\mu)=\left(\mathbf{A}\mathbf{A}^*+\mu\mathbf{I}\right)^{-1}\left(\mathbf{A}\boldsymbol{c}+\mu\boldsymbol{b}\right). \tag{B.19}$$

Proposition B.3.1. *For any tall $m\times n$ matrix $\mathbf{A}$, any m-vector $\boldsymbol{b}$, and any n-vector $\boldsymbol{c}$, we have*

$$\lim_{\mu\downarrow 0}\boldsymbol{y}(\mu)=(\mathbf{A}^*)^\dagger\boldsymbol{c}+(\mathbf{I}-\mathbf{A}\mathbf{A}^\dagger)\boldsymbol{b}. \tag{B.20}$$

Proof. Let $k=\operatorname{rank}(\mathbf{A})$. If $k=0$ then the claim is trivial since (B.19) reduces to $\boldsymbol{y}(\mu)=\boldsymbol{b}$ for all $\mu>0$. Henceforth, we assume $k>1$. To establish the claim, consider how the compact SVD $\mathbf{A}=\mathbf{U}\mathbf{\Sigma}\mathbf{V}^*$ lets us express

$$\mathbf{A}\mathbf{A}^*+\mu\mathbf{I}_m=\mathbf{H}_\mu+\mathbf{G}_\mu$$

in terms of the Hermitian matrices

$$\mathbf{H}_\mu=\mathbf{U}\left(\mathbf{\Sigma}^2+\mu\mathbf{I}_k\right)\mathbf{U}^*$$
$$\text{and} \quad \mathbf{G}_\mu=\mu(\mathbf{I}_m-\mathbf{U}\mathbf{U}^*).$$

Since $\mathbf{H}_\mu\mathbf{G}_\mu=\mathbf{G}_\mu\mathbf{H}_\mu=\mathbf{0}$, the following identity holds for all positive μ:

$$\left(\mathbf{A}\mathbf{A}^*+\mu\mathbf{I}\right)^{-1}=\mathbf{H}_\mu^\dagger+\mathbf{G}_\mu^\dagger.$$

Furthermore, by expressing

$$\mathbf{H}_\mu^\dagger\mathbf{A}\boldsymbol{c}=\mathbf{U}\left(\mathbf{\Sigma}^2+\mu\mathbf{I}_k\right)^{-1}\mathbf{\Sigma}\mathbf{V}^*\boldsymbol{c}$$
$$\mathbf{G}_\mu^\dagger\mathbf{A}\boldsymbol{c}=\mathbf{0}$$
$$\mu\mathbf{H}_\mu^\dagger\boldsymbol{b}=\mathbf{U}\left(\mathbf{\Sigma}^2/\mu+\mathbf{I}_k\right)^{-1}\mathbf{U}^*\boldsymbol{b}$$
$$\mu\mathbf{G}_\mu^\dagger\boldsymbol{b}=(\mathbf{I}_m-\mathbf{U}\mathbf{U}^*)\boldsymbol{b}$$

we find that

$$\begin{aligned}\boldsymbol{y}(\mu)&=\mathbf{H}_\mu^\dagger(\mathbf{A}\boldsymbol{c}+\mu\boldsymbol{b})+\mathbf{G}_\mu^\dagger(\mathbf{A}\boldsymbol{c}+\mu\boldsymbol{b})\\&\to\mathbf{U}\mathbf{\Sigma}^{-1}\mathbf{V}^*\boldsymbol{c}+(\mathbf{I}_m-\mathbf{U}\mathbf{U}^*)\boldsymbol{b}.\end{aligned}$$

This is equivalent to the desired claim since $(\mathbf{A}^*)^\dagger=\mathbf{U}\mathbf{\Sigma}^{-1}\mathbf{V}^*$ and $\mathbf{A}\mathbf{A}^\dagger=\mathbf{U}\mathbf{U}^*$. □

In light of the above proposition, we take $\boldsymbol{y}(0) = (\mathbf{A}^*)^\dagger \boldsymbol{c} + (\mathbf{I} - \mathbf{A}\mathbf{A}^\dagger)\boldsymbol{b}$ as our canonical solution to the dual problem when $\mu = 0$.

Now we let $\boldsymbol{x}(\mu)$ denote the solution to (3.2) parameterized by $\mu > 0$. It is clear that this is given by

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

It is easy to show that if $\boldsymbol{c}$ is not orthogonal to the kernel of $\mathbf{A}$, then the norms $\|\boldsymbol{x}(\mu)\|$ will diverge to infinity as μ tends to zero. However, if $\boldsymbol{c}$ *is* orthogonal to the kernel of $\mathbf{A}$, then we have

$$\lim_{\mu \downarrow 0} \boldsymbol{x}(\mu) = (\mathbf{A}^*\mathbf{A})^\dagger(\mathbf{A}^*\boldsymbol{b} - \boldsymbol{c}) =: \boldsymbol{x}(0). \tag{B.21}$$

We actually take the limit above as our canonical solution to the primal problem (3.2) regardless of whether or not $\boldsymbol{c}$ is orthogonal to the kernel of $\mathbf{A}$. Our reasons for this are two-fold. First, the values $\boldsymbol{x}(0), \boldsymbol{y}(0)$ given above are unchanged when $\boldsymbol{c}$ is replaced by its orthogonal projection onto range of $\mathbf{A}^*$. Second, the value $\boldsymbol{y}(0)$ is always the limiting solution to the dual problem. Meanwhile, the proposed value for $\boldsymbol{x}(0)$ relates to $\boldsymbol{y}(0)$ by $\boldsymbol{y}(0) = \boldsymbol{b} - \mathbf{A}\boldsymbol{x}(0)$.

B.4 Minimizing regularized quadratics

Appendix B.4.1 provides a brief introduction to kernel ridge regression (KRR). It covers the finite-dimensional linear algebraic formulation and the Hilbert space formulation of this regression model, and it explains how ridge regression can be understood in the KRR framework. Appendix B.4.2 presents a novel preconditioner-generation procedure for solving a *sketch* of the regularized quadratic minimization problem (3.1).

B.4.1 A primer on kernel ridge regression

Kernel ridge regression (KRR) is a type of nonparametric regression for learning real-valued nonlinear functions $f : \mathcal{X} \to \mathbb{R}$. It can be formulated as a linear algebra problem as follows: we are given $\lambda > 0$, an $m \times m$ psd "kernel matrix" $\mathbf{K}$, and a vector of observations $\boldsymbol{h}$ in $\mathbb{R}^m$; we want to solve

$$\operatorname*{argmin}_{\boldsymbol{\alpha} \in \mathbb{R}^m} \tfrac{1}{m}\|\mathbf{K}\boldsymbol{\alpha} - \boldsymbol{h}\|_2^2 + \lambda\, \boldsymbol{\alpha}^*\mathbf{K}\boldsymbol{\alpha}. \tag{B.22}$$

Equivalently, we want to solve the *KRR normal equations* $(\mathbf{K} + m\lambda\,\mathbf{I})\boldsymbol{\alpha} = \boldsymbol{h}$. The normal equations formulation makes it clear that KRR is an instance of (3.1).

A standard library for RandNLA would be well-served to not dwell on how $\mathbf{K}$ is defined; it should instead only focus on how $\mathbf{K}$ can be accessed. However, strictly speaking, (B.22) only encodes a KRR problem when the entries of $\mathbf{K}$ are given by pairwise evaluations of a suitable two-argument *kernel function* on some datapoints $\{\boldsymbol{x}_i\}_{i=1}^m \subset \mathcal{X}$. Letting $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ denote this kernel function, the user will take $\boldsymbol{\alpha}$ that approximately solves (B.22) to define the learned model $g(\boldsymbol{z}) = \sum_{i=1}^m \alpha_i k(\boldsymbol{x}_i, \boldsymbol{z})$.

A more technical description

The kernel function k induces a *reproducing kernel Hilbert space*, $\mathcal{H}$, of real-valued functions on $\mathcal{X}$. This space is (up to closure) equal to the set of real-linear combinations of functions $\boldsymbol{y} \mapsto k^{\boldsymbol{u}}(\boldsymbol{y}) := k(\boldsymbol{y}, \boldsymbol{u})$ parameterized by $\boldsymbol{u} \in \mathcal{X}$. Additionally, if the function

$$\boldsymbol{y} \mapsto f(\boldsymbol{y}) = \sum_{i=1}^{m} \alpha_i k(\boldsymbol{y}, \boldsymbol{x}_i)$$

is parameterized by $\boldsymbol{\alpha} \in \mathbb{R}^m$ and $\{\boldsymbol{x}_i\}_{i=1}^m \subset \mathcal{X}$, then its squared norm is given by

$$\|f\|_{\mathcal{H}}^2 = \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j k(\boldsymbol{x}_i, \boldsymbol{x}_j).$$

Using the kernel matrix $\mathbf{K}$ with entries $K_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$, we can express that squared norm as $\|f\|_{\mathcal{H}}^2 = \boldsymbol{\alpha}^* \mathbf{K} \boldsymbol{\alpha}$. Furthermore, for any $\boldsymbol{u} \in \mathcal{X}$ and any $f \in \mathcal{H}$ we have $f(\boldsymbol{u}) = \langle f, k^{\boldsymbol{u}} \rangle_{\mathcal{H}}$. For details on reproducing kernel Hilbert spaces we refer the reader to [Aro50].

KRR problem data consists of observations $\{(\boldsymbol{x}_i, h_i)\}_{i=1}^m \subset \mathcal{X} \times \mathbb{R}$ and a positive regularization parameter λ. We presume there are functions g in $\mathcal{H}$ for which $g(\boldsymbol{x}_i) \approx h_i$, and we try to obtain such a function by solving

$$\min_{g \in \mathcal{H}} \ \frac{1}{m} \sum_{i=1}^{m} (g(\boldsymbol{x}_i) - h_i)^2 + \lambda \|g\|_{\mathcal{H}}^2. \tag{B.23}$$

It follows from [KW70] that the solution to (B.23) is in the span of the functions $\{k^{\boldsymbol{x}_i}\}_{i=1}^m$. Specifically, the solution is $g_\star = \sum_{i=1}^m \alpha_i k^{\boldsymbol{x}_i}$ where $\boldsymbol{\alpha}$ solves (B.22).

Why is ridge regression a special case of kernel ridge regression?

Suppose we have an $m \times n$ matrix $\mathbf{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n]$ with linearly independent columns, and that we want to estimate a linear functional $\hat{g} : \mathbb{R}^m \to \mathbb{R}$ given access to the n point evaluations $(\boldsymbol{x}_i, \hat{g}(\boldsymbol{x}_i))_{i=1}^n$.

Given a regularization parameter $\lambda > 0$, ridge regression concerns finding the linear function $g : \mathbb{R}^m \to \mathbb{R}$ that minimizes

$$L(g) = \left\| \begin{bmatrix} g(\boldsymbol{x}_1) \\ \vdots \\ g(\boldsymbol{x}_n) \end{bmatrix} - \begin{bmatrix} \hat{g}(\boldsymbol{x}_1) \\ \vdots \\ \hat{g}(\boldsymbol{x}_n) \end{bmatrix} \right\|_2^2 + n\lambda \|g\|^2.$$

To make this concrete, let us represent $\hat{g}$ and g by m-vectors $\hat{\boldsymbol{g}}$ and $\boldsymbol{g}$ respectively, and set $\boldsymbol{h} = \mathbf{X}^* \hat{\boldsymbol{g}}$. We also adopt a slight abuse of notation to write $L(g) = L(\boldsymbol{g})$, so that the task of ridge regression can be framed as minimizing

$$L(g) = \|\mathbf{X}^* \boldsymbol{g} - \boldsymbol{h}\|_2^2 + \lambda n \|\boldsymbol{g}\|_2^2.$$

Remark B.4.1. We pause to emphasize that this is a KRR problem with n datapoints that define functions on $\mathcal{X} = \mathbb{R}^m$. The parameter "$m$" here has nothing to do with the number of datapoints in the problem; our notational choices for (m, n) here are for consistency with Section 3.

The essential part of framing ridge regression as a type of KRR is showing that the optimal estimate $\boldsymbol{g}$ is in the range of $\mathbf{X}$. To see why this is the case, let $\mathbf{P}$ denote the orthogonal projector onto the range of $\mathbf{X}$. Using $\mathbf{X}^*\mathbf{P}\boldsymbol{g} = \mathbf{X}^*\boldsymbol{g}$, we have that

$$\begin{aligned} L(\boldsymbol{g}) &= \|\mathbf{X}^*\mathbf{P}\boldsymbol{g} - \boldsymbol{h}\|_2^2 + \lambda n \left(\|\mathbf{P}\boldsymbol{g}\|_2^2 + \|(\mathbf{I}-\mathbf{P})\boldsymbol{g}\|_2^2\right) \\ &\geq \|\mathbf{X}^*\mathbf{P}\boldsymbol{g} - \boldsymbol{h}\|_2^2 + \lambda n \|\mathbf{P}\boldsymbol{g}\|_2^2 \\ &= L(\mathbf{P}\boldsymbol{g}), \end{aligned}$$

and so g minimizes L only if $L(\mathbf{P}\boldsymbol{g}) = L(\boldsymbol{g})$. Since $L(\mathbf{P}\boldsymbol{g}) = L(\boldsymbol{g})$ holds if and only if $\mathbf{P}\boldsymbol{g} = \boldsymbol{g}$, we have that $\boldsymbol{g} = \mathbf{X}\boldsymbol{\alpha}$ for some $\boldsymbol{\alpha}$ in $\mathbb{R}^n$. Therefore, under our stated assumption that the columns of $\mathbf{X}$ are linearly independent, the following problems are equivalent

$$\begin{aligned} &\arg\min\{L(g) \,:\, g \text{ is a linear functional on } \mathbb{R}^m\}, \\ &\arg\min\{\|\mathbf{X}^*\mathbf{X}\boldsymbol{\alpha} - \boldsymbol{h}\|_2^2 + \lambda n \|\mathbf{X}\boldsymbol{\alpha}\|_2^2 \,:\, \boldsymbol{\alpha} \in \mathbb{R}^n\}, \text{ and} \\ &\arg\min\{\|\mathbf{X}\boldsymbol{\alpha} - \hat{\boldsymbol{g}}\|_2^2 + \lambda n \|\boldsymbol{\alpha}\|_2^2 \,:\, \boldsymbol{\alpha} \in \mathbb{R}^n\}. \end{aligned}$$

The second of these problems is KRR with a scaled objective and the $n \times n$ kernel matrix $\mathbf{K} = \mathbf{X}^*\mathbf{X}$. The last of these problems is ridge regression in the familiar form.

Given this description of ridge regression, one obtains KRR by applying the so-called "kernel trick" (see, e.g., [Mur12, §14]). That is, one replaces $h_j = \boldsymbol{x}_j^*\hat{\boldsymbol{g}}$ by

$$h_j = \langle k^{\boldsymbol{x}_j}, \hat{g} \rangle_{\mathcal{H}} = \hat{g}(\boldsymbol{x}_j)$$

and expresses the point evaluation of $g = \sum_{i=1}^n \alpha_i k^{\boldsymbol{x}_i}$ at $\boldsymbol{x}_j$ by

$$g(\boldsymbol{x}_j) = \sum_{i=1}^n \alpha_i \langle k^{\boldsymbol{x}_i}, k^{\boldsymbol{x}_j} \rangle_{\mathcal{H}}.$$

We note that within the KRR formalism it is allowed for $\mathbf{K}$ to be singular, so long as it is psd. This is because if $\boldsymbol{\beta}$ is any vector in the kernel of $\mathbf{K}$ then the function $\boldsymbol{u} \mapsto \sum_{i=1}^n \beta_i k(\boldsymbol{x}_i, \boldsymbol{u})$ is identically equal to zero.

B.4.2 Efficient sketch-and-solve for regularized quadratics

Let $\mathbf{G}$ be an $m \times m$ psd matrix and μ be a positive regularization parameter. The sketch-and-solve approach to KRR from [AM15] can be considered generically as a sketch-and-solve approach to the regularized quadratic minimization problem (3.1). The generic formulation is to approximate $\mathbf{G} \approx \mathbf{A}\mathbf{A}^*$ with an $m \times n$ matrix $\mathbf{A}$ ($m \gg n$) and then solve

$$(\mathbf{A}\mathbf{A}^* + \mu\mathbf{I})\,\boldsymbol{z} = \boldsymbol{h}. \tag{B.24}$$

Identifying $\boldsymbol{b} = \boldsymbol{h}/\mu$, $\boldsymbol{c} = \mathbf{0}$, and $\boldsymbol{y} = \boldsymbol{z}$ shows that this amounts to a dual saddle point problem of the form (3.3). Here we explain how the sketch-and-precondition paradigm can efficiently be applied to solve (B.24) under the assumption that $\mathbf{A}\mathbf{A}^*$ defines a Nyström approximation of $\mathbf{G}$.

Let $\mathbf{S}_o$ be an initial $m \times n$ sketching operator. The resulting sketch $\mathbf{Y} = \mathbf{G}\mathbf{S}_o$ and factor $\mathbf{R} = \texttt{chol}(\mathbf{S}_o^*\mathbf{Y})$ together define $\mathbf{A} = \mathbf{Y}\mathbf{R}^{-1}$. This defines a Nyström approximation since

$$\mathbf{A}\mathbf{A}^* = (\mathbf{K}\mathbf{S}_o)\,(\mathbf{S}_o^*\mathbf{K}\mathbf{S}_o)^\dagger\,(\mathbf{K}\mathbf{S}_o)^*\,.$$

Recall that the problem of preconditioner generation entails finding an orthogonalizer of a sketch of $\mathbf{A}_\mu = [\mathbf{A}; \sqrt{\mu}\mathbf{I}]$. The fact that $\mathbf{A}$ is only represented implicitly makes this delicate, but it remains doable, as we explain below.

For the sketching phase of preconditioner generation, we sample a $d \times m$ operator $\mathbf{S}$ (with $d \gtrsim n$) and set

$$\mathbf{A}_\mu^{\mathrm{sk}} = \begin{bmatrix} \mathbf{S} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \mathbf{A}_\mu = \begin{bmatrix} \mathbf{SY} \\ \sqrt{\mu}\mathbf{R} \end{bmatrix} \mathbf{R}^{-1}.$$

We then compute the SVD of the augmented matrix

$$\begin{bmatrix} \mathbf{SY} \\ \sqrt{\mu}\mathbf{R} \end{bmatrix} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*$$

and find that setting $\mathbf{M} = \mathbf{R}\mathbf{V}\mathbf{\Sigma}^{-1}$ satisfies $\mathbf{A}_\mu^{\mathrm{sk}}\mathbf{M} = \mathbf{U}$. The preconditioned linear operator $\mathbf{A}_\mu\mathbf{M}$ (and its adjoint) should be applied in the iterative solver by noting the identity

$$\begin{bmatrix} \mathbf{A} \\ \sqrt{\mu}\mathbf{I} \end{bmatrix} \mathbf{M} = \begin{bmatrix} \mathbf{Y} \\ \sqrt{\mu}\mathbf{R} \end{bmatrix} \mathbf{V}\mathbf{\Sigma}^{-1}.$$

This identity is important since it shows that $\mathbf{R}^{-1}$ need never be applied at any point in the sketch-and-precondition approach to (B.24).

Appendix C

Details on Low-Rank Approximation

C.1 Theory for submatrix-oriented decompositions

C.1.1 Approximation quality in low-rank ID

Proposition C.1.1 (Restatement of Proposition 4.1.3)**.** *Let* $\tilde{\mathbf{A}}$ *be any rank-*k *approximation of* $\mathbf{A}$ *that satisfies the spectral norm error bound* $\|\mathbf{A} - \tilde{\mathbf{A}}\|_2 \leq \epsilon$*. If* $\tilde{\mathbf{A}} = \tilde{\mathbf{A}}[:, J]\mathbf{X}$ *for some* $k \times n$ *matrix* $\mathbf{X}$ *and an index vector* J*, then* $\hat{\mathbf{A}} = \mathbf{A}[:, J]\mathbf{X}$ *is a low-rank column ID that satisfies*

$$\|\mathbf{A} - \hat{\mathbf{A}}\|_2 \leq (1 + \|\mathbf{X}\|_2)\epsilon. \qquad \text{((4.16), restated)}$$

Furthermore, if $|X_{ij}| \leq M$ *for all* (i, j)*, then*

$$\|\mathbf{X}\|_2 \leq \sqrt{1 + M^2 k(n-k)}. \qquad \text{((4.17), restated)}$$

Proof. Proceeding in the grand tradition of adding zero and applying the triangle inequality, we have the bound

$$\begin{aligned} \|\mathbf{A} - \hat{\mathbf{A}}\|_2 &= \|(\mathbf{A} - \tilde{\mathbf{A}}) + (\tilde{\mathbf{A}} - \hat{\mathbf{A}})\|_2 \\ &\leq \|\mathbf{A} - \tilde{\mathbf{A}}\|_2 + \|\tilde{\mathbf{A}} - \hat{\mathbf{A}}\|_2. \end{aligned} \qquad \text{(C.1)}$$

We prove (4.16) by bounding the two terms in (C.1). The first term is trivial since we have already assumed $\|\mathbf{A} - \tilde{\mathbf{A}}\|_2 \leq \epsilon$. We bound the second term by using the

identity $\tilde{\mathbf{A}} - \hat{\mathbf{A}} = (\tilde{\mathbf{A}}[:, J] - \mathbf{A}[:, J])\mathbf{X}$ and then invoking submultiplicivity of the spectral norm:

$$\|\tilde{\mathbf{A}} - \hat{\mathbf{A}}\|_2 \leq \|\tilde{\mathbf{A}}[:, J] - \mathbf{A}[:, J]\|_2 \|\mathbf{X}\|_2. \tag{C.2}$$

Finally, since the spectral norm of a matrix is at least as large as the spectral norm of any of its submatrices, we obtain $\|\tilde{\mathbf{A}}[:, J] - \mathbf{A}[:, J]\|_2 \leq \|\tilde{\mathbf{A}} - \mathbf{A}\|_2 \leq \epsilon$. Combining this with (C.2) shows that (C.1) implies (4.16).

Now we address (4.17). For this, consider the $m \times k$ matrix $\mathbf{C} = \tilde{\mathbf{A}}[:, J]$. Because we have assumed $\tilde{\mathbf{A}} = \mathbf{C}\mathbf{X}$ has rank k and that $\mathbf{X}$ is $k \times n$, we can infer that $\mathbf{C}$ is full column-rank. We can also extract the columns from both sides of $\tilde{\mathbf{A}} = \mathbf{C}\mathbf{X}$ at indices J to find the identity $\mathbf{C} = \mathbf{C}\mathbf{X}[:, J]$. Multiplying this identity through on the left by $\mathbf{C}^\dagger$, we can use $\mathbf{C}^\dagger\mathbf{C} = \mathbf{I}_k$ to obtain $\mathbf{X}[:, J] = \mathbf{I}_k$.

Now if $|X_{ij}| \leq M$ for all (i, j) *in addition to* $\mathbf{X}[:, J] = \mathbf{I}_k$, then there exists a permutation P of the column index set $[\![n]\!]$ where $\mathbf{X}[:, P] = [\mathbf{I}_k, \mathbf{V}]$ for a $k \times (n-k)$ matrix $\mathbf{V}$ satisfying $|V_{ij}| \leq M$. Since permuting the columns of $\mathbf{X}$ does not change its spectral norm, it suffices to bound $\|[\mathbf{I}_k, \mathbf{V}]\|_2$. Towards this end, we claim without proof that for any block matrix $\mathbf{W} = [\mathbf{U}, \mathbf{V}]$, one has

$$\|\mathbf{W}\|_2 \leq \sqrt{\|\mathbf{U}\|_2^2 + \|\mathbf{V}\|_{\mathrm{F}}^2}.$$

This, combined with $\|\mathbf{I}_k\|_2 = 1$ and $\|\mathbf{V}\|_{\mathrm{F}}^2 \leq M^2 k(n-k)$, gives (4.17). □

Remark C.1.2. The bound (4.17) is not the best possible. Indeed, looking at the final steps in the proposition's proof, we see that it suffices for M to bound the entries of $\mathbf{X}$ that are not part of the identity submatrix.

C.1.2 Truncation in column-pivoted matrix decompositions

This part of the appendix follows up on Section 4.3.3. It examines how changing basis of a column-pivoted decomposition can affect approximation quality when truncating these decompositions. To begin, we set forth some definitions.

Our matrix of interest, $\mathbf{G}$, is $r \times c$ and given through a decomposition $\mathbf{G}\mathbf{P} = \mathbf{F}\mathbf{T}$ for a permutation matrix $\mathbf{P}$ and upper-triangular $\mathbf{T}$. Let us say that $\mathbf{F}$ has $w = \min\{c, r\}$ columns (and hence that $\mathbf{T}$ has as many rows). We use $\mathcal{U}$ to denote the set invertible upper-triangular matrices of order w. For a positive integer $k < \operatorname{rank}(\mathbf{G})$, we consider the matrix-valued function g_k that is defined on $\mathcal{U}$ according to

$$g_k(\mathbf{U}) = \mathbf{F}(\mathbf{U}^{-1})[:, :k]\mathbf{U}[:k, :]\mathbf{T}\mathbf{P}^*.$$

Note that for every diagonal $\mathbf{D} \in \mathcal{U}$ we have $g_k(\mathbf{D}) = (\mathbf{F}[:, :k])(\mathbf{T}[:k, :])\mathbf{P}^*$.

Proposition C.1.3. *Partition the factors* $\mathbf{F}$ *and* $\mathbf{T}$ *into blocks* $[\mathbf{F}_1, \mathbf{F}_2]$ *and* $[\mathbf{T}_1; \mathbf{T}_2]$ *so that* $\mathbf{F}_1$ *has* k *columns and* $\mathbf{T}_1$ *has* k *rows. If* $\mathbf{U}_\star$ *is an optimal solution to*

$$\min_{\mathbf{U} \in \mathcal{U}} \|\mathbf{G} - g_k(\mathbf{U})\|_{\mathrm{F}}. \tag{C.3}$$

then the following identity holds in any unitarily invariant norm:

$$\|\mathbf{G} - g_k(\mathbf{U}_\star)\| = \left\| \left(\mathbf{I} - \mathbf{F}_1\mathbf{F}_1^\dagger\right)\mathbf{F}_2\mathbf{T}_2 \right\|.$$

Furthermore, we have $\|\mathbf{G} - g_k(\mathbf{I}_{w\times w})\| = \|\mathbf{F}_2\mathbf{T}_2\|$*, and the identity matrix is optimal for* (C.3) *if and only if* $\operatorname{range}(\mathbf{F}_2)$ *and* $\operatorname{range}(\mathbf{F}_1)$ *are orthogonal.*

Proof. Our proof requires working with several block matrices. First, the matrices $\mathbf{T}_1$ and $\mathbf{T}_2$ are further partitioned so that

$$\mathbf{T} = \begin{bmatrix} \mathbf{T}_1 \\ \mathbf{T}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{T}_{11} & \mathbf{T}_{12} \\ \mathbf{0} & \mathbf{T}_{22} \end{bmatrix}$$

where $\mathbf{T}_{11}$ is $k \times k$ and $\mathbf{T}_{22}$ is $(w-k) \times (c-k)$. Next, we introduce $\mathbf{U} \in \mathcal{U}$ and partition it twice:

$$\mathbf{U} = \begin{bmatrix} \mathbf{U}_1 \\ \mathbf{U}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{U}_{11} & \mathbf{U}_{12} \\ \mathbf{0} & \mathbf{U}_{22} \end{bmatrix}.$$

In the expressions above, $\mathbf{U}_1$ is a wide matrix of shape $k \times w$, $\mathbf{U}_{11}$ is a square matrix of order k, and $\mathbf{U}_{22}$ a square matrix of order $w-k$. Note that since $\mathbf{U}$ is upper-triangular, the same is true of $\mathbf{V} = \mathbf{U}^{-1}$. We partition $\mathbf{V} = [\mathbf{V}_1, \mathbf{V}_2]$ into a leading block of k columns and a trailing block of $w-k$ columns.

The point of all this notation is to help us find useful expressions for $g_k(\mathbf{U})$. Our first such expression is

$$g_k(\mathbf{U}) = \mathbf{F}\mathbf{V}_1\mathbf{U}_1\mathbf{T}\mathbf{P}^*. \tag{C.4}$$

As a step towards finding the next expression, we apply block a matrix-inversion identity to compute

$$\mathbf{F}\mathbf{V}_1 = \mathbf{F}_1\mathbf{U}_{11}^{-1}.$$

Meanwhile, a simple block matrix multiply gives

$$\mathbf{U}_1\mathbf{T} = \mathbf{U}_{11}\mathbf{T}_1 + \mathbf{U}_{12}\mathbf{T}_2.$$

We plug these two identities into (C.4) to obtain

$$g_k(\mathbf{U}) = \mathbf{F}_1 \left(\mathbf{T}_1 + \mathbf{U}_{11}^{-1}\mathbf{U}_{12}\mathbf{T}_2 \right) \mathbf{P}^*.$$

This expression is just what we need. By combining it with the identity $\mathbf{G} = \mathbf{F}\mathbf{T}\mathbf{P}^*$, we easily compute the difference

$$\mathbf{G} - g_k(\mathbf{U}) = (\mathbf{F}_2\mathbf{T}_2 - \mathbf{F}_1\mathbf{U}_{11}^{-1}\mathbf{U}_{12}\mathbf{T}_2)\mathbf{P}^*. \tag{C.5}$$

Having access to (C.5) marks a checkpoint in our proof. With it, we obtain the following identity for the distance from $\mathbf{G}$ to $g_k(\mathbf{U})$ in any unitarily invariant norm:

$$\|\mathbf{G} - g_k(\mathbf{U})\| = \|\mathbf{F}_2\mathbf{T}_2 - \mathbf{F}_1\mathbf{U}_{11}^{-1}\mathbf{U}_{12}\mathbf{T}_2\|.$$

This implies our claim about $g_k(\mathbf{I}_{w\times w})$, since if $\mathbf{U}$ is diagonal, then $\mathbf{U}_{12} = \mathbf{0}$. Therefore all that remains is our claim about matrices that solve (C.3). The truth of this claim is easier to see after a change variables. Upon replacing $\mathbf{U}_{11}^{-1}\mathbf{U}_{12}$ by a general $k \times (w-k)$ matrix, we can express

$$\min_{\mathbf{U}\in\mathcal{U}} \|\mathbf{G} - g_k(\mathbf{U})\|_{\mathrm{F}} = \min \left\{ \|\mathbf{F}_2\mathbf{T}_2 - \mathbf{F}_1\mathbf{B}\mathbf{T}_2\|_{\mathrm{F}} \;\middle|\; \mathbf{M} \in \mathbb{R}^{k\times(w-k)} \right\},$$

and it is easily shown that the matrix $\mathbf{M}_\star = \mathbf{F}_1^\dagger\mathbf{F}_2$ is an optimal solution to the problem on the right-hand side of this equation. □

C.2 Computational routine interfaces and implementations

As we explained in Section 4, the design space for low-rank approximation algorithms is quite large. Here we illustrate the breadth and depth of that design space with pseudocode for computational routines needed for four drivers: `SVD1`, `EVD1`, `EVD2`, and `CURD1` (Algorithms 3 through 6, respectively). All pseudocode here uses Python-style zero-based indexing.

The dependency structure of these drivers and their supporting functions is given in Figure C.1. From the figure we see that the following three interfaces are central to low-rank approximation.

- $\mathbf{Y} = \texttt{Orth}(\mathbf{X})$ returns an orthonormal basis for the range of a tall input matrix; the number of columns in $\mathbf{Y}$ will never be larger than that of $\mathbf{X}$ and may be smaller. The simplest implementation of `Orth` is to return the orthogonal factor from an economic QR decomposition of $\mathbf{X}$.
- $\mathbf{S} = \texttt{SketchOpGen}(\ell, k)$ returns an $\ell \times k$ oblivious sketching operator sampled from some predetermined distribution. The most common distributions used for low-rank approximation were covered in Section 2.3. In actual implementations, this function would accept an input representing the state of the random number generator.
- $\mathbf{Y} = \texttt{Stabilizer}(\mathbf{X})$ has similar semantics `Orth`. It differs in that it only requires $\mathbf{Y}$ to be better-conditioned than $\mathbf{X}$ while preserving its range. The relaxed semantics open up the possibility of methods that are less expensive than computing an orthonormal basis, such as taking the lower-triangular factor from an LU decomposition with column pivoting.

We explain the remaining interfaces as they arise in our implementations.

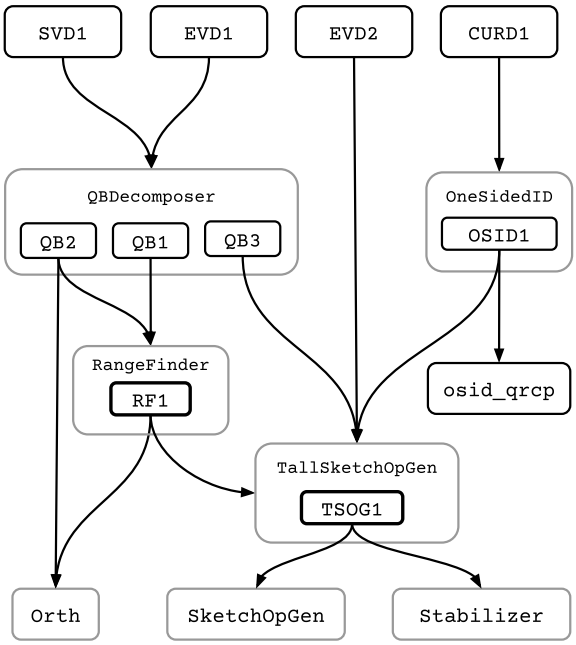

Figure C.1: Dependency illustration for low-rank approximation functionality. Lighter gray boxes correspond to abstract *interfaces* which specify semantics. Any interface can have many different *implementations*. To keep things at a reasonable length the only interface with multiple implementations is `QBDecomposer`. Section 4.3 describes many algorithms that could be used for any such interface.

The computational routines represented in Figure C.1 include Algorithms 9 through 14. This appendix provides pseudocode for one additional function that is not reflected in the figure: Algorithm 15 shows one way to perform row or column subset selection. We note that while Algorithm 15 is not used in the drivers mentioned above, it could easily have been used in a different implementation of `CURD1`.

C.2.1 Power iteration for data-aware sketching

When a `TallSketchOpGen` is called with parameters $(\mathbf{A}, k)$, it produces an $n \times k$ sketching operator where range($\mathbf{S}$) is reasonably well-aligned with the subspace spanned by the k leading right singular vectors of $\mathbf{A}$. Here, "reasonably" is assessed with respect to the computational cost incurred by running `TallSketchOpGen`. One extreme case of interest is to return an oblivious sketching operator without reading any entries of $\mathbf{A}$.

This method uses a p-step power iteration technique. When $p = 0$, the method returns an oblivious sketching operator. It is recommended that one use $p > 0$ (e.g., $p \in \{2, 3\}$) when the singular values of $\mathbf{A}$ exhibit "slow" decay.

Algorithm 8 `TSOG1` : a `TallSketchOpGen` based on a power method, conceptually following [ZM20]. The returned sketching operator is suitable for sketching $\mathbf{A}$ from the right for purposes of low-rank approximation.

1: **function** `TSOG1`($\mathbf{A}, k$)

Inputs:
$\mathbf{A}$ is $m \times n$, and $k \ll \min\{m, n\}$ is a positive integer.

Output:
$\mathbf{S}$ is $n \times k$, intended for later use in computing $\mathbf{Y} = \mathbf{AS}$.

Abstract subroutines:
`SketchOpGen` and `Stabilizer`

Tuning parameters:
$p \geq 0$ controls the number of steps in the power method. It is equal to the total number of matrix-matrix multiplications that will involve either $\mathbf{A}$ or $\mathbf{A}^*$. If $p = 0$ then this function returns an oblivious sketching operator.

$q \geq 1$ is the number of matrix-matrix multiplications with $\mathbf{A}$ or $\mathbf{A}^*$ that accumulate before the stabilizer is called.

2: $p_{\text{done}} = 0$
3: **if** p is even **then**
4: $\mathbf{S} = \texttt{SketchOpGen}(n, k)$
5: **else**
6: $\mathbf{S} = \mathbf{A}^*\texttt{SketchOpGen}(m, k)$
7: $p_{\text{done}} = p_{\text{done}} + 1$
8: **if** $p_{\text{done}} \mod q = 0$ **then**
9: $\mathbf{S} = \texttt{stabilizer}(\mathbf{S})$
10: **while** $p - p_{\text{done}} \geq 2$ **do**
11: $\mathbf{S} = \mathbf{AS}$
12: $p_{\text{done}} = p_{\text{done}} + 1$
13: **if** $p_{\text{done}} \mod q = 0$ **then**
14: $\mathbf{S} = \texttt{stabilizer}(\mathbf{S})$
15: $\mathbf{S} = \mathbf{A}^*\mathbf{S}$
16: $p_{\text{done}} = p_{\text{done}} + 1$
17: **if** $p_{\text{done}} \mod q = 0$ **then**
18: $\mathbf{S} = \texttt{stabilizer}(\mathbf{S})$
19: **return** $\mathbf{S}$

C.2.2 RangeFinders and QB decompositions

A general `RangeFinder` takes in a matrix $\mathbf{A}$ and a target rank parameter k, and returns a matrix $\mathbf{Q}$ of rank $d = \min\{k, \text{rank}(\mathbf{A})\}$ such that the range of $\mathbf{Q}$ is an approximation to the space spanned by $\mathbf{A}$'s top d left singular vectors.

The rangefinder problem may also be viewed in the following way: given a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ and a target rank $k \ll \min(m, n)$, find a matrix $\mathbf{Q}$ with k columns such that the error $\|\mathbf{A} - \mathbf{QQ}^*\mathbf{A}\|$ is "reasonably" small. Some `RangeFinder` implementations are iterative and can accept a target accuracy as a third argument.

The `RangeFinder` below, `RF1`, is very simple. It relies on an implementation of the `TallSketchOpGen` interface (e.g., `TSOG1`) as well as the `Orth` interface.

Algorithm 9 `RF1` : a `RangeFinder` that orthogonalizes a single row sketch

function `RF1`($\mathbf{A}, k$)

Inputs:
$\mathbf{A}$ is $m \times n$, and $k \ll \min\{m, n\}$ is a positive integer

Output:
$\mathbf{Q}$ is a column-orthonormal matrix with $d = \min\{k, \operatorname{rank} \mathbf{A}\}$ columns. We have $\operatorname{range}(\mathbf{Q}) \subset \operatorname{range}(\mathbf{A})$; it is intended that $\operatorname{range}(\mathbf{Q})$ is an approximation to the space spanned by $\mathbf{A}$'s top d left singular vectors.

Abstract subroutines and tuning parameters:
`TallSketchOpGen`

$\mathbf{S} = \texttt{TallSketchOpGen}(\mathbf{A}, k)$ # $\mathbf{S}$ is $n \times k$
$\mathbf{Y} = \mathbf{AS}$
$\mathbf{Q} = \texttt{orth}(\mathbf{Y})$
return $\mathbf{Q}$

The conceptual goal of QB decomposition algorithms is to produce an approximation $\|\mathbf{A} - \mathbf{QB}\| \leq \epsilon$ (for some unitarily-invariant norm), where $\operatorname{rank}(\mathbf{QB}) \leq \min\{k, \operatorname{rank}(\mathbf{A})\}$. Our next three algorithms are different implementations of the `QBDecomposer` interface. The first two of these algorithms require an implementation of the `RangeFinder` interface. The ability of the implementation `QB1` to control accuracy is completely dependent on that of the underlying rangefinder.

Algorithm 10 `QB1` : a `QBDecomposer` that falls back on an abstract rangefinder

function `QB1`($\mathbf{A}, k, \epsilon$)

Inputs:
$\mathbf{A}$ is an $m \times n$ matrix and $k \ll \min\{m, n\}$ is a positive integer.

ϵ is a target for the relative error $\|\mathbf{A} - \mathbf{QB}\|/\|\mathbf{A}\|$ measured in some unitarily-invariant norm. This parameter is passed directly to the `RangeFinder`, which determines its precise interpretation.

Output:
$\mathbf{Q}$ an $m \times d$ matrix returned by the underlying `RangeFinder` and $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$ is $d \times n$; we can be certain that $d \leq \min\{k, \operatorname{rank}(\mathbf{A})\}$. The matrix $\mathbf{QB}$ is a low-rank approximation of $\mathbf{A}$.

Abstract subroutines and tuning parameters:
`RangeFinder`

$\mathbf{Q} = \texttt{RangeFinder}(\mathbf{A}, k, \epsilon)$
$\mathbf{B} = \mathbf{Q}^*\mathbf{A}$
return $\mathbf{Q}, \mathbf{B}$

The following algorithm builds up a QB decomposition incrementally. It's said to be *fully-adaptive* because it has fine-grained control over the error $\|\mathbf{A}-\mathbf{QB}\|_{\mathrm{F}}$. If the algorithm is called with $k = \min\{m, n\}$, then its output will satisfy $\|\mathbf{A}-\mathbf{QB}\|_{\mathrm{F}} \leq \epsilon$.

Algorithm 11 `QB2` : a `QBDecomposer` that's fully-adaptive
(see [YGL18, Algorithm 2])

1: **function** `QB2`($\mathbf{A}, k, \epsilon$)

Inputs:

$\mathbf{A}$ is an $m \times n$ matrix and $k \ll \min\{m, n\}$ is a positive integer.

ϵ is a target for the relative error $\|\mathbf{A} - \mathbf{QB}\|_{\mathrm{F}} / \|\mathbf{A}\|_{\mathrm{F}}$. This parameter is used as a termination criterion upon reaching the desired accuracy.

Output:

$\mathbf{Q}$ an $m \times d$ matrix combined of successive outputs from the underlying `RangeFinder` and $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$ is $d \times n$; we can be certain that $d \leq \min\{k, \mathrm{rank}(\mathbf{A})\}$. The matrix $\mathbf{QB}$ is a low-rank approximation of $\mathbf{A}$.

Abstract subroutines:

`RangeFinder`

Tuning parameters:

block_size ≥ 1 - at every iteration (except possibly for the final iteration), block_size columns are added to the matrix $\mathbf{Q}$.

2: $d = 0$
3: $\mathbf{Q} = [\,] \in \mathbb{R}^{m \times d}$ # Preallocation is dangerous; $k = \min\{m, n\}$ is allowed.
4: $\mathbf{B} = [\,] \in \mathbb{R}^{d \times n}$
5: squared_error $= \|\mathbf{A}\|_{\mathrm{F}}^2$
6: **while** $k > d$ **do**
7: block_size $= \min\{\text{block\_size},\ k - d\}$
8: $\mathbf{Q}_i =$ `RangeFinder`$(\mathbf{A},$ block_size$)$
9: $\mathbf{Q}_i =$ `orth`$(\mathbf{Q}_i - \mathbf{Q}\,(\mathbf{Q}^* \mathbf{Q}_i))$ # for numerical stability
10: $\mathbf{B}_i = \mathbf{Q}_i^* \mathbf{A}$ # original matrix $\mathbf{A}$ is valid here
11: $\mathbf{B} = \begin{bmatrix} \mathbf{B} \\ \mathbf{B}_i \end{bmatrix}$
12: $\mathbf{Q} = \begin{bmatrix} \mathbf{Q} & \mathbf{Q}_i \end{bmatrix}$
13: $d = d +$ block_size
14: $\mathbf{A} = \mathbf{A} - \mathbf{Q}_i \mathbf{B}_i$ # modification can be implicit, but is required by Line 8
15: squared_error $=$ squared_error $- \|\mathbf{B}_i\|_{\mathrm{F}}^2$ # compute by a stable method
16: **if** squared_error $\leq \epsilon^2$ **then**
17: `break`
18: **return** $\mathbf{Q}, \mathbf{B}$

Our third and final QB algorithm also builds up its approximation incrementally. It is called *pass-efficient* because it does not access the data matrix $\mathbf{A}$ within its main loop (see [DKM06a] for the original definition of the pass-efficient model). The algorithm can use a requested error tolerance as an early-stopping criterion. This function should never be called with $k = \min\{m, n\}$. We note that it takes a fair amount of algebra to prove that this algorithm produces a correct result.

Algorithm 12 `QB3` : a `QBDecomposer` that's pass-efficient and partially adaptive (based on [YGL18, Algorithm 4])

function `QB3`($\mathbf{A}, k, \epsilon$)

Inputs:

$\mathbf{A}$ is an $m \times n$ matrix and $k \ll \min\{m, n\}$ is a positive integer.

ϵ is a target for the relative error $\|\mathbf{A} - \mathbf{QB}\|_{\mathrm{F}} / \|\mathbf{A}\|_{\mathrm{F}}$. This parameter is used as a termination criterion upon reaching the desired accuracy.

Output:

$\mathbf{Q}$ an $m \times d$ matrix combined of successively-computed orthonormal bases $\mathbf{Q}_i$ and $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$ is $d \times n$; we can be certain that $d \leq \min\{k, \mathrm{rank}(\mathbf{A})\}$. The matrix $\mathbf{QB}$ is a low-rank approximation of $\mathbf{A}$.

Abstract subroutines:

`TallSketchOpGen`

Tuning parameters:

block_size is a positive integer; at every iteration (except possibly for the last), we add block_size columns to $\mathbf{Q}$.

$\mathbf{Q} = [\,] \in \mathbb{R}^{m \times 0}$ # It would be preferable to preallocate.
$\mathbf{B} = [\,] \in \mathbb{R}^{0 \times n}$
squared_error $= \|\mathbf{A}\|_{\mathrm{F}}^2$
$\mathbf{S} =$ `TallSketchOpGen`$(\mathbf{A}, k)$
$\mathbf{G} = \mathbf{AS},\ \mathbf{H} = \mathbf{A}^* \mathbf{G}$ # Can be done in one pass over $\mathbf{A}$
max_blocks $= \lceil k / \text{block\_size} \rceil$
$i = 0$
while $i <$ max_blocks **do**
    $b_{\text{start}} = i \cdot \text{block\_size} + 1$
    $b_{\text{end}} = \min\{(i+1) \cdot \text{block\_size},\ k\}$
    $\mathbf{S}_i = \mathbf{S}[:,\ b_{\text{start}} : b_{\text{end}}]$
    $\mathbf{Y}_i = \mathbf{G}[:,\ b_{\text{start}} : b_{\text{end}}] - \mathbf{Q}(\mathbf{BS}_i)$
    $\mathbf{Q}_i, \mathbf{R}_i =$ `qr`$(\mathbf{Y}_i)$ # the next three lines are for numerical stability
    $\mathbf{Q}_i = \mathbf{Q}_i - \mathbf{Q}(\mathbf{Q}^* \mathbf{Q}_i)$
    $\mathbf{Q}_i, \hat{\mathbf{R}}_i =$ `qr`$(\mathbf{Q}_i)$
    $\mathbf{R}_i = \hat{\mathbf{R}}_i \mathbf{R}_i$
    $\mathbf{B}_i = (\mathbf{H}[:,\ b_{\text{start}} : b_{\text{end}}])^* - (\mathbf{Y}_i \mathbf{Q})\mathbf{B} - (\mathbf{BS}_i)^* \mathbf{B}$
    $\mathbf{B}_i = (\mathbf{R}_i^*)^{-1} \mathbf{B}_i$ # in-place triangular solve
    $\mathbf{B} = \begin{bmatrix} \mathbf{B} \\ \mathbf{B}_i \end{bmatrix}$
    $\mathbf{Q} = \begin{bmatrix} \mathbf{Q} & \mathbf{Q}_i \end{bmatrix}$
    squared_error $=$ squared_error $- \|\mathbf{B}_i\|_{\mathrm{F}}^2$ # compute by a stable method
    $i = i + 1$
    if squared_error $\leq \epsilon^2$ **then**
        `break`
return $\mathbf{Q}, \mathbf{B}$

C.2.3 ID and subset selection

As we indicated in Sections 4.2.3 and 4.3.3, the collective design space of algorithms for ID, subset selection, and CUR is very large. This appendix presents one randomized algorithm for one-sided ID (Algorithm 14) and an analogous randomized algorithm for subset selection (Algorithm 15). These algorithms are implemented in our Python prototype. The Python prototype has two more randomized algorithms which are not reproduced here (one for one-sided and one for two-sided ID).

We need two deterministic functions in order to state these algorithms. The first deterministic function – called as $\mathbf{Q}, \mathbf{R}, J = \texttt{qrcp}(\mathbf{F}, k)$ – returns data for an economic QR decomposition with column pivoting, where the decomposition is restricted to rank k and may be incomplete. The second deterministic function (Algorithm 13, below) is the canonical way to use QRCP for one-sided ID. It produces a column ID when the final argument "axis" is set to one; otherwise, it produces a row ID. When used for column ID, it's typical for $\mathbf{Y} \in \mathbb{R}^{\ell \times w}$ to be (very) wide and for k to be only slightly smaller than ℓ (say, $\ell/2 \leq k \leq \ell$).

Algorithm 13 deterministic one-sided ID based on QRCP

function `osid_qrcp`$(\mathbf{Y}, k, \text{axis})$

Inputs:

$\mathbf{Y}$ is an $\ell \times w$ matrix, typically a sketch of some larger matrix.

k is an integer, typically close to $\min\{\ell, w\}$.

axis is an integer, equals 1 for row ID and 2 for column ID.

Outputs:

When axis $= 1$:

$\mathbf{Z}$ is $\ell \times k$ and I is a length-k index vector.

Together, they satisfy $\mathbf{Y}[I, :] = (\mathbf{Z}\mathbf{Y}[I, :])[I, :]$.

When axis $= 2$:

$\mathbf{X}$ is $k \times w$ and J is a length-k index vector.

Together, they satisfy $\mathbf{Y}[:, J] = (\mathbf{Y}[:, J]\mathbf{X})[:, J]$.

Abstract subroutines:

`qrcp`

if axis == 2 **then**
  $(\ell, w) =$ the number of (rows, columns) in $\mathbf{Y}$
  assert $k \leq \min\{\ell, w\}$
  $\mathbf{Q}, \mathbf{R}, J = \texttt{qrcp}(\mathbf{Y}, k)$
  $\mathbf{T} = (\mathbf{R}[:k,\ :k])^{-1}\, \mathbf{R}[:k, k+1:]$ # use `trsm` from BLAS 3
  $\mathbf{X} = \texttt{zeros}(k, w)$
  $\mathbf{X}[:, J] = [\mathbf{I}_{k \times k}, \mathbf{T}]$
  $J = J[:k]$
  return $\mathbf{X}, J$
else
  $\mathbf{X}, I = \texttt{osid\_qrcp}(\mathbf{Y}^*, k, \text{axis} = 1)$
  $\mathbf{Z} = \mathbf{X}^*$
  return $\mathbf{Z}, I$

The one-sided ID interface is

$$\mathbf{M}, P = \texttt{OneSidedID}(\mathbf{A}, k, s, \text{axis}).$$

The output value $\mathbf{M}$ is the interpolation matrix and P is the length-k vector of skeleton indices. When axis $= 1$ we are considering a row ID and so obtain the approximation $\hat{\mathbf{A}} = \mathbf{M}\mathbf{A}[P, :]$ to $\mathbf{A}$. When axis $= 2$, we are considering the low-rank column ID $\hat{\mathbf{A}} = \mathbf{A}[:, P]\mathbf{M}$. Implementations of this interface perform internal calculations with sketches of rank $k + s$.

Algorithm 14 `OSID1` : implements `OneSidedID` by re-purposing an ID of a sketch. Besides the original source [VM16, §5.1], more information on this algorithm can be found in [Mar18, §10.4] and [MT20, §13.4].

1: **function** `OSID1`($\mathbf{A}, k, \text{axis}$)

Inputs:

$\mathbf{A}$ is an $m \times n$ matrix and $k \ll \min\{m, n\}$ is a positive integer.

axis is an integer, equal to 1 for row ID or 2 for column ID.

Output:

A matrix $\mathbf{Z}$ and vector I satisfying $\mathbf{Y}[I, :] = (\mathbf{Z}\mathbf{Y}[I, :])[I, :]$

or

a matrix $\mathbf{X}$ and vector J satisfying $\mathbf{Y}[:, J] = (\mathbf{Y}[:, J]\mathbf{X})[:, J]$.

Abstract subroutines:

`TallSketchOpGen` and `osid_qrcp`

Tuning parameters:

s is a nonnegative integer. The algorithm internally works with a sketch of rank $k + s$.

2: **if** axis $==$ 1 **then** # row ID
3: $\mathbf{S} = \texttt{TallSketchOpGen}(\mathbf{A}, k + s)$
4: $\mathbf{Y} = \mathbf{A}\mathbf{S}$
5: $\mathbf{Z}, I = \texttt{osid\_qrcp}(\mathbf{Y}, k, \text{axis} = 0)$
6: **return** $\mathbf{Z}, I$
7: **else**
8: $\mathbf{S} = \texttt{TallSketchOpGen}(\mathbf{A}^*, k + s)^*$
9: $\mathbf{Y} = \mathbf{S}\mathbf{A}$
10: $\mathbf{X}, J = \texttt{osid\_qrcp}(\mathbf{Y}, k, \text{axis} = 1)$
11: **return** $\mathbf{X}, J$

Consider the following interface for (randomized) row and column subset selection algorithms

$$P = \texttt{RowOrColSelection}(\mathbf{A}, k, s, \text{axis}).$$

The index vector P and oversampling parameter is understood in the same way as the `OneSidedID` interface. That is, P is a partial permutation of the row index set $[\![m]\!]$ (when axis $= 1$) or the column index set $[\![n]\!]$ (when axis $= 2$). Implementations are supposed to perform internal calculations with sketches of rank $k + s$.

Algorithm 15 `ROCS1` : implements `RowOrColSelection` by QRCP on a sketch

function `ROCS1`($\mathbf{A}, k, s,$ axis)

Inputs:

$\mathbf{A}$ is an $m \times n$ matrix and $k \ll \min\{m, n\}$ is a positive integer.

axis is an integer, equal to 1 for row selection or 2 for column selection.

Output:

I: a row selection vector of length k

or

J: a column selection vector of length k.

Abstract subroutines:

`TallSketchOpGen`

Tuning parameters:

s is a nonnegative integer. The algorithm internally works with a sketch of rank $k + s$.

```
if axis == 1 then
    S = TallSketchOpGen(A, k + s)
    Y = AS
    Q, R, I = qrcp(Y*)
    return I[: k]
else
    S = TallSketchOpGen(A*, k + s)
    Y = SA
    Q, R, J = qrcp(Y)
    return J[: k]
```

Appendix D

Correctness of Preconditioned Cholesky QRCP

In this appendix we prove Proposition 5.1.2. Since this would involve a fair amount of bookkeeping if we used the notation of Algorithm 7, we begin with a more detailed statement of the algorithm.

Let $\mathbf{A}$ be $m \times n$ and $\mathbf{S}$ be $d \times m$ with $n \leq d \ll m$.

1. Compute the sketch $\mathbf{A}^{\text{sk}} = \mathbf{S}\mathbf{A}$
2. Decompose $[\mathbf{Q}^{\text{sk}}, \mathbf{R}^{\text{sk}}, J] = \texttt{qrcp}(\mathbf{A}^{\text{sk}})$
   (a) J is a permutation vector for the index set $[\![n]\!]$.
   (b) Abbreviating $\mathbf{A}^{\text{sk}}_J = \mathbf{A}^{\text{sk}}[:, J]$, we have $\mathbf{A}^{\text{sk}}_J = \mathbf{Q}^{\text{sk}}\mathbf{R}^{\text{sk}}$.
   (c) Let $k = \text{rank}(\mathbf{A}^{\text{sk}})$.
   (d) $\mathbf{Q}^{\text{sk}}$ is $m \times k$ and column-orthonormal.
   (e) $\mathbf{R}^{\text{sk}} = [\mathbf{R}^{\text{sk}}_1,\ \mathbf{R}^{\text{sk}}_2]$ is $k \times n$ upper-triangular.
   (f) $\mathbf{R}^{\text{sk}}_1$ is $k \times k$ and nonsingular.
3. Abbreviate $\mathbf{A}_J = \mathbf{A}[:, J]$ and explicitly from $\mathbf{A}^{\text{pre}} = \mathbf{A}_J[:, :k](\mathbf{R}^{\text{sk}}_1)^{-1}$.
4. Compute an unpivoted QR decomposition $\mathbf{A}^{\text{pre}} = \mathbf{Q}\mathbf{R}^{\text{pre}}$.
   (a) If $\text{rank}(\mathbf{A}) = k$ then $\mathbf{Q}$ is an orthonormal basis for the range of $\mathbf{A}$.
   (b) For the purposes of this appendix, it does not matter what algorithm we use to compute this decomposition. We assume the decomposition is exact.
5. Explicitly form $\mathbf{R} = \mathbf{R}^{\text{pre}}\mathbf{R}^{\text{sk}}$

The goal of this proof is to show that the equality $\mathbf{A}[:, J] = \mathbf{Q}\mathbf{R}$ holds under the assumption that $\text{rank}(\mathbf{S}\mathbf{A}) = \text{rank}(\mathbf{A})$. Let us first establish some useful identities. By steps 3 and 4 of the algorithm description above, we know that

$$\mathbf{R}^{\text{pre}} = \mathbf{Q}^* \mathbf{A}_J[:, :k](\mathbf{R}^{\text{sk}}_1)^{-1}.$$

Combining this with the characterization of $\mathbf{R}$ from Steps 2e and 5, we have

$$\mathbf{R} = \mathbf{Q}^* \mathbf{A}_J[:, :k](\mathbf{R}_1^{\text{sk}})^{-1}[\mathbf{R}_1^{\text{sk}},\ \mathbf{R}_2^{\text{sk}}].$$

We may further expand this expression as such:

$$\mathbf{R} = \mathbf{Q}^* \mathbf{A}_J[:, :k][\mathbf{I}_{k\times k},\ (\mathbf{R}_1^{\text{sk}})^{-1}\mathbf{R}_2^{\text{sk}}].$$

Since $\mathbf{Q}$ is an orthonormal basis for the range of $\mathbf{A}$ and, consequently, $\mathbf{A}_J$, we have that

$$\mathbf{QR} = \mathbf{A}_J[:, :k][\mathbf{I}_{k\times k},\ (\mathbf{R}_1^{\text{sk}})^{-1}\mathbf{R}_2^{\text{sk}}]. \tag{D.1}$$

We use (D.1) to establish the claim by a columnwise argument. That is, we show that $\mathbf{QR}[:, \ell] = \mathbf{A}_J[:, \ell]$ for all $1 \le \ell \le n$.

First, consider the case when $\ell \le k$. Let $\boldsymbol{\delta}_\ell^n$ be the ℓ^{th} standard basis vector in $\mathbb{R}^n$. Then, consider the following series of identities:

$$\begin{aligned}\mathbf{QR}[:, \ell] &= \mathbf{QR}\boldsymbol{\delta}_\ell^n \\ &= \mathbf{A}_J[:, :k][\mathbf{I}_{k\times k},\ (\mathbf{R}_1^{\text{sk}})^{-1}\mathbf{R}_2^{\text{sk}}]\boldsymbol{\delta}_\ell^n \\ &= \mathbf{A}_J[:, :k]\boldsymbol{\delta}_\ell^k = \mathbf{A}_J[:, \ell],\end{aligned}$$

hence the desired statement holds for $\ell \le k$.

It remains to show that $\mathbf{QR}[:, \ell] = \mathbf{A}_J[:, \ell]$ for $\ell > k$. Note that

$$\begin{aligned}\mathbf{QR}[:, \ell] &= \mathbf{A}_J[:, :k][\mathbf{I}_{k\times k},\ (\mathbf{R}_1^{\text{sk}})^{-1}\mathbf{R}_2^{\text{sk}}]\boldsymbol{\delta}_\ell^n \\ &= \mathbf{A}_J[:, :k]((\mathbf{R}_1^{\text{sk}})^{-1}\mathbf{R}_2^{\text{sk}})[:, \ell - k].\end{aligned}$$

Let $\boldsymbol{\gamma} = ((\mathbf{R}_1^{\text{sk}})^{-1}\mathbf{R}_2^{\text{sk}})[:, \ell - k]$. Therefore, in order to obtain the desired identity for $\ell > k$, we will need to show that

$$\mathbf{A}_J[:, k]\boldsymbol{\gamma} = \mathbf{A}_J[:, \ell].$$

Proposition D.0.1. *If $\mathbf{A}_J^{\text{sk}}[:, \ell] = \mathbf{A}_J^{\text{sk}}[:, :k]\boldsymbol{u}$ for some $\boldsymbol{u} \in \mathbb{R}^k$, then $\mathbf{A}_J[:, \ell] = \mathbf{A}_J[:, :k]\boldsymbol{u}$.*

Proof. To simplify notation, define the $m \times k$ matrix $\mathbf{X} = \mathbf{A}_J[:, :k]$ and the m-vector $\boldsymbol{y} = \mathbf{A}_J[:, \ell]$.

Suppose to the contrary that $\boldsymbol{y} \neq \mathbf{X}\boldsymbol{u}$ and $\mathbf{S}\boldsymbol{y} = \mathbf{SX}\boldsymbol{u}$. Then, $\mathbf{SX}\boldsymbol{u} - \mathbf{S}\boldsymbol{y} = 0$. Define $U = \ker(\mathbf{S}[\mathbf{X},\ \boldsymbol{y}])$ and $V = \ker([\mathbf{X},\ \boldsymbol{y}])$. Clearly, U contains V. Additionally, if U contains a nonzero vector that is not in V, then $\dim(U) > \dim(V)$. This would further imply that $\operatorname{rank}(\mathbf{S}[\mathbf{X},\ \boldsymbol{y}]) < \operatorname{rank}([\mathbf{X},\ \boldsymbol{y}])$.

If $\mathbf{SX}\boldsymbol{u} - \mathbf{S}\boldsymbol{y} = 0$, then $(\boldsymbol{u}, -1)$ is a nonzero vector in U that is not in V. However, by our assumption, the sketch does not drop rank. Consequently, no such vector $(\boldsymbol{u}, -1)$ can exist, and we must have $\boldsymbol{y} = \mathbf{X}\boldsymbol{u}$. □

We now prove that $\mathbf{A}_J^{\text{sk}}[:, :k]\boldsymbol{\gamma} = \mathbf{A}_J^{\text{sk}}[:, \ell]$. To do this, start by noting that $\mathbf{A}_J^{\text{sk}}[:, :k] = \mathbf{Q}^{\text{sk}}\mathbf{R}_1^{\text{sk}}$. Plugging in the definition of $\boldsymbol{\gamma}$, we have

$$\mathbf{A}_J^{\text{sk}}[:, :k]\boldsymbol{\gamma} = \mathbf{Q}^{\text{sk}}\mathbf{R}_1^{\text{sk}}(\mathbf{R}_1^{\text{sk}})^{-1}(\mathbf{R}_2^{\text{sk}})[:, \ell - k] = \mathbf{Q}^{\text{sk}}(\mathbf{R}_2^{\text{sk}})[:, \ell - k].$$

The next step is to use the simple observation that $\mathbf{R}_2^{\text{sk}}[:, \ell - k] = \mathbf{R}^{\text{sk}}[:, \ell]$ to find

$$\mathbf{A}_J^{\text{sk}}[:, :k]\boldsymbol{\gamma} = (\mathbf{Q}^{\text{sk}}\mathbf{R}^{\text{sk}})[:, \ell] = \mathbf{A}_J^{\text{sk}}[:, \ell].$$

Combining the above results and Proposition D.0.1 proves Proposition 5.1.2.

Appendix E

Bootstrap Methods for Error Estimation

Whenever a randomized algorithm produces a solution, a question immediately arises: Is the solution sufficiently accurate? In many situations, it is possible to estimate numerically the error of the solution using the available problem data — a process that is often referred to as *(a posteriori) error estimation*.[1] In addition to resolving uncertainty about the quality of a solution, another key benefit of error estimation is that it enables computations to be done more adaptively. For instance, error estimates can be used to determine if additional iterations should be performed, or if tuning parameters should be modified. In this way, error estimates can help to incrementally refine a rough initial solution so that "just enough" work is done to reach a desired level of accuracy.

In this appendix, we provide a brief overview of *bootstrap methods* for error estimation in RandNLA. Up to now, these tools (which are common in statistics and statistical data analysis) have been designed for a handful of sketch-and-solve type algorithms, and the development of bootstrap methods for a wider range of randomized algorithms is an open direction of research. Our main purpose in writing this appendix is to record the consideration we have given to bootstrap methods. Our secondary purpose is to provide a starting point for non-experts to survey this literature as it evolves.

[1] This should be contrasted with *(a priori) error bounds* often used in theoretical development of RandNLA algorithms, in which one bounds rather than estimates the error, and does so in a worst-case way that does not depend on the problem data.

E.1 Bootstrap methods in a nutshell

Bootstrap methods have been studied extensively in the statistics literature for more than four decades, and they comprise a very general framework for quantifying uncertainty [ET94; ST12]. One of the most common uses of these methods in statistics is to assess the accuracy of parameter estimates. This use-case provides the connection between bootstrap methods and error estimation in RandNLA. Indeed, an exact solution to a linear algebra problem can be viewed as an "unknown parameter," and a randomized algorithm can be viewed as providing an "estimate" of that parameter. Taking the analogy a step further, a random sketch of a matrix can also be viewed as a "dataset" from which the estimate of the "population" quantity is computed. Likewise, when bootstrap methods are applied in RandNLA, the rows or columns of a sketched matrix often play the role of "data vectors".

We now formulate the task of error estimation in a way that is convenient for discussion of bootstrap methods. First, suppose the existence of some fixed but unknown "true parameter" $\theta \in \mathbb{R}$. Suppose we estimate this parameter by a value $\hat{\theta}$ depending on random samples from some probability distribution. The error of $\hat{\theta}$ is defined as $\hat{\epsilon} = |\hat{\theta} - \theta|$, which we emphasize is both random and unknown. From this standpoint, it is natural to seek the tightest upper bound on $\hat{\epsilon}$ that holds with a specified probability, say $1 - \alpha$. This ideal bound is known as the $(1 - \alpha)$-quantile of $\hat{\epsilon}$, and is defined more formally as

$$q_{1-\alpha} = \inf\{t \in [0, \infty) \,|\, \mathbb{P}(\hat{\epsilon} \leq t) \geq 1 - \alpha\}.$$

An error estimation problem is considered solved if it is possible to construct a quantile estimate $\hat{q}_{1-\alpha}$ such that the inequality $\hat{\epsilon} \leq \hat{q}_{1-\alpha}$ holds with probability that is close to $1 - \alpha$.

The bootstrap approach to estimating $q_{1-\alpha}$ is based on imagining a scenario where it is possible to generate many independent samples $\hat{\hat{\epsilon}}_1, \ldots, \hat{\hat{\epsilon}}_N$ of the random variable $\hat{\epsilon}$. Of course, this is not possible in practice, but if it were, then an estimate of $q_{1-\alpha}$ could be easily obtained using the empirical $(1 - \alpha)$-quantile of the samples $\hat{\hat{\epsilon}}_1, \ldots, \hat{\hat{\epsilon}}_N$. The key idea that bootstrap methods use to circumvent the difficulty is to generate "approximate samples" of $\hat{\epsilon}$, which *can* be done in practice.

To illustrate how approximate samples of $\hat{\epsilon}$ can be constructed, consider a generic situation where the estimate $\hat{\theta}$ is computed as a function of a dataset $X_1, \ldots, X_n$. That is, suppose $\hat{\theta} = f(X_1, \ldots, X_n)$ for some function f. Then, a *bootstrap sample* of $\hat{\epsilon}$, denoted $\hat{\hat{\epsilon}}$, is computed as follows:

- Sample n points $\{\hat{\hat{X}}_i\}_{i=1}^n$ with replacement from the original dataset $\{X_i\}_{i=1}^n$.
- Compute $\hat{\hat{\theta}} := f(\hat{\hat{X}}_1, \ldots, \hat{\hat{X}}_n)$
- Compute $\hat{\hat{\epsilon}} := |\hat{\hat{\theta}} - \hat{\theta}|$.

By performing N independent iterations of this process, a collection of bootstrap samples $\hat{\hat{\epsilon}}_1, \ldots, \hat{\hat{\epsilon}}_N$ can be generated. Then, the desired quantile estimate $\hat{q}_{1-\alpha}$ can be computed as the smallest number $t \geq 0$ for which the inequality

$$\frac{1}{N} \sum_{i=1}^{N} \mathbb{I}\{\hat{\hat{\epsilon}}_i \leq t\} \geq 1 - \alpha$$

is satisfied, where $\mathbb{I}\{\cdot\}$ refers to the $\{0,1\}$-valued indicator function. This quantity is also known as the empirical $(1-\alpha)$-quantile of $\hat{\hat{\epsilon}}_1, \dots, \hat{\hat{\epsilon}}_n$. We will sometimes denote it by $\text{quantile}[\hat{\hat{\epsilon}}_1, \dots, \hat{\hat{\epsilon}}_n; 1-\alpha]$.

To provide some intuition for the bootstrap, the random variable $\hat{\hat{\theta}}$ can be viewed as a "perturbed version" of $\hat{\theta}$, where the perturbing mechanism is designed so that the deviations of $\hat{\hat{\theta}}$ around $\hat{\theta}$ are statistically similar to the deviations of $\hat{\theta}$ around θ [ET94]. Equivalently, this means that the histogram of $\hat{\hat{\epsilon}}_1, \dots, \hat{\hat{\epsilon}}_N$ will serve as a good approximation to the distribution of the actual random error variable $\hat{\epsilon}$. Furthermore, it turns out that this approximation is asymptotically valid (i.e., $n \to \infty$) and supported by quantitative guarantees in a broad range of situations [ST12].

E.2 Sketch-and-solve least squares

There is a direct analogy between the discussion above and the setting of sketch-and-solve algorithms for least squares. First, the "true parameter" θ is the exact solution $\boldsymbol{x}_\star = \text{argmin}_{\boldsymbol{x}\in\mathbb{R}^n} \|\mathbf{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2$. Second, the dataset $X_1, \dots, X_n$ corresponds to the sketches $[\hat{\mathbf{A}}, \hat{\boldsymbol{b}}] = \mathbf{S}[\mathbf{A}, \boldsymbol{b}]$. Third, the estimate $\hat{\theta}$ corresponds to the sketch-and-solve solution $\hat{\boldsymbol{x}} = \text{argmin}_{\boldsymbol{x}\in\mathbb{R}^n} \|\hat{\mathbf{A}}\boldsymbol{x} - \hat{\boldsymbol{b}}\|_2^2$. Fourth, the error variable can be defined as $\hat{\epsilon} = \rho(\hat{\boldsymbol{x}}, \boldsymbol{x}_\star)$, for a preferred metric ρ, such as that induced by the ℓ_2 or ℓ_∞ norms.

Once these correspondences are recognized, the previous bootstrap sampling scheme can be applied. For further background, as well as extensions to error estimation for iterative randomized algorithms for least squares, we refer to [LWM18].

Method 1 (Bootstrap error estimation for sketch-and-solve least squares).

Input: A positive integer B, the sketches $\hat{\mathbf{A}} \in \mathbb{R}^{d\times n}$, $\hat{\boldsymbol{b}} \in \mathbb{R}^d$, and $\hat{\boldsymbol{x}} \in \mathbb{R}^n$.

For $\ell \in [\![B]\!]$ **do in parallel**

1. Draw a vector $I := (i_1, \dots, i_d)$ by sampling d numbers with replacement from $[\![d]\!]$.
2. Form the matrix $\hat{\hat{\mathbf{A}}} := \hat{\mathbf{A}}[I, :]$, and vector $\hat{\hat{\boldsymbol{b}}} := \hat{\boldsymbol{b}}[I]$.
3. Compute the following vector and scalar,

$$\hat{\hat{\boldsymbol{x}}} := \operatorname*{arg\,min}_{\boldsymbol{x}\in\mathbb{R}^n} \|\hat{\hat{\mathbf{A}}}\boldsymbol{x} - \hat{\hat{\boldsymbol{b}}}\|_2 \qquad \text{and} \qquad \hat{\hat{\epsilon}}_\ell := \|\hat{\hat{\boldsymbol{x}}} - \hat{\boldsymbol{x}}\|. \tag{E.1}$$

Return: The estimate $\text{quantile}[\hat{\hat{\epsilon}}_1, \dots, \hat{\hat{\epsilon}}_B; 1-\alpha]$ for the $(1-\alpha)$-quantile of $\|\hat{\boldsymbol{x}} - \boldsymbol{x}_\star\|$.

To briefly comment on some of the computational characteristics of this method, it should be emphasized that the for loop can be implemented in an embarrassingly parallel manner, which is typical of most bootstrap methods. Second, the method only relies on access to sketched quantities, and hence does not require any access to the full matrix $\mathbf{A}$. Likewise, the computational cost of the method is independent of the number of rows of $\mathbf{A}$.

E.3 Sketch-and-solve one-sided SVD

We call the problem of computing the singular values and right singular vectors of a matrix a "one-sided SVD." We further use the term "sketch-and-solve one-sided SVD" for an algorithm that approximates the top k singular values and singular vectors of $\mathbf{A}$ by those of a sketch $\hat{\mathbf{A}} = \mathbf{SA}$. Here we consider estimating the error incurred by such an algorithm. As matters of notation, we let $\{(\sigma_j, \boldsymbol{v}_j)\}_{j=1}^k$ denote the top k singular values and right singular vectors of $\mathbf{A}$ and $\{(\hat{\sigma}_j, \hat{\boldsymbol{v}}_j)\}_{j=1}^k$ the corresponding quantities for $\hat{\mathbf{A}}$. We suppose that error is measured uniformly over $j \in [\![k]\!]$, which leads us to consider error variables of the form

$$\epsilon_{\Sigma} := \max_{j \in [\![k]\!]} |\hat{\sigma}_j - \sigma_j| \quad \text{and} \quad \epsilon_{V} := \max_{j \in [\![k]\!]} \rho(\hat{\boldsymbol{v}}_j, \boldsymbol{v}_j).$$

The following bootstrap method, developed in [LEM20], provides estimates for the $(1-\alpha)$-quantiles of ϵ_{Σ} and ϵ_{V}.

Method 2 (Bootstrap error estimation for sketch-and-solve SVD).

Input: The sketch $\hat{\mathbf{A}} \in \mathbb{R}^{d \times n}$ and its top k singular values and right singular vectors $(\hat{\sigma}_1, \hat{\boldsymbol{v}}_1), \ldots, (\hat{\sigma}_k, \hat{\boldsymbol{v}}_k)$, a number of samples B, a parameter $\alpha \in (0, 1)$.

- **For** $\ell \in [\![B]\!]$ **do in parallel**

  1. Form $\hat{\hat{\mathbf{A}}} \in \mathbb{R}^{d \times n}$ by sampling d rows from $\hat{\mathbf{A}}$ with replacement.
  2. Compute the top k singular values and right singular vectors of $\hat{\hat{\mathbf{A}}}$, denoted as $\hat{\hat{\sigma}}_1, \ldots, \hat{\hat{\sigma}}_k$ and $\hat{\hat{\boldsymbol{v}}}_1, \ldots, \hat{\hat{\boldsymbol{v}}}_k$. Then, compute the bootstrap samples

$$\hat{\hat{\epsilon}}_{\mathbf{\Sigma},\ell} := \max_{j \in [\![k]\!]} |\hat{\hat{\sigma}}_j - \hat{\sigma}_j| \tag{E.2}$$

$$\hat{\hat{\epsilon}}_{\mathbf{V},\ell} := \max_{j \in [\![k]\!]} \rho(\hat{\hat{\boldsymbol{v}}}_j, \hat{\boldsymbol{v}}_j). \tag{E.3}$$

Return: The estimates quantile$[\hat{\hat{\epsilon}}_{\mathbf{\Sigma},1}, \ldots, \hat{\hat{\epsilon}}_{\mathbf{\Sigma},B}; 1-\alpha]$ and quantile$[\hat{\hat{\epsilon}}_{\mathbf{V},1}, \ldots, \hat{\hat{\epsilon}}_{\mathbf{V},B}; 1-\alpha]$ for the $(1-\alpha)$-quantiles of ϵ_{Σ} and ϵ_{V}.

Although this method is only presented with regard to singular values and right singular vectors, it is also possible to apply a variant of it to estimate the errors of approximate left singular vectors. However, a few extra technical details are involved, which may be found in [LEM20].

Another technique to estimate error in the setting of sketch-and-solve one-sided SVD is through the spectral norm $\|\hat{\mathbf{A}}^* \hat{\mathbf{A}} - \mathbf{A}^* \mathbf{A}\|_2$. Due to the Weyl and Davis-Kahan inequalities, an upper bound on $\|\hat{\mathbf{A}}^* \hat{\mathbf{A}} - \mathbf{A}^* \mathbf{A}\|_2$ directly implies upper bounds on the errors of all the sketched singular values $\hat{\sigma}_1, \ldots, \hat{\sigma}_n$ and sketched right singular vectors $\hat{\boldsymbol{v}}_1, \ldots, \hat{\boldsymbol{v}}_n$. Furthermore, the quantiles of the error variable $\|\hat{\mathbf{A}}^* \hat{\mathbf{A}} - \mathbf{A}^* \mathbf{A}\|_2$ can be estimated via the bootstrap, as shown in [LEM23].

Bibliography

[AAB+17] A. Abdelfattah, H. Anzt, A. Bouteiller, A. Danalis, J. Dongarra, M. Gates, A. Haidar, J. Kurzak, P. Luszczek, S. Tomov, S. Wood, P. Wu, I. Yamazaki, and A. YarKhan. *Roadmap for the Development of a Linear Algebra Library for Exascale Computing: SLATE: Software for Linear Algebra Targeting Exascale*. SLATE Working Notes 01, ICL-UT-17-02. June 2017.

[ABB+99] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Jan. 1999.

[AC06] N. Ailon and B. Chazelle. "Approximate nearest neighbors and the Fast Johnson-Lindenstrauss Transform". In: *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing (STOC)*. STOC '06. Seattle, WA, USA: Association for Computing Machinery, 2006, pp. 557–563. ISBN: 1595931341.

[AC09] N. Ailon and B. Chazelle. "The Fast Johnson–Lindenstrauss Transform and approximate nearest neighbors". In: *SIAM Journal on Computing* 39.1 (Jan. 2009), pp. 302–322.

[Ach03] D. Achlioptas. "Database-friendly random projections: Johnson-Lindenstrauss with binary coins". In: *Journal of Computer and System Sciences* 66.4 (2003), pp. 671–687.

[ACW17a] H. Avron, K. L. Clarkson, and D. P. Woodruff. "Sharper bounds for regularized data fitting". In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques* (2017).

[ACW17b] H. Avron, K. L. Clarkson, and D. P. Woodruff. "Faster kernel ridge regression using sketching and preconditioning". In: *SIAM Journal on Matrix Analysis and Applications* 38.4 (2017), pp. 1116–1138.

[AD20] N. Anari and M. Dereziński. "Isotropy and log-concave polynomials: accelerated sampling and high-precision counting of matroid bases". In: *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE. 2020, pp. 1331–1344.

[ADD+09] E. Agullo, J. Demmel, J. Dongarra, B. Hadri, J. Kurzak, J. Langou, H. Ltaief, P. Luszczek, and S. Tomov. "Numerical linear algebra on emerging architectures: the PLASMA and MAGMA projects". In: *Journal of Physics: Conference Series* 180 (July 2009), p. 012037.

[ADN20] P. Ahren, J. Demmel, and H.-D. Nguyen. "Algorithms for efficient reproducible floating point summation". In: *ACM Transactions on Mathematical Software* 46.3 (2020).

[ADR92] M. Arioli, I. Duff, and D. Ruiz. "Stopping criteria for iterative solvers". In: *SIAM Journal on Matrix Analysis and Applications* 13.1 (Jan. 1992), pp. 138–144.

[ADV+22] N. Anari, M. Dereziński, T.-D. Vuong, and E. Yang. "Domain sparsification of discrete distributions using entropic independence". In: *ACM Symposium on Discrete Algorithms (SODA)*. 2022.

[AGL98] C. Ashcraft, R. G. Grimes, and J. G. Lewis. "Accurate symmetric indefinite linear equation solvers". In: *SIAM Journal on Matrix Analysis and Applications* 20.2 (Jan. 1998), pp. 513–561.

[AGR16] N. Anari, S. O. Gharan, and A. Rezaei. "Monte carlo markov chain algorithms for sampling strongly rayleigh distributions and determinantal point processes". In: *Conference on Learning Theory (COLT)*. PMLR. 2016, pp. 103–115.

[AK01] O. Axelsson and I. Kaporin. "Error norm estimation and stopping criteria in preconditioned conjugate gradient iterations". In: *Numerical Linear Algebra with Applications* 8.4 (2001), pp. 265–286.

[AKK+20] T. Ahle, M. Kapralov, J. Knudsen, R. Pagh, A. Velingker, D. Woodruff, and A. Zandieh. "Oblivious sketching of high-degree polynomial kernels". In: *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2020, pp. 141–160.

[AM15] A. E. Alaoui and M. W. Mahoney. "Fast randomized kernel methods with statistical guarantees". In: *Annual Advances in Neural Information Processing Systems*. 2015, pp. 775–783.

[Ame22] S. Ameli. *IMATE, a high-performance python package for implicit matrix trace estimation*. https://pypi.org/project/imate/. 2022.

[AMT10] H. Avron, P. Maymounkov, and S. Toledo. "Blendenpik: Supercharging LAPACK's Least-Squares Solver". In: *SIAM Journal on Scientific Computing* 32.3 (Jan. 2010), pp. 1217–1236.

[ANW14] H. Avron, H. Nguyen, and D. Woodruff. "Subspace embeddings for the polynomial kernel". In: *Advances in Neural Information Processing Systems*. Vol. 27. Curran Associates, Inc., 2014.

[Aro50] N. Aronszajn. "Theory of reproducing kernels". In: *Transactions of the American mathematical society* 68.3 (1950), pp. 337–404.

[Bac13] F. Bach. "Sharp analysis of low-rank kernel matrix approximations". In: *Proceedings of the 26th Annual Conference on Learning Theory (COLT)*. 2013, pp. 185–209.

[Bal22a] O. Balabanov. *randKrylov, a MATLAB library for linear systems and eigenvalue problems*. https://github.com/obalabanov/randKrylov. 2022.

[Bal22b] O. Balabanov. *Randomized Cholesky QR factorizations*. 2022. arXiv: 2210.09953.

[BBB+14] M. Baboulin, D. Becker, G. Bosilca, A. Danalis, and J. Dongarra. "An efficient distributed randomized algorithm for solving large dense symmetric indefinite linear systems". In: *Parallel Computing* 40.7 (July 2014), pp. 213–223.

[BBB15] D. J. Biagioni, D. Beylkin, and G. Beylkin. "Randomized interpolative decomposition of separated representations". In: *Journal of Computational Physics* 281 (2015), pp. 116–134.

[BBG+22] O. Balabanov, M. Beaupere, L. Grigori, and V. Lederer. *Block subsampled randomized Hadamard transform for low-rank approximation on distributed architectures*. 2022. arXiv: 2210.11295.

[BBK18] C. Battaglino, G. Ballard, and T. G. Kolda. "A practical randomized CP tensor decomposition". In: *SIAM Journal on Matrix Analysis and Applications* 39.2 (2018), pp. 876–901.

[BDH+13] M. Baboulin, J. Dongarra, J. Herrmann, and S. Tomov. "Accelerating linear system solutions using randomization techniques". In: *ACM Trans. Math. Softw.* 39.2 (Feb. 2013).

[BDN15] J. Bourgain, S. Dirksen, and J. Nelson. "Toward a unified theory of sparse dimensionality reduction in Euclidean space". In: *Geometric and Functional Analysis* 25.4 (July 2015), pp. 1009–1088.

[BDR+17] M. Baboulin, J. Dongarra, A. Rémy, S. Tomov, and I. Yamazaki. "Solving dense symmetric indefinite systems using GPUs". In: *Concurrency and Computation: Practice and Experience* 29.9 (2017). e4055 cpe.4055, e4055.

[BFG96] Z. Bai, G. Fahey, and G. Golub. "Some large-scale matrix computation problems". In: *Journal of Computational and Applied Mathematics* 74.1 (1996), pp. 71–89.

[BG13] C. Boutsidis and A. Gittens. "Improved matrix algorithms via the subsampled randomized Hadamard transform". In: *SIAM Journal on Matrix Analysis and Applications* 34.3 (Jan. 2013), pp. 1301–1340.

[BG21] O. Balabanov and L. Grigori. *Randomized block Gram-Schmidt process for solution of linear systems and eigenvalue problems*. 2021. arXiv: 2111.14641.

[BG22] O. Balabanov and L. Grigori. "Randomized Gram–Schmidt process with application to GMRES". In: *SIAM Journal on Scientific Computing* 44.3 (2022), A1450–A1474.

[BG65] P. Businger and G. H. Golub. "Linear least squares solutions by householder transformations". In: *Numerische Mathematik* 7.3 (June 1965), pp. 269–276.

[BGL05] M. Benzi, G. H. Golub, and J. Liesen. "Numerical solution of saddle point problems". In: *Acta Numerica* 14 (2005), pp. 1–137.

[Bha97] R. Bhatia. *Matrix Analysis*. Springer New York, 1997.

[Bja19] E. K. Bjarkason. "Pass-efficient randomized algorithms for low-rank matrix approximation using any number of views". In: *SIAM Journal on Scientific Computing* 41.4 (Jan. 2019), A2355–A2383.

[Bjö15] Å. Björck. *Numerical Methods in Matrix Computations*. Vol. 59. 2015. ISBN: 978-3-319-05088-1.

[Bjö96] Å. Björck. *Numerical Methods for Least Squares Problems*. Society for Industrial and Applied Mathematics, Jan. 1996.

[BK21] Z. Bujanovic and D. Kressner. "Norm and trace estimation with random rank-one vectors". In: *SIAM Journal on Matrix Analysis and Applications* 42.1 (2021), pp. 202–223.

[BK77] J. R. Bunch and L. Kaufman. "Some stable methods for calculating inertia and solving symmetric linear systems". In: *Mathematics of Computation* 31.137 (1977), pp. 163–179.

[BKW21] S. Bamberger, F. Krahmer, and R. Ward. *Johnson–Lindenstrauss Embeddings with Kronecker Structure*. 2021. arXiv: 2106.13349.

[BLR14] M. Baboulin, X. S. Li, and F. Rouet. "Using random butterfly transformations to avoid pivoting in sparse direct methods". In: *High Performance Computing for Computational Science - VECPAR*. Ed. by M. J. Daydé, O. Marques, and K. Nakajima. Vol. 8969. Lecture Notes in Computer Science. Springer, 2014, pp. 135–144.

[BM58] G. E. P. Box and M. E. Muller. "A note on the generation of random normal deviates". In: *The Annals of Mathematical Statistics* 29.2 (1958), pp. 610–611.

[BMD09] C. Boutsidis, M. W. Mahoney, and P. Drineas. "An improved approximation algorithm for the column subset selection problem". In: *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2009, pp. 968–977.

[BMM+22] V. Bharadwaj, O. A. Malik, R. Murray, A. Buluç, and J. Demmel. *Distributed-Memory Randomized Algorithms for Sparse Tensor CP Decomposition*. 2022. arXiv: 2210.05105.

[Boh07] M. Bohr. "A 30 year retrospective on Dennard's MOSFET scaling paper". In: *Solid-State Circuits Newsletter, IEEE* 12.1 (2007), pp. 11–13.

[BV21] D. Blackman and S. Vigna. "Scrambled linear pseudorandom number generators". In: *ACM Trans. Math. Softw.* 47.4 (Sept. 2021). Software available at https://prng.di.unimi.it/.

[CDD+96] J. Choi, J. Demmel, I. Dhillon, J. Dongarra, S. Ostrouchov, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. "ScaLAPACK: a portable linear algebra library for distributed memory computers—design issues and performance". In: *Computer Physics Communications* 97.1-2 (1996), pp. 1–15.

[CDO+95] J. Choi, J. Dongarra, S. Ostrouchov, A. Petitet, D. W. Walker, and R. C. Whaley. "A proposal for a set of parallel basic linear algebra subprograms". In: *Proceedings of the Second International Workshop on Applied Parallel Computing, Computations in Physics, Chemistry and Engineering Science*. PARA '95. Berlin, Heidelberg: Springer-Verlag, 1995, pp. 107–114. ISBN: 3540609024.

[CDV20] D. Calandriello, M. Dereziński, and M. Valko. "Sampling from a k-DPP without looking at all items". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 6889–6899.

[CET+22] Y. Chen, E. N. Epperly, J. A. Tropp, and R. J. Webber. *Randomly pivoted Cholesky: Practical approximation of a kernel matrix with few entry evaluations*. 2022. arXiv: 2207.06503.

[CFG95] M. T. Chu, R. E. Funderlic, and G. H. Golub. "A rank–one reduction formula and its applications to matrix factorizations". In: *SIAM Review* 37.4 (Dec. 1995), pp. 512–530.

[CFS21] C. Cartis, J. Fiala, and Z. Shao. *Hashing embeddings of optimal dimension, with applications to linear least squares*. 2021. arXiv: 2105.11815.

[CH22] T. Chen and E. Hallman. *Krylov-aware stochastic trace estimation*. 2022. arXiv: 2205.01736.

[CH88] S. Chatterjee and A. Hadi. *Sensitivity Analysis in Linear Regression*. New York: John Wiley & Sons, 1988.

[CKN22] A. Cortinovis, D. Kressner, and Y. Nakatsukasa. *Speeding up Krylov subspace methods for computing f(A)b via randomization*. 2212.12758. 2022.

[CLA+20] A. Chowdhury, P. London, H. Avron, and P. Drineas. "Faster randomized infeasible interior point methods for tall/wide linear programs". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 8704–8715.

[CLN+20] K. Chen, Q. Li, K. Newton, and S. J. Wright. "Structured random sketching for PDE inverse problems". In: *SIAM Journal on Matrix Analysis and Applications* 41.4 (2020), pp. 1742–1770.

[CLV17] D. Calandriello, A. Lazaric, and M. Valko. "Distributed adaptive sampling for kernel matrix approximation". In: *Artificial Intelligence and Statistics*. PMLR. 2017, pp. 1421–1429.

[ÇM09] A. Çivril and M. Magdon-Ismail. "On selecting a maximum volume submatrix of a matrix and related problems". In: *Theoretical Computer Science* 410.47-49 (Nov. 2009), pp. 4801–4811.

[CMD+15] A. Cichocki, D. Mandic, L. De Lathauwer, G. Zhou, Q. Zhao, C. Caiafa, and H. A. Phan. "Tensor decompositions for signal processing applications: from two-way to multiway component analysis". In: *IEEE Signal Processing Magazine* 32.2 (2015), pp. 145–163.

[CMX+22] N. Cheng, O. A. Malik, Y. Xu, S. Becker, A. Doostan, and A. Narayan. *Quadrature Sampling of Parametric Models with Bi-fidelity Boosting*. 2022. arXiv: 2209.05705.

[Coh16] M. B. Cohen. "Nearly tight oblivious subspace embeddings by trace inequalities". In: *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. Society for Industrial and Applied Mathematics, Dec. 2016.

[CP15] M. B. Cohen and R. Peng. "Lp row sampling by lewis weights". In: *Proceedings of the forty-seventh annual ACM Symposium on Theory of Computing (STOC)*. 2015, pp. 183–192.

[CPL+16] D. Cheng, R. Peng, Y. Liu, and I. Perros. "SPALS: fast alternating least squares via implicit leverage scores sampling". In: *Advances in Neural Information Processing Systems*. 2016, pp. 721–729.

[CTU22] T. Chen, T. Trogdon, and S. Ubaru. *Randomized matrix-free quadrature for spectrum and spectral sum approximation*. 2022. arXiv: 2204.01941.

[CW09] K. L. Clarkson and D. P. Woodruff. "Numerical linear algebra in the streaming model". In: *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing (STOC)*. STOC '09. Bethesda, MD, USA: Association for Computing Machinery, 2009, pp. 205–214. ISBN: 9781605585062.

[CW13] K. L. Clarkson and D. P. Woodruff. "Low rank approximation and regression in input sparsity time". In: *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing (STOC)*. Palo Alto, California, USA: Association for Computing Machinery, 2013, pp. 81–90. ISBN: 9781450320290.

[CW17] K. L. Clarkson and D. P. Woodruff. "Low-rank approximation and regression in input sparsity time". In: *J. ACM* 63.6 (Jan. 2017). This is the journal version of a 2013 STOC article by the same name.

[DB08] Z. Drmač and Z. Bujanović. "On the failure of rank-revealing qr factorization software – a case study". In: *ACM Trans. Math. Softw.* 35.2 (July 2008).

[DCM+19] M. Dereziński, K. L. Clarkson, M. W. Mahoney, and M. K. Warmuth. "Minimax experimental design: bridging the gap between statistical and worst-case approaches to least squares regression". In: *Conference on Learning Theory (COLT)*. PMLR. 2019, pp. 1050–1069.

[DCV19] M. Dereziński, D. Calandriello, and M. Valko. "Exact sampling of determinantal point processes with sublinear time preprocessing". In: *Advances in Neural Information Processing Systems* 32 (2019).

[DDD+87] J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, and D. Sorensen. *Prospectus for the development of a linear algebra library for high-performance computers*. https://netlib.org/lapack/lawns/. LAPACK Working Note 01. Sept. 1987.

[DDG+22] J. Demmel, J. Dongarra, M. Gates, G. Henry, J. Langou, X. Li, P. Luszczek, W. Pereira, J. Riedy, and C. Rubio-González. *Proposed Consistent Exception Handling for the BLAS and LAPACK*. 2022. arXiv: 2207.09281.

[DDH+09] A. Dasgupta, P. Drineas, B. Harb, R. Kumar, and M. W. Mahoney. "Sampling algorithms and coresets for ℓ_p regression". In: *SIAM Journal on Computing* 38 (2009), pp. 2060–2078.

[DDH+88] J. J. Dongarra, J. Du Croz, S. Hammarling, and R. J. Hanson. "An extended set of fortran basic linear algebra subprograms". In: *ACM Trans. Math. Softw.* 14.1 (Mar. 1988), pp. 1–17.

[DDH+90] J. J. Dongarra, J. Du Croz, S. Hammarling, and I. S. Duff. "A set of level 3 basic linear algebra subprograms". In: *ACM Trans. Math. Softw.* 16.1 (Mar. 1990), pp. 1–17.

[DDH07] J. Demmel, I. Dumitriu, and O. Holtz. "Fast linear algebra is stable". In: *Numerische Mathematik* 108.1 (Oct. 2007), pp. 59–91.

[DDL+20] J. Demmel, J. Dongarra, J. Langou, J. Langou, P. Luszczek, and M. W. Mahoney. *Prospectus for the Next LAPACK and ScaLAPACK Libraries: Basic ALgebra LIbraries for Sustainable Technology with Interdisciplinary Collaboration (BALLISTIC)*. http://www.netlib.org/lapack/lawnspdf/lawn297.pdf. July 2020.

[DDM01] J. Demmel, B. Diament, and G. Malajovich. "On the complexity of computing error bounds". In: *Foundations of Computational Mathematics* 1.1 (Jan. 2001), pp. 101–125.

[Dem92] J. Demmel. "The componentwise distance to the nearest singular matrix". In: *SIAM Journal on Matrix Analysis and Applications* 13.1 (Jan. 1992), pp. 10–19.

[Der19] M. Dereziński. "Fast determinantal point processes via distortion-free intermediate sampling". In: *Conference on Learning Theory (COLT)*. PMLR. 2019, pp. 1029–1049.

[Der22a] M. Dereziński. *Algorithmic Gaussianization through Sketching: Converting Data into Sub-gaussian Random Designs*. 2022. eprint: 2206.10291.

[Der22b] M. Dereziński. *Stochastic Variance-Reduced Newton: Accelerating Finite-Sum Minimization with Large Batches*. 2022. arXiv: 2206.02702.

[DG03] S. Dasgupta and A. Gupta. "An elementary proof of a theorem of Johnson and Lindenstrauss". In: *Random Structures and Algorithms* 22.1 (2003), pp. 60–65.

[DG17] J. A. Duersch and M. Gu. "Randomized QR with column pivoting". In: *SIAM Journal on Scientific Computing* 39.4 (Jan. 2017), pp. C263–C291.

[DGG+15] J. Demmel, L. Grigori, M. Gu, and H. Xiang. "Communication-avoiding rank-revealing QR decomposition". In: *SIAM Journal on Matrix Analysis and its Applications* 36.1 (2015), pp. 55–89.

[DGH+19] J. Dongarra, M. Gates, A. Haidar, J. Kurzak, P. Luszczek, P. Wu, I. Yamazaki, A. Yarkhan, M. Abalenkovs, N. Bagherpour, S. Hammarling, J. Šístek, D. Stevens, M. Zounon, and S. D. Relton. "PLASMA". In: *ACM Transactions on Mathematical Software* 45.2 (June 2019), pp. 1–35.

[DGR+74] R. Dennard, F. Gaensslen, V. Rideout, E. Bassous, and A. LeBlanc. "Design of ion-implanted mosfet's with very small physical dimensions". In: *Solid-State Circuits, IEEE Journal of* 9.5 (Oct. 1974), pp. 256–268.

[DGR19] J. Demmel, L. Grigori, and A. Rusciano. *An improved analysis and unified perspective on deterministic and randomized low rank matrix approximations*. 2019. arXiv: 1910.00223.

[DHK+06] J. Demmel, Y. Hida, W. Kahan, X. S. Li, S. Mukherjee, and E. J. Riedy. "Error bounds from extra-precise iterative refinement". In: *ACM Trans. Math. Softw.* 32.2 (June 2006), pp. 325–351.

[Dix83] J. D. Dixon. "Estimating extremal eigenvalues and condition numbers of matrices". In: *SIAM Journal on Numerical Analysis* 20.4 (1983), pp. 812–814.

[DJS+19] H. Diao, R. Jayaram, Z. Song, W. Sun, and D. P. Woodruff. *Optimal Sketching for Kronecker Product Regression and Low Rank Approximation*. 2019. arXiv: 1909.13384.

[DKM06a] P. Drineas, R. Kannan, and M. W. Mahoney. "Fast Monte Carlo algorithms for matrices I: approximating matrix multiplication". In: *SIAM Journal on Computing* 36 (2006), pp. 132–157.

[DKM06b] P. Drineas, R. Kannan, and M. W. Mahoney. "Fast Monte Carlo algorithms for matrices II: computing a low-rank approximation to a matrix". In: *SIAM Journal on Computing* 36 (2006), pp. 158–183.

[DKM20] M. Dereziński, R. Khanna, and M. W. Mahoney. "Improved guarantees and a multiple-descent curve for Column Subset Selection and the Nyström method". In: *Annual Advances in Neural Information Processing Systems*. 2020, pp. 4953–4964.

[DKS10] A. Dasgupta, R. Kumar, and T. Sarlos. "A sparse Johnson-Lindenstrauss transform". In: *Proceedings of the Forty-Second ACM Symposium on Theory of Computing (STOC)*. STOC '10. Cambridge, Massachusetts, USA: Association for Computing Machinery, 2010, pp. 341–350. ISBN: 9781450300506.

[DLD+21] M. Dereziński, Z. Liao, E. Dobriban, and M. Mahoney. "Sparse sketches with small inversion bias". In: *Conference on Learning Theory (COLT)*. PMLR. 2021, pp. 1467–1510.

[DLL+20] M. Dereziński, F. T. Liang, Z. Liao, and M. W. Mahoney. "Precise expressions for random projections: low-rank approximation and randomized Newton". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 18272–18283.

[DLP+21] M. Dereziński, J. Lacotte, M. Pilanci, and M. W. Mahoney. "Newton-LESS: sparsification without trade-offs for the sketched Newton update". In: *Advances in Neural Information Processing Systems* 34 (2021).

[DM05] P. Drineas and M. W. Mahoney. "On the Nyström method for approximating a Gram matrix for improved kernel-based learning". In: *Journal of Machine Learning Research* 6 (2005), pp. 2153–2175.

[DM10] P. Drineas and M. Mahoney. *Effective Resistances, Statistical Leverage, and Applications to Linear Equation Solving*. 2010. arXiv: 1005.3097.

[DM16] P. Drineas and M. W. Mahoney. "RandNLA: randomized numerical linear algebra". In: *Communications of the ACM* 59 (2016), pp. 80–90.

[DM18] P. Drineas and M. W. Mahoney. "Lectures on randomized numerical linear algebra". In: *The Mathematics of Data*. Ed. by M. W. Mahoney, J. C. Duchi, and A. C. Gilbert. IAS/Park City Mathematics Series. Available at https://arxiv.org/abs/1712.08880. AMS/IAS/SIAM, 2018, pp. 1–48.

[DM21a] M. Dereziński and M. W. Mahoney. "Determinantal point processes in randomized numerical linear algebra". In: *Notices of the AMS* 68.1 (2021), pp. 34–45.

[DM21b] Y. Dong and P.-G. Martinsson. *Simpler is better: A comparative study of randomized algorithms for computing the CUR decomposition*. 2021. arXiv: 2104.05877.

[DMB+79] J. J. Dongarra, C. B. Moler, J. R. Bunch, and G. W. Stewart. *LINPACK users guide*. SIAM, 1979.

[DMM+11] P. Drineas, M. W. Mahoney, S. Muthukrishnan, and T. Sarlós. "Faster least squares approximation". In: *Numerische Mathematik* 117.2 (2011), pp. 219–249.

[DMM+12] P. Drineas, M. Magdon-Ismail, M. W. Mahoney, and D. P. Woodruff. "Fast approximation of matrix coherence and statistical leverage". In: *Journal of Machine Learning Research* 13 (2012), pp. 3475–3506.

[DMM06] P. Drineas, M. W. Mahoney, and S. Muthukrishnan. "Sampling algorithms for ℓ_2 regression and applications". In: *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2006, pp. 1127–1136.

[DMM08] P. Drineas, M. W. Mahoney, and S. Muthukrishnan. "Relative-error CUR matrix decompositions". In: *SIAM Journal on Matrix Analysis and Applications* 30.2 (Jan. 2008). This is a longer journal version of two conference papers from 2006., pp. 844–881.

[Drm22] Z. Drmac. *A LAPACK implementation of the Dynamic Mode Decomposition I*. https://netlib.org/lapack/lawns/. LAPACK Working Note 298. Oct. 2022.

[DRV+06] A. Deshpande, L. Rademacher, S. S. Vempala, and G. Wang. "Matrix approximation and projective clustering via volume sampling". In: *Theory of Computing* 2.1 (2006), pp. 225–247.

[DSS+18] H. Diao, Z. Song, W. Sun, and D. Woodruff. "Sketching for Kronecker product regression and P-splines". In: *Proceedings of the 21st International Conference on Artificial Intelligence and Statistics*. 2018, pp. 1299–1308.

[DV06] A. Deshpande and S. Vempala. "Adaptive sampling and fast low-rank matrix approximation". In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Ed. by J. Díaz, K. Jansen, J. D. P. Rolim, and U. Zwick. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 292–303. ISBN: 978-3-540-38045-0.

[EBK19] N. B. Erichson, S. L. Brunton, and J. N. Kutz. "Compressed dynamic mode decomposition for background modeling". In: *Journal of Real-Time Image Processing* 16.5 (2019), pp. 1479–1492.

[ED16] N. B. Erichson and C. Donovan. "Randomized low-rank dynamic mode decomposition for motion detection". In: *Computer Vision and Image Understanding* 146 (2016), pp. 40–50.

[EMB+20] N. B. Erichson, K. Manohar, S. L. Brunton, and J. N. Kutz. "Randomized CP tensor decomposition". In: *Machine Learning: Science and Technology* 1.2 (2020), p. 025012.

[EMK+19] N. B. Erichson, L. Mathelin, J. N. Kutz, and S. L. Brunton. "Randomized dynamic mode decomposition". In: *SIAM Journal on Applied Dynamical Systems* 18.4 (2019), pp. 1867–1891.

[EMW+18] N. B. Erichson, A. Mendible, S. Wihlborn, and N. J. Kutz. "Randomized nonnegative matrix factorization". In: *Pattern Recognition Letters* 104 (2018), pp. 1–7.

[Epp23] E. Epperly. *Stochastic Trace Estimation*. https://www.ethanepperly.com/index.php/2023/01/26/stochastic-trace-estimation/. Accessed: 2023-03-27. Jan. 2023.

[ET94] B. Efron and R. J. Tibshirani. *An introduction to the bootstrap*. CRC press, 1994.

[ETW23] E. N. Epperly, J. A. Tropp, and R. J. Webber. *XTrace: Making the most of every sample in stochastic trace estimation*. 2023. arXiv: 2301.07825.

[EVB+19] N. B. Erichson, S. Voronin, S. L. Brunton, and J. N. Kutz. "Randomized matrix decompositions using R". In: *Journal of Statistical Software* 89.11 (2019).

[EZM+20] N. B. Erichson, P. Zheng, K. Manohar, S. L. Brunton, J. N. Kutz, and A. Y. Aravkin. "Sparse principal component analysis via variable projection". In: *SIAM Journal on Applied Mathematics* 80.2 (2020), pp. 977–1002.

[FFG22] M. Fahrbach, T. Fu, and M. Ghadiri. *Subquadratic Kronecker Regression with Applications to Tensor Decomposition*. 2022. arXiv: 2209.04876.

[FGL21] Y. Fan, Y. Guo, and T. Lin. *A Novel Randomized XR-Based Preconditioned CholeskyQR Algorithm*. 2021. arXiv: 2111.11148.

[FHH99] R. D. Fierro, P. C. Hansen, and P. S. K. Hansen. "UTV tools: Matlab templates for rank-revealing UTV decompositions". In: *Numerical Algorithms* 20.2 (1999), pp. 165–194.

[FKV04] A. Frieze, R. Kannan, and S. Vempala. "Fast Monte-Carlo algorithms for finding low-rank approximations". In: *Journal of the ACM* 51.6 (2004), pp. 1025–1041.

[FS11] D. C.-L. Fong and M. Saunders. "LSMR: an iterative algorithm for sparse least-squares problems". In: 33.5 (Jan. 2011), pp. 2950–2971.

[FTU21] Z. Frangella, J. A. Tropp, and M. Udell. *Randomized Nyström Preconditioning*. 2021. arXiv: 2110.02820 [math.NA].

[FXG18] Y. Feng, J. Xiao, and M. Gu. "Randomized complete pivoting for solving symmetric indefinite linear systems". In: *SIAM Journal on Matrix Analysis and Applications* 39.4 (Jan. 2018), pp. 1616–1641.

[FXG19] Y. Feng, J. Xiao, and M. Gu. "Flip-flop spectrum-revealing QR factorization and its applications to singular value decomposition". In: *ETNA - Electronic Transactions on Numerical Analysis* 51 (2019), pp. 469–494.

[GCG+19] C. Gorman, G. Chávez, P. Ghysels, T. Mary, F.-H. Rouet, and X. S. Li. "Robust and accurate stopping criteria for adaptive randomized sampling in matrix-free hierarchically semiseparable construction". In: *SIAM Journal on Scientific Computing* 41.5 (2019), S61–S85.

[GDX11] L. Grigori, J. Demmel, and H. Xiang. "CALU: a communication optimal LU factorization algorithm". In: *SIAM Journal on Matrix Analysis and Applications* 32 (2011), pp. 1317–1350.

[GE95] M. Gu and S. C. Eisenstat. "A divide-and-conquer algorithm for the bidiagonal SVD". In: *SIAM Journal on Matrix Analysis and Applications* 16.1 (Jan. 1995), pp. 79–92.

[GE96] M. Gu and S. C. Eisenstat. "Efficient algorithms for computing a strong rank-revealing QR factorization". In: *SIAM Journal on Scientific Computing* 17.4 (July 1996), pp. 848–869.

[Gem80] S. Geman. "A limit theorem for the norm of random matrices". In: *The Annals of Probability* 8.2 (1980), pp. 252–261.

[GIG21] N. Gazagnadou, M. Ibrahim, and R. M. Gower. *`RidgeSketch`: A Fast sketching based solver for large scale ridge regression*. 2021. arXiv: 2105.05565 [math.OC].

[Gir87] D. Girard. *Un algorithme simple et rapid pour la validation croisee géenéralisée sur des probléms de grande taille*. 1987.

[Gir89] A. Girard. "A fast 'Monte-Carlo cross-validation' procedure for large least squares problems with noisy data". In: *Numerische Mathematik* 56.1 (1989), pp. 1–23.

[GLA+17] M. Gates, P. Luszczek, A. Abdelfattah, J. Kurzak, J. Dongarra, K. Arturov, C. Cecka, and C. Freitag. *C++ API for BLAS and LAPACK*. Tech. rep. 02, ICL-UT-17-03. Revision 02-21-2018. June 2017.

[GM10] G. H. Golub and G. Meurant. *Matrices, Moments and Quadrature with Applications*. Princeton University Press, 2010. ISBN: 9780691143415.

[GM16] A. Gittens and M. W. Mahoney. "Revisiting the Nyström method for improved large-scale machine learning". In: *Journal of Machine Learning Research* 17.117 (2016), pp. 1–65.

[GM18] A. Gopal and P.-G. Martinsson. *The PowerURV algorithm for computing rank-revealing full factorizations*. 2018. arXiv: 1812.06007.

[GR15] R. M. Gower and P. Richtárik. "Randomized iterative methods for linear systems". In: *SIAM Journal on Matrix Analysis and Applications* 36.4 (2015), pp. 1660–1690.

[Gri16] O. Grisel. *SciKit-Learn PR #5299: [MRG+3] Collapsing PCA and RandomizedPCA*. https://github.com/scikit-learn/scikit-learn/pull/5299. Released in SciPy 0.18.0. Website accessed: 2023-04-03. 2016.

[GS12] V. Guruswami and A. K. Sinop. "Optimal column-based low-rank matrix reconstruction". In: *Proceedings of the twenty-third annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM. 2012, pp. 1207–1214.

[GS22] S. Güttel and M. Schweitzer. *Randomized sketching for Krylov approximations of large-scale matrix functions*. 2022. arXiv: 2208.11447.

[GSO17] A. S. Gambhir, A. Stathopoulos, and K. 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.

[GTZ97] S. Goreinov, E. Tyrtyshnikov, and N. Zamarashkin. "A theory of pseudoskeleton approximations". In: *Linear Algebra and its Applications* 261.1-3 (Aug. 1997), pp. 1–21.

[GV13] G. H. Golub and C. F. Van Loan. *Matrix Computations*. en. 4th ed. Johns Hopkins Studies in the Mathematical Sciences. Baltimore, MD: Johns Hopkins University Press, Feb. 2013.

[GV61] G. H. Golub and R. S. Varga. "Chebyshev semi-iterative methods, successive overrelaxation iterative methods, and second order Richardson iterative methods". In: *Numerische Mathematik* 3.1 (Dec. 1961), pp. 157–168.

[GZT95] S. A. Goreĭnov, N. L. Zamarashkin, and E. E. Tyrtyshnikov. "Pseudo-skeleton approximations of matrices". In: *Dokl. Akad. Nauk* 343.2 (1995), pp. 151–152.

[Hig02] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. Second. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2002, pp. xxx+680. ISBN: 0-89871-521-0.

[Hig08] N. J. Higham. *Functions of Matrices*. Society for Industrial and Applied Mathematics, Jan. 2008.

[Hig97] N. J. Higham. "Iterative refinement for linear systems and LAPACK". In: *IMA Journal of Numerical Analysis* 17.4 (1997), pp. 495–509.

[HL69] R. J. Hanson and C. L. Lawson. "Extensions and applications of the householder algorithm for solving linear least squares problems". In: *Mathematics of Computation* 23.108 (1969), pp. 787–812.

[HMT11] N. Halko, P. G. Martinsson, and J. A. Tropp. "Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions". In: *SIAM Review* 53.2 (Jan. 2011), pp. 217–288.

[HS52] M. R. Hestenes and E. Stiefel. "Methods of conjugate gradients for solving linear systems". In: *Journal of Research of the National Bureau of Standards* 49.1 (1952).

[Hut90] M. Hutchinson. "A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines". In: *Communications in Statistics - Simulation and Computation* 19.2 (Jan. 1990), pp. 433–450.

[IEE19] IEEE. "IEEE Standard for Floating-Point Arithmetic". In: *IEEE Std 754-2019 (Revision of IEEE 754-2008)* (2019), pp. 1–84.

[IM98] P. Indyk and R. Motwani. "Approximate nearest neighbors: towards removing the curse of dimensionality". In: *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC)*. 1998, pp. 604–613.

[INR+20] M. A. Iwen, D. Needell, E. Rebrova, and A. Zare. *Lower Memory Oblivious (Tensor) Subspace Embeddings with Fewer Random Bits: Modewise Methods for Least Squares*. 2020. arXiv: 1912.08294.

[Int19] Intel. *Notes for oneMKL Vector Statistics*. Tech. rep. Intel Corporation, 2019, p. 120.

[Ips09] I. C. F. Ipsen. *Numerical Matrix Analysis: Linear Systems and Least Squares*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2009, pp. xiv+128. ISBN: 978-0-898716-76-4.

[JKW20] R. Jin, T. G. Kolda, and R. Ward. "Faster Johnson–Lindenstrauss transforms via Kronecker products". In: *Information and Inference: A Journal of the IMA* 10.4 (Oct. 2020), pp. 1533–1562.

[JL84] W. Johnson and J. Lindenstrauss. "Extensions of Lipshitz mapping into Hilbert space". In: *Contemporary Mathematics* 26 (1984), pp. 189–206.

[JZ13] R. Johnson and T. Zhang. "Accelerating stochastic gradient descent using predictive variance reduction". In: *Advances in Neural Information Processing Systems* 26 (2013).

[KAI+15] G. Kollias, H. Avron, Y. Ineichen, C. Bekas, A. Curioni, V. Sindhwani, and K. Clarkson. *libSkylark: A Framework for High-Performance Matrix Sketching for Statistical Computing*. http://sc15.supercomputing.org/sites/all/themes/SC15images/tech_poster/poster_files/post213s2-file3.pdf. 2015.

[KB09] T. G. Kolda and B. W. Bader. "Tensor decompositions and applications". In: *SIAM Review* 51.3 (Aug. 2009), pp. 455–500.

[KC21] M. F. Kaloorazi and J. Chen. "Projection-based QLP algorithm for efficiently computing low-rank approximation of matrices". In: *IEEE Transactions on Signal Processing* 69 (2021), pp. 2218–2232.

[KCL21] M. F. Kaloorazi, J. Chen, and R. C. de Lamare. *A QLP Decomposition via Randomization*. 2021. arXiv: 2110.01011.

[KMT09a] S. Kumar, M. Mohri, and A. Talwalkar. "Ensemble Nyström method". In: *Annual Advances in Neural Information Processing Systems*. 2009.

[KMT09b] S. Kumar, M. Mohri, and A. Talwalkar. "Sampling techniques for the Nyström method". In: *Proceedings of the 12th Tenth International Workshop on Artificial Intelligence and Statistics*. 2009, pp. 304–311.

[KN12] D. M. Kane and J. Nelson. "Sparser Johnson-Lindenstrauss Transforms". In: *Proceedings of the 2012 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2012, pp. 1195–1206.

[KN14] D. M. Kane and J. Nelson. "Sparser Johnson-Lindenstrauss Transforms". In: *J. ACM* 61.1 (Jan. 2014). Notes: journal version of a 2012 SODA paper by the same name; called "OSNAPs" in a related 2013 paper.

[KRS+10] S. P. Kasiviswanathan, M. Rudelson, A. Smith, and J. Ullman. "The price of privately releasing contingency tables and the spectra of random matrices with correlated rows". In: *Proceedings of the Forty-Second ACM Symposium on Theory of Computing*. 2010, pp. 775–784.

[KT12] A. Kulesza and B. Taskar. "Determinantal point processes for machine learning". In: *Foundations and Trends® in Machine Learning* 5.2–3 (2012), pp. 123–286.

[KV17a] R. Kannan and S. Vempala. "Randomized algorithms in numerical linear algebra". In: *Acta Numerica* 26 (2017), pp. 95–135.

[KV17b] W. Kong and G. Valiant. "Spectrum estimation from samples". In: *The Annals of Statistics* 45.5 (2017), pp. 2218–2247.

[KW70] G. S. Kimeldorf and G. Wahba. "A correspondence between Bayesian estimation on stochastic processes and smoothing by splines". In: *The Annals of Mathematical Statistics* 41.2 (1970), pp. 495–502.

[KW92] J. Kuczyński and H. 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.

[KWG+17] J. Kurzak, P. Wu, M. Gates, I. Yamazaki, P. Luszczek, G. Ragghianti, and J. Dongarra. *Designing SLATE: Software for Linear Algebra Targeting Exascale*. SLATE Working Notes 03, ICL-UT-17-06. Oct. 2017.

[LEM20] M. E. Lopes, N. B. Erichson, and M. Mahoney. "Error estimation for sketched SVD via the bootstrap". In: *Proceedings of the 37thInternational Conference on Machine Learning (ICML)*. Ed. by H. D. III and A. Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, July 2020, pp. 6382–6392.

[LEM23] M. E. Lopes, N. B. Erichson, and M. W. Mahoney. "Bootstrapping the operator norm in high dimensions: Error estimation for covariance matrices and sketching". In: *Bernoulli* 29.1 (2023), pp. 428–450.

[LHK+79] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. "Basic linear algebra subprograms for fortran usage". In: *ACM Trans. Math. Softw.* 5.3 (Sept. 1979), pp. 308–323.

[Li92] K.-h. Li. "Generation of random matrices with orthonormal columns and multivariate normal variates with given sample mean and covariance". In: *Journal of Statistical Computation and Simulation* 43.1-2 (Oct. 1992), pp. 11–18.

[Lib09] E. Liberty. "Accelerated dense random projections". PhD thesis. Yale University, May 2009.

[Lin16] L. Lin. "Randomized estimation of spectral densities of large matrices made accurate". In: *Numerische Mathematik* 136.1 (Aug. 2016), pp. 183–213.

[LK20] B. W. Larsen and T. G. Kolda. *Practical Leverage-Based Sampling for Low-Rank Tensor Decomposition*. v3 released in 2022. 2020. arXiv: 2006.16438.

[LKL10] M. Li, J. Kwok, and B.-L. Lu. "Making large-scale Nyström approximation possible". In: *Proceedings of the 27th International Conference on Machine Learning (ICML)*. 2010, pp. 631–638.

[LLD20] N. Lindquist, P. Luszczek, and J. Dongarra. "Replacing pivoting in distributed Gaussian elimination with randomized techniques". In: *2020 IEEE/ACM 11th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA)*. 2020, pp. 35–43.

[LLS+17] H. Li, G. C. Linderman, A. Szlam, K. P. Stanton, Y. Kluger, and M. Tygert. "Algorithm 971: an implementation of a randomized algorithm for principal component analysis". In: *ACM Trans. Math. Softw.* 43.3 (Jan. 2017).

[LP19] J. Lacotte and M. Pilanci. *Faster least squares optimization*. 2019.

[LSS13] Q. Le, T. Sarlós, and A. Smola. "Fastfood-computing Hilbert space expansions in loglinear time". In: *International Conference on Machine Learning*. PMLR. 2013, pp. 244–252.

[LWM+07] E. Liberty, F. Woolfe, P. G. Martinsson, V. Rokhlin, and M. Tygert. "Randomized algorithms for the low-rank approximation of matrices". In: *Proceedings of the National Academy of Sciences* 104.51 (2007), pp. 20167–20172.

[LWM18] M. E. Lopes, S. Wang, and M. Mahoney. "Error estimation for randomized least-squares algorithms via the bootstrap". In: *Proceedings of the 35th International Conference on Machine Learning (ICML)*. Ed. by J. Dy and A. Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, July 2018, pp. 3217–3226.

[Mah11] M. W. Mahoney. *Randomized algorithms for matrices and data*. Foundations and Trends in Machine Learning. Boston: NOW Publishers, 2011.

[Mah16] M. W. Mahoney. *Lecture Notes on Randomized Linear Algebra*. 2016.

[Mal22] O. A. Malik. "More efficient sampling for tensor decomposition with worst-case guarantees". In: *Proceedings of the 39th International Conference on Machine Learning*. Vol. 162. Proceedings of Machine Learning Research. PMLR, 2022, pp. 14887–14917.

[Mar15] P. G. Martinsson. *Blocked rank-revealing QR factorizations: How randomized sampling can be used to avoid single-vector pivoting*. 2015. arXiv: 1505.08115.

[Mar18] P.-G. Martinsson. "Randomized methods for matrix computations". In: *The Mathematics of Data* 25.4 (2018). Note: preprint arXiv:1607.01649 published in 2016, updated in 2019., pp. 187–239.

[Mar22a] P. G. Martinsson. A remark on the precision of random number generation for RandNLA. Personal communication. 2022.

[Mar22b] P. G. Martinsson. A remark on pivoting methods in randomized algorithms for low-rank interpolative decomposition. Personal communication. 2022.

[MB18] O. A. Malik and S. Becker. "Low-rank Tucker decomposition of large tensors using TensorSketch". In: *Advances in Neural Information Processing Systems*. Vol. 31. Curran Associates, Inc., 2018.

[MB20] O. A. Malik and S. Becker. "Guarantees for the Kronecker fast Johnson–Lindenstrauss transform using a coherence and sampling argument". In: *Linear Algebra and its Applications* 602 (Oct. 2020), pp. 120–137.

[MB21] O. A. Malik and S. Becker. "A sampling-based method for tensor ring decomposition". In: *International Conference on Machine Learning*. PMLR, 2021, pp. 7400–7411.

[MBM22] O. A. Malik, V. Bharadwaj, and R. Murray. *Sampling-Based Decomposition Algorithms for Arbitrary Tensor Networks*. 2022. arXiv: 2210.03828.

[MCD+22] G. Meanti, L. Carratino, E. De Vito, and L. Rosasco. "Efficient hyperparameter tuning for large scale kernel ridge regression". In: *(to appear in) Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*. 2022.

[MCR+20] G. Meanti, L. Carratino, L. Rosasco, and A. Rudi. "Kernel methods through the roof: handling billions of points efficiently". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 14410–14422.

[MD09] M. W. Mahoney and P. Drineas. "CUR matrix decompositions for improved data analysis". In: *Proceedings of the National Academy of Sciences* 106.3 (2009), pp. 697–702.

[MD16] M. W. Mahoney and P. Drineas. "Structural properties underlying high-quality randomized numerical linear algebra algorithms". In: *Handbook of Big Data*. Ed. by P. Bühlmann, P. Drineas, M. Kane, and M. van de Laan. CRC Press, 2016, pp. 137–154.

[Mez07] F. Mezzadri. "How to generate random matrices from the classical compact groups". In: *Notices of the AMS* 54.5 (2007), pp. 592–604.

[MG15] C. Melgaard and M. Gu. *Gaussian Elimination with Randomized Complete Pivoting*. 2015.

[MHG17] P.-G. Martinsson, G. Q. O. N. Heavner, and R. van de Geijn. "Householder QR factorization with randomization for column pivoting (HQRRP)". In: *SIAM Journal on Scientific Computing* 39.2 (Jan. 2017), pp. C96–C115.

[MM13] X. Meng and M. W. Mahoney. "Low-distortion subspace embeddings in input-sparsity time and applications to robust linear regression". In: *Proceedings of the 45th Annual ACM Symposium on Theory of Computing*. 2013, pp. 91–100.

[MM15] C. Musco and C. Musco. "Randomized block Krylov methods for stronger and faster approximate singular value decomposition". In: *Neural Information Processing Systems*. 2015, pp. 1396–1404.

[MM17] C. Musco and C. Musco. "Recursive sampling for the Nyström method". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc., 2017.

[MMM+21] R. A. Meyer, C. Musco, C. Musco, and D. P. Woodruff. "Hutch++: optimal stochastic trace estimation". In: *Symposium on Simplicity in Algorithms (SOSA)*. Society for Industrial and Applied Mathematics, Jan. 2021, pp. 142–155.

[MMY15] P. Ma, M. W. Mahoney, and B. Yu. "A statistical perspective on algorithmic leveraging". In: *Journal of Machine Learning Research* 16 (2015), pp. 861–911.

[Moo65] G. E. Moore. "Cramming more components onto integrated circuits". In: *Electronics* 38.8 (1965), pp. 114–117.

[MQH19] P. G. Martinsson, G. Quintana-Ortí, and N. Heavner. "RandUTV: A blocked randomized algorithm for computing a rank-revealing UTV factorization". In: *ACM Trans. Math. Softw.* 45.1 (Mar. 2019).

[MRS+14] P.-G. Martinsson, V. Rokhlin, Y. Shkolinsky, and M. Tygert. *ID: A software package for low-rank approximation of matrices via interpolative decompositions, Version 0.4*. http://www.tygert.com/id_doc.4.pdf. Available in SciPy. See also https://github.com/klho/PyMatrixID. Mar. 2014.

[MS22] L. Ma and E. Solomonik. *Cost-efficient Gaussian Tensor Network Embeddings for Tensor-structured Inputs*. 2022. arXiv: 2205.13163.

[MSM14] X. Meng, M. A. Saunders, and M. W. Mahoney. "LSRN: a parallel iterative solver for strongly over- or underdetermined systems". In: *SIAM Journal on Scientific Computing* 36.2 (Jan. 2014). Software at https://web.stanford.edu/group/SOL/software/lsrn/, pp. C95–C118.

[MT00] G. Marsaglia and W. W. Tsang. "The ziggurat method for generating random variables". In: *Journal of Statistical Software* 5.8 (2000), pp. 1–7.

[MT20] P.-G. Martinsson and J. A. Tropp. "Randomized numerical linear algebra: Foundations and Algorithms". In: *Acta Numerica* 29 (2020), pp. 403–572.

[Mur12] K. P. Murphy. *Machine Learning: A Probabilistic Perspective (Adaptive Computation and Machine Learning series)*. English. Hardcover. The MIT Press, Aug. 24, 2012, p. 1104.

[MV16] P.-G. Martinsson and S. Voronin. "A randomized blocked algorithm for efficiently computing rank-revealing factorizations of matrices". In: *SIAM Journal on Scientific Computing* 38.5 (Jan. 2016), S485–S507.

[MXC+22] O. A. Malik, Y. Xu, N. Cheng, S. Becker, A. Doostan, and A. Narayan. *Fast Algorithms for Monotone Lower Subsets of Kronecker Least Squares Problems*. 2022. arXiv: 2209.05662.

[MZX+22] P. Ma, X. Zhang, X. Xing, J. Ma, and M. W. Mahoney. "Asymptotic analysis of sampling estimators for randomized numerical linear algebra algorithms". In: *Journal of Machine Learning Research* 23.177 (2022). Journal version of a 2020 PMLR paper of the same name., pp. 1–45.

[Nak20] Y. Nakatsukasa. *Fast and stable randomized low-rank matrix approximation*. 2020. arXiv: 2009.11392.

[NDM22] S. Na, M. Dereziński, and M. W. Mahoney. *Hessian Averaging in Stochastic Newton Methods Achieves Superlinear Convergence*. 2022. arXiv: 2204.09266.

[NDT09] N. H. Nguyen, T. T. Do, and T. D. Tran. "A fast and efficient algorithm for low-rank approximation of a matrix". In: *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing (STOC)*. STOC '09. Bethesda, MD, USA: Association for Computing Machinery, 2009, pp. 215–224. ISBN: 9781605585062.

[Ngu07] H. Nguyen, ed. *GPU Gems 3*. First. Addison-Wesley Professional, 2007. ISBN: 9780321545428.

[NN13] J. Nelson and H. L. Nguyen. "OSNAP: faster numerical linear algebra algorithms via sparser subspace embeddings". In: *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*. IEEE, Oct. 2013.

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

[NT21] Y. Nakatsukasa and J. A. Tropp. *Fast & Accurate Randomized Algorithms for Linear Systems and Eigenvalue Problems*. 2021. arXiv: 2111.00113 [math.NA].

[NTD10] R. Nath, S. Tomov, and J. Dongarra. "Accelerating GPU kernels for dense linear algebra". In: *Proceedings of the 2009 International Meeting on High Performance Computing for Computational Science, VECPAR'10*. Berkeley, CA: Springer, June 2010.

[OA17] D. Orban and M. Arioli. *Iterative Solution of Symmetric Quasi-Definite Linear Systems*. Society for Industrial and Applied Mathematics, Apr. 2017.

[OP64] W. Oettli and W. Prager. "Compatibility of approximate solution of linear equations with given error bounds for coefficients and right-hand sides". In: *Numerische Mathematik* 6.1 (Dec. 1964), pp. 405–409.

[OPA19] I. K. Ozaslan, M. Pilanci, and O. Arikan. "Iterative Hessian sketch with momentum". In: *2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2019, pp. 7470–7474.

[OT17] S. Oymak and J. A. Tropp. "Universality laws for randomized dimension reduction, with applications". In: *Information and Inference: A Journal of the IMA* 7.3 (2017), pp. 337–446.

[Pag13] R. Pagh. "Compressed matrix multiplication". In: *ACM Transactions on Computation Theory* 5.3 (Aug. 2013), 9:1–9:17.

[Pan00] C.-T. Pan. "On the existence and computation of rank-revealing lu factorizations". In: *Linear Algebra and its Applications* 316.1 (2000). Special Issue: Conference celebrating the 60th birthday of Robert J. Plemmons, pp. 199–222.

[Par95] D. S. Parker. *Random Butterfly Transformations with Applications in Computational Linear Algebra*. Tech. rep. University of California, Los Angeles, 1995.

[PCK21] D. Persson, A. Cortinovis, and D. Kressner. *Improved variants of the Hutch++ algorithm for trace estimation*. 2021. arXiv: 2109.10659.

[Pec21] J. Peca-Medlin. "Numerical, spectral, and group properties of random butterfly matrices". PhD thesis. University of California, Irvine, 2021.

[PJM22] V. Patel, M. Jahangoshahi, and D. A. Maldonado. *Randomized Block Adaptive Linear System Solvers*. 2022. arXiv: 2204.01653.

[PK22] D. Persson and D. Kressner. *Randomized low-rank approximation of monotone matrix functions*. 2022. arXiv: 2209.11023.

[Pla05] J. Platt. "FastMap, MetricMap, and Landmark MDS are all Nyström algorithms". In: *Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics*. 2005, pp. 261–268.

[PMG+13] J. Poulson, B. Marker, R. A. van de Geijn, J. R. Hammond, and N. A. Romero. "Elemental". In: 39.2 (Feb. 2013). https://github.com/elemental/Elemental, pp. 1–24.

[Pou20] J. Poulson. "High-performance sampling of generic determinantal point processes". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 378.2166 (Jan. 2020), p. 20190059.

[PP13] N. Pham and R. Pagh. "Fast and scalable polynomial kernels via explicit feature maps". In: *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '13. New York, NY, USA: ACM, 2013, pp. 239–247. ISBN: 978-1-4503-2174-7.

[PS82] C. C. Paige and M. A. Saunders. "LSQR: an algorithm for sparse linear equations and sparse least squares". In: *ACM Trans. Math. Softw.* 8.1 (Mar. 1982), pp. 43–71.

[PW16] M. Pilanci and M. J. Wainwright. "Iterative Hessian Sketch: fast and accurate solution approximation for constrained least-squares". In: *J. Mach. Learn. Res.* 17.1 (Jan. 2016), pp. 1842–1879.

[PW17] M. Pilanci and M. J. Wainwright. "Newton Sketch: a near linear-time optimization algorithm with linear-quadratic convergence". In: *SIAM Journal on Optimization* 27.1 (Jan. 2017), pp. 205–245.

[RB20] H. Ren and Z.-J. Bai. *Single-pass randomized QLP decomposition for low-rank approximation*. 2020. arXiv: 2011.06855.

[RCC+18] A. Rudi, D. Calandriello, L. Carratino, and L. Rosasco. "On fast leverage score sampling and optimal learning". In: *Advances in Neural Information Processing Systems* 31 (2018).

[RCR15] A. Rudi, R. Camoriano, and L. Rosasco. *Less is More: Nyström Computational Regularization*. 2015. arXiv: 1507.04717.

[RDA18] J. Riedy, J. Demmel, and P. Ahrens. "Reproducible BLAS: Make Addition Associative Again!" In: *SIAM News* (Oct. 2018).

[RM19] F. Roosta-Khorasani and M. W. Mahoney. "Sub-sampled Newton methods". In: *Mathematical Programming* 174.1-2 (2019), pp. 293–326.

[RR07] A. Rahimi and B. Recht. "Random features for large-scale kernel machines". In: *Advances in Neural Information Processing Systems*. Ed. by J. Platt, D. Koller, Y. Singer, and S. Roweis. Vol. 20. Curran Associates, Inc., 2007.

[RR20] B. Rakhshan and G. Rabusseau. "Tensorized random projections". In: *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*. Ed. by S. Chiappa and R. Calandra. Vol. 108. Proceedings of Machine Learning Research. PMLR, Aug. 2020, pp. 3306–3316.

[RR21] B. T. Rakhshan and G. Rabusseau. "Rademacher random projections with tensor networks". In: *NeurIPS Workshop on Quantum Tensor Networks in Machine Learning*. 2021.

[RST10] V. Rokhlin, A. Szlam, and M. Tygert. "A randomized algorithm for principal component analysis". In: *SIAM Journal on Matrix Analysis and Applications* 31.3 (Jan. 2010), pp. 1100–1124.

[RT08] V. Rokhlin and M. 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.

[Rud12] M. Rudelson. "Row products of random matrices". In: *Advances in Mathematics* 231.6 (2012), pp. 3199–3231.

[SAI17] A. K. Saibaba, A. Alexanderian, and I. C. F. Ipsen. "Randomized matrix-free trace and log-determinant estimators". In: *Numerische Mathematik* 137.2 (Apr. 2017), pp. 353–395.

[Sar06] T. Sarlos. "Improved approximation algorithms for large matrices via random projections". In: *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. FOCS '06. USA: IEEE Computer Society, 2006, pp. 143–152. ISBN: 0769527205.

[SCS10] Y. Saad, J. Chelikowsky, and S. Shontz. "Numerical methods for electronic structure calculations of materials". In: *SIAM Review* 52.1 (2010), pp. 3–54.

[SDF+17] N. D. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E. E. Papalexakis, and C. Faloutsos. "Tensor decomposition for signal processing and machine learning". In: *IEEE Transactions on Signal Processing* 65.13 (2017), pp. 3551–3582.

[SG21] A. Sobczyk and E. Gallopoulos. "Estimating leverage scores via rank revealing methods and randomization". In: *SIAM Journal on Matrix Analysis and Applications* 42.3 (2021), pp. 1199–1228.

[SG22] A. Sobczyk and E. Gallopoulos. *pylspack: Parallel algorithms and data structures for sketching, column subset selection, regression and leverage scores*. 2022. arXiv: 2203.02798.

[SGT+18] Y. Sun, Y. Guo, J. A. Tropp, and M. Udell. "Tensor random projection for low memory dimension reduction". In: *NeurIPS Workshop on Relational Representation Learning*. 2018.

[Sil85] J. W. Silverstein. "The smallest eigenvalue of a large dimensional Wishart matrix". In: *The Annals of Probability* 13.4 (1985), pp. 1364–1368.

[SMD+11] J. K. Salmon, M. A. Moraes, R. O. Dror, and D. E. Shaw. "Parallel random numbers: as easy as 1, 2, 3". In: *SC '11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. 2011, pp. 1–12.

[SNM17] P. Seshadri, A. Narayan, and S. Mahadevan. "Effectively subsampled quadratures for least squares polynomial approximations". In: *SIAM/ASA Journal on Uncertainty Quantification* 5.1 (2017), pp. 1003–1023.

[SSA+18] G. Shabat, Y. Shmueli, Y. Aizenbud, and A. Averbuch. "Randomized LU decomposition". In: *Applied and Computational Harmonic Analysis* 44.2 (Mar. 2018). Available on arXiv in 2013., pp. 246–272.

[ST02] Z. Strakoš and P. Tichý. "On error estimation in the conjugate gradient method and why it works in finite precision computations". In: *Electron. Trans. Numer. Anal.* 13 (2002), pp. 56–80.

[ST05] Z. Strakoš and P. Tichý. "Error estimation in preconditioned conjugate gradients". In: *BIT Numerical Mathematics* 45.4 (Dec. 2005). Extends a related 2002 paper by the same authors., pp. 789–817.

[ST12] J. Shao and D. Tu. *The jackknife and bootstrap*. Springer Science & Business Media, 2012.

[Ste77] G. Stewart. "Research, development, and LINPACK". In: *Mathematical Software*. Elsevier, 1977, pp. 1–14.

[Ste80] G. W. Stewart. "The efficient generation of random orthogonal matrices with an application to condition estimators". In: *SIAM Journal on Numerical Analysis* 17.3 (1980), pp. 403–409.

[Ste92] G. Stewart. "An updating algorithm for subspace tracking". In: *IEEE Transactions on Signal Processing* 40.6 (June 1992), pp. 1535–1541.

[Ste93] G. Stewart. "Updating a rank-revealing ULV decomposition". In: *SIAM Journal on Matrix Analysis and Applications* 14.2 (Apr. 1993), pp. 494–499.

[Ste99] G. W. Stewart. "The QLP approximation to the singular value decomposition". In: *SIAM J. Sci. Comput.* 20.4 (Jan. 1999), pp. 1336–1348.

[SV08] T. Strohmer and R. Vershynin. "A randomized Kaczmarz algorithm with exponential convergence". In: *Journal of Fourier Analysis and Applications* 15.2 (Apr. 2008), pp. 262–278.

[SWY+21] Z. Song, D. Woodruff, Z. Yu, and L. Zhang. "Fast sketching of polynomial kernels of polynomial degree". In: *Proceedings of the 38th International Conference on Machine Learning (ICML)*. Vol. 139. Proceedings of Machine Learning Research. PMLR, 2021, pp. 9812–9823.

[TDB10] S. Tomov, J. Dongarra, and M. Baboulin. "Towards dense linear algebra for hybrid GPU accelerated manycore systems". In: *Parallel Computing* 36.5-6 (June 2010), pp. 232–240.

[TNX15] Tao, A. Narayan, and D. Xiu. "Weighted discrete least-squares polynomial approximation using randomized quadratures". In: *Journal of Computational Physics* 298 (2015), pp. 787–800.

[TRL+14] J. H. Tu, C. W. Rowley, D. M. Luchtenburg, S. L. Brunton, and J. N. Kutz. "On dynamic mode decomposition: theory and applications". In: *Journal of Computational Dynamics* 1.2 (2014), pp. 391–421.

[Tro11] J. A. Tropp. "Improved analysis of the subsampled randomized Hadamard transform". In: *Advances in Adaptive Data Analysis* 03.01n02 (Apr. 2011), pp. 115–126.

[Tro15] J. A. Tropp. "An introduction to matrix concentration inequalities". In: *Foundations and Trends® in Machine Learning* 8.1-2 (2015), pp. 1–230.

[Tro19] J. A. Tropp. *Matrix Concentration & Computational Linear Algebra*. Lecture notes for a course at École Normale Supérieure, Paris. July 2019.

[Tro20] J. A. Tropp. *Randomized Algorithms for Matrix Computations*. Lecture notes (available online in April 2021). Mar. 2020.

[Tyg22] M. Tygert. A suggestion for sparse sketching operators. Personal communication. 2022.

[TYU+17a] J. A. Tropp, A. Yurtsever, M. Udell, and V. Cevher. "Fixed-rank approximation of a positive-semidefinite matrix from streaming data". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc., 2017.

[TYU+17b] J. A. Tropp, A. Yurtsever, M. Udell, and V. Cevher. "Practical sketching algorithms for low-rank matrix approximation". In: *SIAM Journal on Matrix Analysis and Applications* 38.4 (Jan. 2017), pp. 1454–1485.

[UCS17] S. Ubaru, J. Chen, and Y. Saad. "Fast estimation of tr(f(A)) via stochastic Lanczos quadrature". In: *SIAM Journal on Matrix Analysis and Applications* 38.4 (Jan. 2017), pp. 1075–1099.

[Ura13] Y. Urano. "A fast randomized algorithm for linear least-squares regression via sparse transforms". MA thesis. New York University, Jan. 2013.

[VBG+18] J. S. Vetter, R. Brightwell, M. Gokhale, P. McCormick, R. Ross, J. Shalf, K. Antypas, D. Donofrio, T. Humble, C. Schuman, B. Van Essen, S. Yoo, A. Aiken, D. Bernholdt, S. Byna, K. Cameron, F. Cappello, B. Chapman, A. Chien, M. Hall, R. Hartman-Baker, Z. Lan, M. Lang, J. Leidel, S. Li, R. Lucas, J. Mellor-Crummey, P. Peltz Jr., T. Peterka, M. Strout, and J. Wilke. *Extreme Heterogeneity 2018 – Productive Computational Science in the Era of Extreme Heterogeneity: Report for DOE ASCR Workshop on Extreme Heterogeneity*. English. Tech. rep. https://www.osti.gov/servlets/purl/1473756. US DOE Office of Science (SC), Washington, D.C. (United States), Dec. 2018.

[VEK+19] M. Velegar, N. B. Erichson, C. A. Keller, and J. N. Kutz. "Scalable diagnostics for global atmospheric chemistry using ristretto library (version 1.0)". In: *Geoscientific Model Development* 12.4 (2019), pp. 1525–1539.

[Ver18] R. Vershynin. *High-dimensional probability: An introduction with applications in data science*. Cambridge, United Kingdom New York, NY: Cambridge University Press, 2018. ISBN: 9781108231596.

[VM15] S. Voronin and P.-G. Martinsson. *RSVDPACK: An implementation of randomized algorithms for computing the singular value, interpolative, and CUR decompositions of matrices on multi-core and GPU architectures*. 2015.

[VM16] S. Voronin and P.-G. Martinsson. "Efficient algorithms for CUR and interpolative matrix decompositions". In: *Advances in Computational Mathematics* 43.3 (Nov. 2016), pp. 495–516.

[Wan15] S. Wang. *A Practical Guide to Randomized Matrix Computations with MATLAB Implementations*. 2015. arXiv: 1505.07570.

[WGM18] S. Wang, A. Gittens, and M. W. Mahoney. "Sketched ridge regression: optimization perspective, statistical perspective, and model averaging". In: *Journal of Machine Learning Research* 18 (2018), pp. 1–50.

[WLR+08] F. Woolfe, E. Liberty, V. Rokhlin, and M. Tygert. "A fast randomized algorithm for the approximation of matrices". In: *Applied and Computational Harmonic Analysis* 25.3 (2008), pp. 335–366.

[Woo14] D. P. Woodruff. "Sketching as a tool for numerical linear algebra". In: *Found. Trends Theor. Comput. Sci.* 10.1–2 (Oct. 2014), pp. 1–157.

[WS00] C. Williams and M. Seeger. "Using the nyström method to speed up kernel machines". In: *Advances in Neural Information Processing Systems*. Ed. by T. Leen, T. Dietterich, and V. Tresp. Vol. 13. MIT Press, 2000.

[WX20] N. Wu and H. Xiang. "Randomized QLP decomposition". In: *Linear Algebra and its Applications* 599 (Aug. 2020), pp. 18–35.

[WZ20] D. Woodruff and A. Zandieh. "Near input sparsity time kernel embeddings via adaptive sampling". In: *Proceedings of the 37th International Conference on Machine Learning*. Vol. 119. Proceedings of Machine Learning Research. PMLR, 2020, pp. 10324–10333.

[WZ22] D. Woodruff and A. Zandieh. "Leverage score sampling for tensor product matrices in input sparsity time". In: *Proceedings of the 39th International Conference on Machine Learning*. Vol. 162. Proceedings of Machine Learning Research. PMLR, 2022, pp. 23933–23964.

[XG16] J. Xiao and M. Gu. "Spectrum-revealing Cholesky factorization for kernel methods". In: *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, Dec. 2016.

[XGL17] J. Xiao, M. Gu, and J. Langou. "Fast parallel randomized QR with column pivoting algorithms for reliable low-rank matrix approximations". In: *2017 IEEE 24th International Conference on High Performance Computing (HiPC)*. 2017, pp. 233–242.

[XRM17] P. Xu, F. Roosta-Khorasani, and M. W. Mahoney. *Newton-Type Methods for Non-Convex Optimization Under Inexact Hessian Information*. 2017. arXiv: 1708.07164.

[YCR+18] J. Yang, Y.-L. Chow, C. Re, and M. W. Mahoney. "Weighted SGD for Lp regression with randomized preconditioning". In: *Journal of Machine Learning Research* 18.211 (2018), pp. 1–43.

[YGL+17] W. Yu, Y. Gu, J. Li, S. Liu, and Y. Li. "Single-pass PCA of large high-dimensional data". In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*. 2017, pp. 3350–3356.

[YGL18] W. Yu, Y. Gu, and Y. Li. "Efficient randomized algorithms for the fixed-precision low-rank matrix approximation". In: *SIAM Journal on Matrix Analysis and Applications* 39.3 (Jan. 2018), pp. 1339–1359.

[YMM16] J. Yang, X. Meng, and M. W. Mahoney. "Implementing randomized matrix algorithms in parallel and distributed environments". In: *Proceedings of the IEEE* 104.1 (2016), pp. 58–92.

[YPW17] Y. Yang, M. Pilanci, and M. J. Wainwright. "Randomized sketches for kernels: fast and optimal nonparametric regression". In: *The Annals of Statistics* 45.3 (2017), pp. 991–1023.

[YXR+18] Z. Yao, P. Xu, F. Roosta-Khorasani, and M. W. Mahoney. *Inexact Non-Convex Newton-Type Methods*. 2018. arXiv: 1802.06925.

[ZM20] B. Zhang and M. Mascagni. *Pass-Efficient Randomized LU Algorithms for Computing Low-Rank Matrix Approximation*. 2020. arXiv: 2002.07138.