2019

Randomized Algorithms for Preconditioner Selection with Applications to Kernel Regression

Conner DiPaolo

Randomized Algorithms for Preconditioner Selection, with Applications to Kernel Regression

Conner DiPaolo

Weiqing Gu, Advisor

Nicholas Pippenger, Reader

Department of Mathematics

May, 2019

Abstract

The task of choosing a preconditioner $\boldsymbol{M}$ to use when solving a linear system $\boldsymbol{Ax} = \boldsymbol{b}$ with iterative methods is often tedious and most methods remain ad-hoc. This thesis presents a randomized algorithm to make this chore less painful through use of randomized algorithms for estimating traces. In particular, we show that the preconditioner stability $\|\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A}\|_{\mathsf{F}}$, known to forecast preconditioner quality, can be computed in the time it takes to run a constant number of iterations of conjugate gradients through use of sketching methods. This is in spite of folklore which suggests the quantity is impractical to compute, and a proof we give that ensures the quantity could not possibly be approximated in a useful amount of time by a deterministic algorithm. Using our estimator, we provide a method which can provably select a quality preconditioner among n candidates using floating operations commensurate with running about $n \log n$ steps of the conjugate gradients algorithm. In the absence of such a preconditioner among the candidates, our method can advise the practitioner to use no preconditioner at all. The algorithm is extremely easy to implement and trivially parallelizable, and along the way we provide theoretical improvements to the literature on trace estimation. In empirical experiments, we show the selection method can be quite helpful. For example, it allows us to create to the best of our knowledge the first preconditioning method for kernel regression which never uses more iterations over the non-preconditioned analog in standard settings.

Contents

List of Figures

List of Tables

Acknowledgments

Thank you, Professor Gu, for constantly believing in my ability through all these years. I don't think I'd be where I am now without your help. I'd also like to thank Nicholas Pippenger for an extremely fruitful discussion early in my progress, and Karl Wimmer for helping me to understand the main result in Wimmer et al. (2014). And of course, I couldn't be where I am today without the continued and extreme support and happiness given by my friends and family.

Chapter 1

Introduction

For a minute, pretend you are a data scientist working for an insurance company. Due to an increase in the incidence of fire on California's coast, more people are looking to buy home insurance. Unfortunately, the current system relies on humans to estimate the value of each property before individuals can be insured, and as a result thousands of people are being asked to wait months before they can feel safe about their homes. To resolve this issue, your boss asks you to build a model to predict the value of a home, which can get people in the system much quicker before any potential fine tuning is needed. You have access to a collection of 500 features about previous customers' homes (the number of bedrooms, the distance to the nearest school, sale price of nearby homes, etc.), as well as how much the human valued the home at previously, adjusted for inflation and other economic factors. For each customer i of all $15,000,000$ previous valuations, you concatenate these features into a vector $\boldsymbol{x}_i \in \mathbb{R}^{500}$, and aim to predict the logarithm $y_i \in \mathbb{R}$ of the value, in hundreds of thousands of dollars, of the house.

The task at hand is known as 'machine learning,' the act[1] of taking in a large amount of historical data to create a model that can predict the outcome of new, unseen situations. This is intimately related to statistics, where we aim to infer the structure of the world through historical data. Of course, sometimes these goals align but for the purposes of this thesis we will ignore a philosophical exposition of machine learning as it related to statistics, optimization, and other similar fields and treat it on a case-by-case

[1]This encompasses a large majority of machine learning problems, anyway. Other common tasks appear more statistical in nature, like clustering.

basis.

A common way to treat this housing valuation problem is to create a *linear model*. That is, we wish to find some weights $w_1, w_2, \ldots, w_{500} \in \mathbb{R}$ (that we can concatenate into a vector $\boldsymbol{w} \in \mathbb{R}^{500}$) so that

$$y_i \approx w_1(\boldsymbol{x}_i)_1 + w_2(\boldsymbol{x}_i)_2 + \cdots + w_{500}(\boldsymbol{x}_i)_{500} = \boldsymbol{w}^*\boldsymbol{x}_i$$

for all i from 1 to $15,000,000$. Finding the 'best' weights $\boldsymbol{w}$ that make this approximation property hold all comes down to how we define best. We could, for example, choose $\boldsymbol{w}$ to minimize the worst-case error between our predicted house price and the truth:

$$\max |y_i - \boldsymbol{w}^*\boldsymbol{x}_i|.$$

This is known as ℓ^∞ regression and reduces quite directly to a linear program. Another practitioner might decide that the 'best' weights minimize the expected deviation over our training set:

$$\frac{1}{15,000,000} \sum_{i=1}^{15,000,000} |y_i - \boldsymbol{w}^*\boldsymbol{x}_i|.$$

This is known as ℓ^1 regression, and again reduces to a linear program. While this last procedure is quite resilient to outliers, the most common approach is to minimize the expected *squared* deviation (the standard deviation of our error):

$$\frac{1}{15,000,000} \sum_{i=1}^{15,000,000} |y_i - \boldsymbol{w}^*\boldsymbol{x}_i|^2.$$

If we construct a matrix $\boldsymbol{A} \in \mathbb{R}^{15,000,000 \times 500}$ which has rows $\boldsymbol{x}_i$, and create a vector $\boldsymbol{y}$ with entries y_i, this is equivalent to minimizing the Euclidean norm of the residual vector

$$\|\boldsymbol{A}\boldsymbol{w} - \boldsymbol{y}\|_2^2$$

across $\boldsymbol{w}$. This approach is known as *least-squares* or ℓ^2 regression, and fortunately the solution can be found in closed form as the solution to the positive semi-definite linear system $\boldsymbol{A}^*\boldsymbol{A}\boldsymbol{w} = \boldsymbol{A}^*\boldsymbol{y}$.

What we just observed is exceedingly common: a machine learning problem, a question about predicting the future based on historical data, reduced itself to a numerical linear algebra problem of solving a linear system. Indeed, even if we wanted to consider the ℓ^1 or ℓ^∞ version of these problems,

the resulting linear programs can be efficiently solved using interior point methods which at the end of day are just a sequence of solutions to a linear system.

After this common reduction, the real question becomes one of numerics. How do we efficiently solve the so-called *normal-equations* $\boldsymbol{A}^*\boldsymbol{A}\boldsymbol{w} = \boldsymbol{A}^*\boldsymbol{y}$ for $\boldsymbol{w}$ when even forming $\boldsymbol{A}^*\boldsymbol{A}$ exactly takes $7,499,999,750,000$ floating point operations (flops)? (Fast computers today can only compute about a hundredth of these operations in a second, while in reality computation would take a whole lot longer since the data wouldn't fit into memory at around 30 *terabytes*.) This ignores the numerical issues associated with computing least squares solutions via the normal equations, which involves severe floating point error when the condition number of the matrix $\boldsymbol{A}$ is on the order of one over the square root of the floating point precision, as opposed to other methods involving a QR-factorization with much better stability bounds but higher computational cost.

The field of *randomized* numerical linear algebra was reborn in the last decade to solve problems just like this. For example, instead of solving the least squares problem exactly, we allow some approximation error, and through this subsample our data from $15,000,000$ initial points to, say, $10,000$ with which to do the actual computation. Such an algorithm has the possibility to fail (see Section 2.1 and Section 3.1.1), but these randomized approaches often allow the user to explicitly control this failure probability to be vanishingly small. The common subsampling approach is known as *leverage score sampling* in the literature (see Ma et al. (2015)), while a related generalization known as sketching (see Woodruff (2014)) allows the user to make a single pass over the dataset when computing a least squares solution.

Sketching is the paradigm we will use most in this thesis, so we'll introduce it before outlining the organization of this report. If $\boldsymbol{S} \in \mathbb{R}^{10,000\times 15,000,000}$ is a matrix with entries which are independent and distributed so that $\boldsymbol{S}_{ij} = \pm 1/\sqrt{15,000,000}$ with equal probability, then one can show (see Woodruff (2014)) that $\|\boldsymbol{S}\boldsymbol{A}\boldsymbol{w} - \boldsymbol{S}\boldsymbol{y}\|_2^2 = (1 \pm \epsilon)\|\boldsymbol{A}\boldsymbol{w} - \boldsymbol{y}\|_2^2$ for *all* $\boldsymbol{w}$ with high probability, where $\epsilon < 1/2$ is some small constant. Computing $\boldsymbol{S}\boldsymbol{A}$ and $\boldsymbol{S}\boldsymbol{y}$ can be done in an online fashion as we pass over the dataset, potentially in parallel. At this point the solution to the much smaller least squares problem $\min \|\boldsymbol{S}\boldsymbol{A}\boldsymbol{w} - \boldsymbol{S}\boldsymbol{y}\|_2^2$ gives us a $1+\epsilon$ approximation to the original problem (with high probability), with a much smaller computational cost. One can make these algorithms useful in practice by changing the sketching matrix $\boldsymbol{S}$ to make the computation of $\boldsymbol{S}\boldsymbol{A}$ possible in about $2\,\text{NNZ}(A)$ floating point operations.

This thesis in general will consider using techniques like the above, though instead of solving optimization problems we will compute scalar quantities which are impractical to compute via traditional means. Chapter 2 will discuss the theory and practice of computing traces of matrices which are most easily accessed via matrix-vector products instead of their individual elements. Using this theory, Chapter 3 will create the first practical algorithms for computing the so-called 'preconditioner stability,' which generalizes to the problem of selecting a preconditioner to use in the preconditioned conjugate gradients algorithm. As a corollary of this work, we create state-of-the-art preconditioned solvers for the kernel/Gaussian Process regression problem, illustrating again how this theory is applicable to the world of statistics/machine learning.

Notation

Boldface letters like $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{b}$ denote vectors while their upper-case analogues such as $\boldsymbol{A}, \boldsymbol{P}, \boldsymbol{Q}$ denote matrices. Such matrices and vectors can be either random or deterministic. The underlying scalar field $\mathbb{F}$ is either the real numbers $\mathbb{R}$ or the complex numbers $\mathbb{C}$ unless otherwise specified. The adjoint of a matrix is denoted by $\boldsymbol{A}^*$. The inner product of two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ is denoted as $\boldsymbol{x}^*\boldsymbol{y}$. The expectation of a random variable will be denotes by $\mathbf{E}$, and the probability of an event E is written $\mathbf{P}(E)$. The norms $\|\boldsymbol{A}\|_{\mathsf{F}}$ and $\|\boldsymbol{x}\|_2$ represent the Frobenius and Euclidean norms, respectively. The Schatten p-norm of a matrix will be denoted as $\|\boldsymbol{A}\|_p$ when necessary. In particular, $\|\boldsymbol{A}\|_{\mathsf{F}} = \|\boldsymbol{A}\|_2$.

Chapter 2

Results on Trace Estimation

The trace of a matrix $\boldsymbol{A} \in \mathbb{R}^{d \times d}$ is a fundamental quantity of interest when working with data. For example, state-of-the-art matrix completion algorithms rely on minimizing the so-called trace norm (Schatten-1 norm) of a matrix (see Candès and Recht (2009)), and as a result even being able to compute the resulting error in approximation requires the ability to compute the trace. If we have access to the matrix $\boldsymbol{A}$ in memory, computing the trace is easy:

$$\operatorname{tr} \boldsymbol{A} = \sum_{i=1}^{d} \boldsymbol{A}_{ii},$$

which we can complete in $d - 1$ floating point operations.

As soon as the individual elements of $\boldsymbol{A}$ aren't easily accessible, the trace becomes surprisingly difficult to examine. For example, suppose that $\boldsymbol{A}$ is symmetric, and we wish to compute the common squared Frobenius norm $\|\boldsymbol{A}\|_2^2 = \operatorname{tr} \boldsymbol{A}^* \boldsymbol{A} = \operatorname{tr} \boldsymbol{A}^2$. Since the trace of a matrix is the sum of its eigenvalues, we could perhaps eigendecompose $\boldsymbol{A} = \boldsymbol{U} \boldsymbol{\Lambda} \boldsymbol{U}^*$ for orthogonal $\boldsymbol{U}$ and diagonal $\boldsymbol{\Lambda}$, at which point we could compute $\operatorname{tr} \boldsymbol{A}^2 = \sum_{i=1}^{d} \boldsymbol{\Lambda}_{ii}^2$ exactly. This takes on the order d^3 floating point operations (see Section 2.3.1 for related analysis). A simpler solution with the same computational complexity would be to compute $\boldsymbol{A}^2$ exactly using $2d^3 - d^2$ floating point operations, and then compute the trace of this matrix with an additional $d - 1$ floating point operations. This isn't an improvement, but suggests a significantly easier procedure by realizing that we don't actually need to

compute most elements of A^2, we only need to consider

$$(A^2)_{ii} = \sum_{j=1}^{d} A_{ij}A_{ji} = \sum_{j=1}^{d} A_{ij}^2.$$

Thus

$$\|A\|_2^2 = \operatorname{tr} A^2 = \sum_{i=1}^{d} (A^2)_{ii} = \sum_{i=1}^{d}\sum_{j=1}^{d} A_{ij}^2 = \sum_{ij} A_{ij}^2$$

can be computed exactly in $2d^2 - 1$ floating point operations.

While this suggested algorithm for computing $\operatorname{tr} A^2$ runs in *input sparsity time* – taking on the order of $\textsc{nnz}(A)$ floating point operations – once we try to compute $\|A\|_4^4 = \operatorname{tr} A^4$ exactly this exhaustive list of practical algorithms will take d^3 floating point operations, and d^3 time algorithms don't scale well with large datasets. The pain point in computing these traces practically is that computing $A_{ii} = e_i^*(Ae_i)$ is about as slow as computing a generic matrix-vector product Ax for any input vector x, whence

$$\operatorname{tr} A = \sum_{i=1}^{d} e_i^*(Ae_i) = \frac{1}{d}\sum_{i=1}^{d} (\sqrt{d}e_i)^* A(\sqrt{d}e_i) = \mathbf{E}\, x^* Ax$$

where in the last expression $x \sim \text{Uniform}\{\sqrt{d}e_1, \sqrt{d}e_2, \ldots, \sqrt{d}e_n\}$. If Ax takes on the order of d^2 floating point operations to compute (which is quite often the case, including when A is a power of some given matrix; see Section 2.3.1), the resulting Monte-Carlo estimate of the trace

$$\operatorname{tr} A \approx T_m = \frac{1}{m}\sum_{i=1}^{m} x_i^* Ax_i$$

where the k vectors $x_i \sim \text{Uniform}\{\sqrt{d}e_1, \sqrt{d}e_2, \ldots, \sqrt{d}e_d\}$ are independently and identically distirbuted will give an increasingly accurate estimate of $\operatorname{tr} A$ as $k \to \infty$ using on the order of md^2 floating point operations of work, less than the seemingly necessary d^3 if m is significantly less than d.

This chapter considers these stochastic estimators of the trace, largely dating back to the seminal work of Hutchinson (1990), in serious detail. We characterize precisely which distributions on vectors x can give rise to Monte-Carlo style estimators like T_n above. Restricting consideration to positive definite matrices A, we further provide asymptotically optimal

upper and lower bounds on the performance of these Monte-Carlo style estimators. Since these asymptotic bounds are a bit unwieldy when actually choosing constants in practice, we show that these lower bounds reflect refined, explicit upper bounds bounds on the sample complexity m needed to achieve with-high-probability relative error bounds we construct for suitably sub-Gaussian distributions on $\boldsymbol{x}$ that improve upon the current state-of-the-art bounds given in Roosta-Khorasani and Ascher (2015). Finally, we conclude by verifying our theoretical results with empirical evidence.

2.1 Why Deterministic Algorithms Can't Work

The most important consideration when creating trace estimation algorithms for matrices where computing an element of $\boldsymbol{A}$ is as hard as computing a matrix-vector product $\boldsymbol{Ax}$ is determining how many matrix-vector products $\boldsymbol{Ax}$ we'd need to compute to accurately estimate the trace of a given matrix. This notion requires a definition of "accuracy." Relying on the convention in the numerics community, (see Golub and Van Loan (2012)) for a given $\epsilon > 0$ we will say that a trace estimator T_m satisfies ϵ-relative error if

$$|T_m - \operatorname{tr} \boldsymbol{A}| \leq \epsilon \operatorname{tr} \boldsymbol{A}.$$

Naturally, numerical algorithms ought to work – satisfy a given relative error – for *all* inputs of a certain class. In this chapter, for instance, we will consider mostly positive definite matrices $\boldsymbol{A}$. Unfortunately, we can show that *any* deterministic algorithm – even if it uses infinite computation power and only works for rank at-most-one positive semi-definite matrices – used to compute the trace of $\boldsymbol{A}$ via a series of possibly adaptive queries to a matrix-vector multiply oracle $O(\boldsymbol{x}) : \boldsymbol{x} \mapsto \boldsymbol{Ax}$ needs at least d queries to achieve any $\epsilon < 1$ relative error. Since we can always compute

$$\operatorname{tr} \boldsymbol{A} = \sum_{i=1}^{d} \boldsymbol{e}_i^* (\boldsymbol{A} \boldsymbol{e}_i) = \sum_{i=1}^{n} \boldsymbol{e}_i^* O(\boldsymbol{e}_i)$$

exactly using d queries, this means that complicated deterministic algorithms of this type can never give speedups over conventional computations. As a result, we have strong motivation to consider randomized procedures which *can* give such speedups.

Definition 2.1 (Deterministic Trace Estimator)**.** Fix a given matrix $\boldsymbol{A}$ and define the function T_m as follows. For some deterministic function f_1 :

$\mathbb{R}^{d\times d} \to \mathbb{R}^d$, we compute $\boldsymbol{x}_1 = O(f_1(\boldsymbol{A}))$. Inductively, we then compute $\boldsymbol{x}_k = O(f_k(\boldsymbol{A}, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{k-1}))$ for some deterministic function $f_k : \mathbb{R}^{d\times d} \times \mathbb{R}^d \times \cdots \times \mathbb{R}^d \to \mathbb{R}^d$ for $k = 2, 3, \ldots, m$ before returning T_m as some deterministic function of $\boldsymbol{A}, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_m$. Such a function T_m is called a deterministic trace estimator.

Again, note that the above definition allows for T_m to take potentially infinite computation time.

Theorem 2.1. Fix $0 < \epsilon < 1$. Any deterministic trace estimator T_m satisfying

$$|T_m - \operatorname{tr} \boldsymbol{A}| \leq \epsilon \operatorname{tr} \boldsymbol{A}$$

for all positive semi-definite matrices $\boldsymbol{A} \in \mathbb{F}^{d\times d}$ of rank at most one requires $m \geq n$.

Proof. We will follow via a resisting oracle. In particular, let's suppose that O returns $O(\boldsymbol{x}_i) = 0$ for the first $d-1$ queries $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{d-1}$. Take $\boldsymbol{v} \in \mathbb{R}^d$ to be orthogonal to $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{d-1}$. At this point, both the zero matrix $\boldsymbol{A}_1 = 0$ and the orthogonal projection $\boldsymbol{A}_2$ onto the span of $\boldsymbol{v}$ would return the 0 for these queries. (Note moreover that $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ have rank at most one.) If we output an estimator $T_m = T_{d-1}$ which satisfied

$$|T_m - \operatorname{tr} \boldsymbol{A}| \leq \epsilon \operatorname{tr} \boldsymbol{A}$$

for all positive semi-definite matrices $\boldsymbol{A} \in \mathbb{F}^{d\times d}$, then since $\boldsymbol{A}_1$ is positive semi-definite this series of responses by O would ensure

$$|T_m| = |T_m - \operatorname{tr} \boldsymbol{A}_1| \leq \epsilon \operatorname{tr} \boldsymbol{A}_1 = 0,$$

so $T_m = 0$. On the other hand, the matrix $\boldsymbol{A}_2$ would give the same sequence of responses by O and so by determinism

$$\operatorname{tr} \boldsymbol{A}_2 = |\operatorname{tr} \boldsymbol{A}_2| = |T_m - \operatorname{tr} \boldsymbol{A}_2| \leq \epsilon \operatorname{tr} \boldsymbol{A}_2 < \operatorname{tr} \boldsymbol{A}_2,$$

a contradiction. It follows that at least n queries to O are needed to guarantee this uniform error bound deterministically. ∎

2.2 Preliminary Results on Trace Estimation

We've shown that determinism is necessary for any accurate trace estimator using the matrix-vector product oracle, but which distributions on $\boldsymbol{x}_i$ allow

for the estimator

$$T_m = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{x}_i^* \boldsymbol{A}\boldsymbol{x}_i$$

to quickly converge to the trace of $\boldsymbol{A}$? Taking the limit as m goes to ∞, the law of large numbers tells us $T_m \to \mathbf{E}\, \boldsymbol{x}_i^* \boldsymbol{A}\boldsymbol{x}_i$ almost surely if each $\boldsymbol{x}_i$ are independent and identically distributed. Thus, it is necessary and sufficient that $\mathbf{E}\, \boldsymbol{x}_i^* \boldsymbol{A}\boldsymbol{x}_i = \operatorname{tr} \boldsymbol{A}$ for these Monte-Carlo estimators to converge. The following Theorem says that even if we allow our algorithm to only work for positive semi-definite matrices of rank at most two, asking whether a Hutchinson-type estimator with identically and independently distributed $\boldsymbol{x}_i$ converges to $\operatorname{tr} \boldsymbol{A}$ is equivalent to asking whether $\mathbf{E}\, \boldsymbol{x}_i \boldsymbol{x}_i^* = \boldsymbol{I}$.

Theorem 2.2. $\mathbf{E}\, \boldsymbol{x}^* \boldsymbol{A}\boldsymbol{x} = \operatorname{tr} \boldsymbol{A}$ for all positive semi-definite $\boldsymbol{A} \in \mathbb{F}^{d\times d}$ of rank at most two if and only if $\mathbf{E}\, \boldsymbol{x}\boldsymbol{x}^* = \boldsymbol{I}$.

Proof. By linearity and the cyclic properties of the trace,

$$\mathbf{E}\, \boldsymbol{x}^* \boldsymbol{A}\boldsymbol{x} = \mathbf{E} \operatorname{tr}(\boldsymbol{x}^* \boldsymbol{A}\boldsymbol{x}) = \mathbf{E} \operatorname{tr}(\boldsymbol{A}\boldsymbol{x}\boldsymbol{x}^*) = \operatorname{tr}\big(\boldsymbol{A}(\mathbf{E}\, \boldsymbol{x}\boldsymbol{x}^*)\big)$$

Let $\boldsymbol{\Sigma} = \mathbf{E}\, \boldsymbol{x}\boldsymbol{x}^*$. If $\boldsymbol{\Sigma} = \boldsymbol{I}$ then clearly $\mathbf{E}\, \boldsymbol{x}^* \boldsymbol{A}\boldsymbol{x} = \operatorname{tr}(\boldsymbol{A}\boldsymbol{\Sigma}) = \operatorname{tr}(\boldsymbol{A})$ for all $\boldsymbol{A} \in \mathbb{F}^{d\times d}$. On the other hand, if

$$\operatorname{tr}(\boldsymbol{A}\boldsymbol{\Sigma}) = \operatorname{tr}(\boldsymbol{A})$$

for all positive *semi*-definite $\boldsymbol{A} \in \mathbb{F}^{d\times d}$ then in particular

$$\boldsymbol{\Sigma}_{ii} = \boldsymbol{e}_i^* \boldsymbol{\Sigma} \boldsymbol{e}_i = \operatorname{tr}(\boldsymbol{e}_i \boldsymbol{e}_i^* \boldsymbol{\Sigma}) = \operatorname{tr}(\boldsymbol{e}_i \boldsymbol{e}_i^*) = 1.$$

But then

$$2 + 2\boldsymbol{\Sigma}_{ij} = \boldsymbol{\Sigma}_{ii} + \boldsymbol{\Sigma}_{jj} + 2\boldsymbol{\Sigma}_{ij} = \operatorname{tr}\big((\boldsymbol{e}_i + \boldsymbol{e}_j)(\boldsymbol{e}_i + \boldsymbol{e}_j)^* \boldsymbol{\Sigma}\big) = \operatorname{tr}(\boldsymbol{e}_i + \boldsymbol{e}_j)(\boldsymbol{e}_i + \boldsymbol{e}_j)^* = 2$$

by the same logic so $\boldsymbol{\Sigma}_{ij} = 0$ for $i \neq j$. In sum $\boldsymbol{\Sigma} = \boldsymbol{I}$ if $\mathbf{E}\, \boldsymbol{x}^* \boldsymbol{A}\boldsymbol{x} = \operatorname{tr} \boldsymbol{A}$ for all positive *semi*-definite $\boldsymbol{A}$. ∎

By constructing positive semi-definite matrices as a limit of strictly positive definite matrices (e.g. by adding $\boldsymbol{I}/k$ and letting $k \to \infty$) this result also holds if the class of positive semi-definite matrices of rank at most two is replaced with the class of positive definite matrices.

In light of the above Theorem and the seminal work of Hutchinson (1990), we will define a Hutchinson-type estimator to be a Monte-Carlo trace estimator using independent and identically distributed queries which is guaranteed to converge in the large-query limit.

Definition 2.2 (Hutchinson-Type Estimator)**.** If $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$ are a sequence of independent and identically distributed copies of some real random vector $\boldsymbol{x}$ satisfying $\mathbf{E}\,\boldsymbol{x}\boldsymbol{x}^* = \boldsymbol{I}$, then for all $m = 1, 2, \ldots$ we call the random variable

$$T_m = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{x}_i^* \boldsymbol{A}\boldsymbol{x}_i$$

a Hutchinson-type estimator for the trace of $\boldsymbol{A}$.

We follow this definition with a couple examples of distributions on $\boldsymbol{x}_i$ which give rise to Hutchinson-type estimators. Some of these have special names and as a result will be given special notation like H_d or G_d instead of T_d.

Example 2.1 (Hutchinson Estimator)**.** Consider a Rademacher vector $\boldsymbol{x} \sim \text{Uniform}\{\pm 1\}^d$. We can compute $\mathbf{E}\,\boldsymbol{x}_i^2 = \mathbf{E}\,1 = 1$ for any $i = 1, 2, \ldots, d$, so the diagonal elements of $\mathbf{E}\,\boldsymbol{x}\boldsymbol{x}^*$ are all one. On the other hand, by independence of the elements $\mathbf{E}\,\boldsymbol{x}_i\boldsymbol{x}_j = \mathbf{E}\,\boldsymbol{x}_i\,\mathbf{E}\,\boldsymbol{x}_j = 0 \cdot 0 = 0$ for $i \neq j$ and hence $\mathbf{E}\,\boldsymbol{x}\boldsymbol{x}^* = \boldsymbol{I}$. The resulting Hutchinson-type estimator is called the Hutchinson estimator, named after the seminal work of Hutchinson (1990), and will be denoted H_m.

Example 2.2 (Gaussian Estimator)**.** If $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ is a standard Gaussian vector then $\mathbf{E}\,\boldsymbol{x}\boldsymbol{x}^* = \mathbf{E}(\boldsymbol{x} - \mathbf{E}\,\boldsymbol{x})(\boldsymbol{x} - \mathbf{E}\,\boldsymbol{x})^* = \boldsymbol{I}$ as desired. The resulting Hutchinson-type trace estimator is called the Gaussian estimator, and is denoted G_m.

Example 2.3 (Sparse Trace Estimators)**.** Suppose the vector $\boldsymbol{x}$ has independent elements distributed so that

$$\boldsymbol{x}_j \sim \begin{cases} -\sqrt{c} & \text{with prob. } \frac{1}{2c} \\ \sqrt{c} & \text{with prob. } \frac{1}{2c} \\ 0 & \text{otherwise.} \end{cases}$$

for some $c > 0$. Computations just as in the Hutchinson case show that $\mathbf{E}\,\boldsymbol{x}\boldsymbol{x}^* = \boldsymbol{I}$. This estimator is not commonly used, but conveniently it results in query vectors which in expectation are $n\frac{c-1}{c}$-sparse. That is, in expectation $\boldsymbol{x}$ has $n\frac{c-1}{c}$ nonzero entries. Since we will see that taking $c \leq 3$ preserves the upper bound on convergence of the Gaussian and Hutchinson estimators exactly for this class, these input vectors give simple-to-construct queries which are sparse enough to ignore up to 2/3 of the matrix in some settings.

The following result says further that such a Hutchinson-type estimator cannot exactly compute the trace of all positive definite matrices. We will rely on this simple sounding result when coming up with extremely tight asymptotic lower bounds for certain common trace estimators in Section 2.4. The proof is similar to Theorem 2.2 above.

Theorem 2.3. If the real random vector $\boldsymbol{x}$ satisfies $\mathbf{E}\,\boldsymbol{x}\boldsymbol{x}^* = \boldsymbol{I}$, then $\operatorname{Var}(\boldsymbol{x}^*\boldsymbol{A}\boldsymbol{x}) > 0$ for some positive semi-definite $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ of rank at most two.

Proof. In light of Theorem 2.2, suppose to the contrary that $\boldsymbol{x}^*\boldsymbol{A}\boldsymbol{x} = \operatorname{tr}\boldsymbol{A}$ almost surely for every positive semi-definite $\boldsymbol{A} \in \mathbb{R}^{n\times n}$. In particular,

$$\boldsymbol{x}_i^2 = \boldsymbol{x}^*\boldsymbol{e}_i\boldsymbol{e}_i^*\boldsymbol{x} = \operatorname{tr}\boldsymbol{e}_i\boldsymbol{e}_i^* = 1$$

almost surely for all $i = 1, 2, \ldots, n$, so $\boldsymbol{x} \in \{-1, 1\}^d$ almost surely. On the other hand, if $\boldsymbol{x}_i \neq \boldsymbol{x}_j$ with positive probability for some $i \neq j$ we would have

$$0 = (\boldsymbol{x}_i + \boldsymbol{x}_j)^2 = \boldsymbol{x}^*(\boldsymbol{e}_i + \boldsymbol{e}_j)(\boldsymbol{e}_i + \boldsymbol{e}_j)^*\boldsymbol{x} = \operatorname{tr}(\boldsymbol{e}_i + \boldsymbol{e}_j)(\boldsymbol{e}_i + \boldsymbol{e}_j)^* = 2$$

with positive probability, a contradiction, so almost surely every component of $\boldsymbol{x}$ is the same. But then $0 = \mathbf{E}\,\boldsymbol{x}_i\boldsymbol{x}_j = \mathbf{E}\,\boldsymbol{x}_i^2 = 1$, contradicting our assumption that $\mathbf{E}\,\boldsymbol{x}\boldsymbol{x}^* = \boldsymbol{I}$. ∎

2.3 Upper Bounds on Sample Complexity

Even though we understand how to construct trace estimators T_m which can estimate traces accurately in a limiting sense with $m \to \infty$, as of yet we don't have any concrete evidence to suggest that this mode of computation is quicker than other methods. In Section 2, for example, stochastic trace estimation would only help us compute $\|\boldsymbol{A}\|_4^4 = \operatorname{tr}\boldsymbol{A}^4$ for a positive definite matrix $\boldsymbol{A}$ if m is sublinear in n. (For example, if $m \geq \log n$ or $m \geq 100$ gave us some desired accuracy, then we would achieve asymptotic superiority over the other obvious algorithms by using these trace estimates.) This section guarantees this is the case: indeed the necessary m to achieve an $\epsilon < 1/2$ relative error trace estimate with probability at least $1 - \delta$ is just $\frac{6}{\epsilon^2}\log\frac{2}{\delta}$ for all the example Hutchinson-type estimators referenced in Section 2.2.

The idea of analysing the non-asymptotic concentration of these Hutchinson-type trace estimators T_m started with Avron and Toledo (2011) after Avron (2010) realized that such bounds would be useful in achieving error bounds in novel algorithms for computing the number of triangles in large graphs.

x_{ij} Distribution	Sample Complexity		
	This Work	[RoostaAscher]	[AvronToledo]
$\mathcal{N}(0,1)$	$\frac{6}{\epsilon^2}\log\frac{2}{\delta}$	$\frac{8}{\epsilon^2}\log\frac{2}{\delta}$	$\frac{20}{\epsilon^2}\log\frac{2}{\delta}$
Uniform$\{\pm 1\}$	$\frac{6}{\epsilon^2}\log\frac{2}{\delta}$	$\frac{6}{\epsilon^2}\log\frac{2}{\delta}$	$\frac{6}{\epsilon^2}\log\frac{2\,\mathrm{rank}(\boldsymbol{A})}{\delta}$
$\pm\sqrt{3}$ wp $\frac{1}{3}$, 0 ow	$\frac{6}{\epsilon^2}\log\frac{2}{\delta}$	$(\frac{6}{\epsilon^2}\log\frac{2}{\delta})$	—
$\pm\sqrt{2}$ wp $\frac{1}{2}$, 0 ow	$\frac{6}{\epsilon^2}\log\frac{2}{\delta}$	—	—
$x_{ij} \overset{\text{dist}}{=} -x_{ij}$, $\mathbf{E}\, x_{ij}^k \leq \mathbf{E}\, Z^k$	$\frac{6}{\epsilon^2}\log\frac{2}{\delta}$	—	—

Table 2.1 Table summary of sample complexity bounds for Hutchinson-type trace estimators using random vectors where each element is identically and independently distributed as the given x_{ij}. Note that while the result for x_{ij} as $\pm\sqrt{3}$ with probability $1/3$ and zero otherwise is evident from the proof of Roosta-Khorasani and Ascher (2015), neither the result nor this distribution are mentioned, hence the parenthesis. Here, $Z \sim \mathcal{N}(0,1)$ and $k = 1, 2, \ldots$ ranges over all positive integers.

Originally, error analysis for the trace estimation procedures starting with Hutchinson (1990) relied solely on controlling the variance of the (unbiased) estimator T_m of $\mathrm{tr}\,\boldsymbol{A}$. This is a reasonable request, and motivated Hutchinson to use Rademacher queries in the first place since it results in minimal variance for identically and independently distributed inputs. That said, answering the natural question "How many matrix-vector products do I need to be 90% confident that my approximation to $\mathrm{tr}\,\boldsymbol{A}$ is within 1/5-relative error of the true value?" directly makes much more sense for algorithmic problems where this is often the question of interest. As a result, we will ignore the variance and focus largely on these concentration guarantees.

This section will extend the state-of-the-art results from Roosta-Khorasani and Ascher (2015), which in turn built on the seminal work of Avron and Toledo (2011) to give tight bounds on the sample complexity[1] of the Gaussian and Hutchinson estimators G_m and H_m. The original sample complexity bound for the Hutchinson estimator H_m given by Avron and Toledo (2011) depended logarithmically on n in the worst case, which was tightened to match the dimension-independent complexity of the Gaussian estimator by Roosta-Khorasani and Ascher (2015). Roosta-Khorasani and Ascher (2015) rely strongly on results from Achlioptas (2001) detailing finite-sample

[1]This is the minimal m needed for ϵ relative error and $1-\delta$ confidence that this error bound is satisfied for *any* fixed positive definite input matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$

guarantees for Rademacher Johnson-Lindenstrauss embeddings. Potentially unknown to Roosta-Khorasani and Ascher (2015) is that by outsourcing the heavy lifting of their proof of Hutchinson estimator concentration they also proved the same sample complexity for the sparse queries used in Example 2.3 when $c = 3$. Our work extends this to any $0 < c \leq 3$, allowing half-zero query vector Hutchinson-type estimators with fast convergence guarantees and only twice as many bits of randomness needed as in the Hutchinson case (for $c = 2$). A summary of this contribution is given in Table 2.1. As we will show in Section 2.4, these upper bounds (in their full form) on sample complexity are tight even up to the leading constant.

We start with our complexity bound for the Gaussian estimator. This proof is modeled after the complexity bound for the Hutchinson estimator from Roosta-Khorasani and Ascher (2015), though it does improve on their result for the Gaussian estimator. This is the first time that the main result from Roosta-Khorasani and Ascher (2015) has been presented in complete on it's own, without outsourcing to another text. As a result we have been a tad more pedantic than necessary.

Theorem 2.4. If $\boldsymbol{A} \in \mathbb{R}^{d\times d}$ is positive semi-definite, then

$$\mathbf{P}\big(|G_m - \operatorname{tr}\boldsymbol{A}| > \epsilon \operatorname{tr}\boldsymbol{A}\big) < 2e^{-\frac{m}{2}\left(\frac{\epsilon^2}{2}-\frac{\epsilon^3}{3}\right)}.$$

In other words, if $m \geq \frac{12\log(\frac{2}{\delta})}{\epsilon^2(3-2\epsilon)}$ then the relative error $|G_m - \operatorname{tr}\boldsymbol{A}| \leq \epsilon \operatorname{tr}\boldsymbol{A}$ with probability at least $1-\delta$. Moreover, if $\epsilon < \frac{1}{2}$, then $m \geq \frac{6}{\epsilon^2}\log\frac{2}{\delta}$ implies the relative error $|G_m - \operatorname{tr}\boldsymbol{A}| \leq \epsilon \operatorname{tr}\boldsymbol{A}$ with probability at least $1-\delta$.

Proof. Diagonalize $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^*$ where $\boldsymbol{U}$ is unitary and $\boldsymbol{\Lambda}$ is diagonal. Let

$z_i = \boldsymbol{U}^* g_i$ so that

$$\begin{aligned}
\mathbf{P}\big(G_m > (1+\epsilon)\operatorname{tr}\boldsymbol{A}\big) &= \mathbf{P}\left(\sum_{i=1}^{m} g_i^* \boldsymbol{A} g_i > m(1+\epsilon)\operatorname{tr}\boldsymbol{A}\right) \\
&= \mathbf{P}\left(\sum_{i=1}^{m} z_i^* \boldsymbol{\Lambda} z_i > m(1+\epsilon)\operatorname{tr}\boldsymbol{A}\right) \\
&= \mathbf{P}\left(\sum_{i=1}^{m}\sum_{j=1}^{n} \lambda_j z_{ij}^2 > m(1+\epsilon)\operatorname{tr}\boldsymbol{A}\right) \\
&= \mathbf{P}\left(\sum_{j=1}^{n} \frac{\lambda_j}{\operatorname{tr}\boldsymbol{A}} \sum_{i=1}^{m} z_{ij}^2 > m(1+\epsilon)\right) \\
&= \mathbf{P}\left(\exp\left(t\sum_{j=1}^{n} \frac{\lambda_j}{\operatorname{tr}\boldsymbol{A}} \sum_{i=1}^{m} z_{ij}^2\right) > e^{(1+\epsilon)mt}\right) \\
&\le e^{-(1+\epsilon)mt}\ \mathbf{E}\exp\left(\sum_{j=1}^{n} \frac{\lambda_j}{\operatorname{tr}\boldsymbol{A}} \sum_{i=1}^{m} t z_{ij}^2\right)
\end{aligned}$$

for $t > 0$ by Markov's inequality. (Wasserman, 2013: Thm 4.1) Now by convexity of the exponential and linearity of expectation

$$\mathbf{E}\exp\left(\sum_{j=1}^{n} \frac{\lambda_j}{\operatorname{tr}\boldsymbol{A}} \sum_{i=1}^{m} t z_{ij}^2\right) \le \sum_{j=1}^{n} \frac{\lambda_j}{\operatorname{tr}\boldsymbol{A}}\, \mathbf{E}\exp\left(\sum_{i=1}^{m} t z_{ij}^2\right) = \sum_{j=1}^{n} \frac{\lambda_j}{\operatorname{tr}\boldsymbol{A}} \prod_{i=1}^{m} \mathbf{E}\, e^{t z_{ij}^2}$$

since $\lambda_i \ge 0$ and $\sum_{i=1}^{n} \lambda_i = \operatorname{tr}\boldsymbol{A}$, and for a fixed j, z_{ij} are independent and identically distributed. Now, to bound the moment generating function, observe that z_i are just independent standard normal vectors by the rotation invariance of the Gaussian and unitary nature of $\boldsymbol{U}$. This implies that $z_{ij} \sim \mathcal{N}(0,1)$ and hence z_{ij}^2 is just a χ^2 random variable with one degree of freedom. We then know (see Wasserman (2013)) that

$$\mathbf{E}\, e^{t z_{ij}^2} = \frac{1}{\sqrt{1-2t}}$$

so long as $t < 1/2$. This implies that

$$\mathbf{E}\exp\left(\sum_{j=1}^{n} \frac{\lambda_j}{\operatorname{tr}\boldsymbol{A}} \sum_{i=1}^{m} t z_{ij}^2\right) \le \left(\frac{1}{\sqrt{1-2t}}\right)^m$$

for these $0 < t < 1/2$ and hence taking $t = \frac{1}{2}\frac{\epsilon}{1+\epsilon} < 1/2$ we have

$$\mathbf{P}\big(G_m > (1+\epsilon)\operatorname{tr} \boldsymbol{A}\big) \le \left(\frac{1}{\sqrt{1-2t}}\right)^m e^{-(1+\epsilon)mt} = \left((1+\epsilon)e^{-\epsilon}\right)^{m/2} < e^{-\frac{m}{2}\left(\frac{\epsilon^2}{2}-\frac{\epsilon^3}{3}\right)}, \tag{2.1}$$

relying on the fact that $(1+\epsilon)e^{-\epsilon} < e^{\frac{\epsilon^3}{3}-\frac{\epsilon^2}{2}}$ for all $\epsilon > 0$, verifiable by simple calculus. Now we perform largely the same argument for the lower tail.

$$\begin{aligned}
\mathbf{P}\big(G_m < (1-\epsilon)\operatorname{tr} \boldsymbol{A}\big) &= \mathbf{P}\left(\sum_{i=1}^m \boldsymbol{g}_i^* \boldsymbol{A} \boldsymbol{g}_i < m(1-\epsilon)\operatorname{tr} \boldsymbol{A}\right) \\
&= \mathbf{P}\left(\sum_{i=1}^m \boldsymbol{z}_i^* \boldsymbol{\Lambda} \boldsymbol{z}_i < m(1-\epsilon)\operatorname{tr} \boldsymbol{A}\right) \\
&= \mathbf{P}\left(\sum_{i=1}^m \sum_{j=1}^n \lambda_j z_{ij}^2 < m(1-\epsilon)\operatorname{tr} \boldsymbol{A}\right) \\
&= \mathbf{P}\left(\sum_{j=1}^n \frac{\lambda_j}{\operatorname{tr} \boldsymbol{A}} \sum_{i=1}^m z_{ij}^2 < m(1-\epsilon)\right) \\
&= \mathbf{P}\left(-t\sum_{j=1}^n \frac{\lambda_j}{\operatorname{tr} \boldsymbol{A}} \sum_{i=1}^m z_{ij}^2 > -(1-\epsilon)mt\right) \\
&= \mathbf{P}\left(\exp\left(-t\sum_{j=1}^n \frac{\lambda_j}{\operatorname{tr} \boldsymbol{A}} \sum_{i=1}^m z_{ij}^2\right) > e^{-(1-\epsilon)mt}\right) \\
&\le e^{(1-\epsilon)mt}\ \mathbf{E}\exp\left(\sum_{j=1}^n \frac{\lambda_j}{\operatorname{tr} \boldsymbol{A}} \sum_{i=1}^m -t z_{ij}^2\right)
\end{aligned}$$

Let $t > 0$. By the same argument as before,

$$\mathbf{E}\exp\left(\sum_{j=1}^n \frac{\lambda_j}{\operatorname{tr} \boldsymbol{A}} \sum_{i=1}^m -t z_{ij}^2\right) \le \sum_{j=1}^n \frac{\lambda_j}{\operatorname{tr} \boldsymbol{A}}\, \mathbf{E}\exp\left(\sum_{i=1}^m -t z_{ij}^2\right) = \sum_{j=1}^n \frac{\lambda_j}{\operatorname{tr} \boldsymbol{A}} \prod_{i=1}^m \mathbf{E}\, e^{-t z_{ij}^2}.$$

Since

$$\mathbf{E}\, e^{-t z_{ij}^2} = \frac{1}{\sqrt{1+2t}}$$

for any $t > -1/2$, these reductions say that

$$\mathbf{P}\big(G_m < (1-\epsilon)\operatorname{tr} \boldsymbol{A}\big) \le \left(\frac{1}{\sqrt{1+2t}}\right)^m e^{(1-\epsilon)mt} = \left((1-\epsilon)e^{\epsilon}\right)^{m/2} < e^{-\frac{m}{2}\left(\frac{\epsilon^2}{2}-\frac{\epsilon^3}{3}\right)} \tag{2.2}$$

by plugging in $t = \frac{1}{2}\frac{\epsilon}{1-\epsilon} > 0$. The last inequality relies on the scalar estimate $(1-\epsilon)e^{\epsilon} < e^{-\frac{\epsilon^2}{2}+\frac{\epsilon^3}{3}}$ as with the upper tail. Combining Equation 2.1 with this lower tail bound gives

$$\mathbf{P}(|G_m - \operatorname{tr} \boldsymbol{A}| > \epsilon \operatorname{tr} \boldsymbol{A}) < 2e^{-\frac{m}{2}\left(\frac{\epsilon^2}{2}-\frac{\epsilon^3}{3}\right)}.$$

■

We note that the only property of the Gaussian distribution used to get the upper tail concentration is that the projection of a Gaussian vector $\boldsymbol{g}$ onto an arbitrary subspace spanned by a unit vector $\boldsymbol{\mu}$ satisfies

$$\mathbf{E}\, e^{t(\boldsymbol{\mu}^* \boldsymbol{g})^2} \leq \frac{1}{\sqrt{1-2t}}$$

for $0 < t < 1/2$. It so happens that equality holds in this case, but this isn't necessary for the result. In particular let's suppose that for some random vector $\boldsymbol{x}$ the moments satisfy $\mathbf{E}(\boldsymbol{\mu}^* \boldsymbol{x})^k \leq \mathbf{E}(\boldsymbol{\mu}^* \boldsymbol{g})^k$ for all $k = 1, 2, \ldots$. Then

$$\mathbf{E}\, e^{t(\boldsymbol{\mu}^* \boldsymbol{x})^2} = 1 + \sum_{k=1}^{\infty} \frac{t^k}{k!} \mathbf{E}(\boldsymbol{\mu}^* \boldsymbol{x})^{2k} \leq 1 + \sum_{k=1}^{\infty} \frac{t^k}{k!} \mathbf{E}(\boldsymbol{\mu}^* \boldsymbol{g})^{2k} = \mathbf{E}\, e^{t(\boldsymbol{\mu}^* \boldsymbol{g})^2} = \frac{1}{\sqrt{1-2t}}$$

for $0 < t < 1/2$, and so the same bound from Theorem 2.4 holds for estimators G_d using vectors $\boldsymbol{x}_i$ instead of $\boldsymbol{g}_i$. This argument will allow us to prove that the bound in Theorem 2.4 holds for a much larger class of suitably sub-Gaussian vectors.

Theorem 2.5. Let Z be a standard normal variable. For $i = 1, 2, \ldots, m$ suppose that $\boldsymbol{x}_i$ are symmetric, independent, and identically distributed random vectors with independent components $\boldsymbol{x}_{ij}$ satisfying the sub-Gaussian moment growth condition $\mathbf{E}\, \boldsymbol{x}_{ij}^m \leq \mathbf{E}\, Z^m$. Moreover, we assume that each $\boldsymbol{x}_{ij}$ have unit variance: $\mathbf{E}\, \boldsymbol{x}_{ij}^2 = 1$. Then if $\boldsymbol{A}$ is positive semidefinite and

$$T_m = \frac{1}{m} \sum_{i=1}^{m} \boldsymbol{x}_i^* \boldsymbol{A} \boldsymbol{x}_i,$$

we know that

$$\mathbf{P}\big(|T_m - \operatorname{tr} \boldsymbol{A}| > \epsilon \operatorname{tr} \boldsymbol{A}\big) < 2e^{-\frac{m}{2}\left(\frac{\epsilon^2}{2}-\frac{\epsilon^3}{3}\right)}$$

for all $\epsilon > 0$. In particular, $m \geq \frac{12 \log(\frac{2}{\delta})}{\epsilon^2(3-2\epsilon)}$ ensures that the relative error $|T_m - \operatorname{tr} \boldsymbol{A}| < \epsilon \operatorname{tr} \boldsymbol{A}$ with probability at least $1 - \delta$. For simplicity, if we know $0 < \epsilon < 1/2$ taking a larger $m \geq \frac{6}{\epsilon^2} \log \frac{2}{\delta}$ gives the same guarantee.

Proof. By the proof of Theorem 2.4, it suffices to show that the moment growth condition implies that $\mathbf{E}\, e^{t(\mu^* x_i)^2} \leq \mathbf{E}\, e^{tZ^2}$ for any fixed unit vector $\mu \in \mathbb{R}^n$. Let $z_{ij} \sim \mathcal{N}(0,1)$ be independent standard normal variables. Then if

$$I_m = \left\{ (i_1, i_2, \ldots, i_n) \in \mathbb{Z}^n : i_j \geq 0 \text{ is even}, \sum_{j=1}^n i_j = m \right\}$$

we can expand using independence

$$\begin{aligned}
\mathbf{E}(\mu^* x_i)^m &= \sum_{i_1,\ldots,i_n \in I_m} \mu_1^{i_1} \mu_2^{i_2} \cdots \mu_n^{i_n}\ \mathbf{E}\, x_{i1}^{i_1}\ \mathbf{E}\, x_{i2}^{i_2} \cdots \mathbf{E}\, x_{in}^{i_n} \\
&\leq \sum_{i_1,\ldots,i_n \in I_m} \mu_1^{i_1} \mu_2^{i_2} \cdots \mu_n^{i_n}\ \mathbf{E}\, z_{i1}^{i_1}\ \mathbf{E}\, z_{i2}^{i_2} \cdots \mathbf{E}\, z_{in}^{i_n} = \mathbf{E}(\mu^* z_i)^m \\
&= \mathbf{E}\, Z^m
\end{aligned}$$

for all $m = 2, 4, 6, \ldots$. We can ignore the odd powers since if i_j was odd we would have the summand including $\mathbf{E}\, x_{ij}^{i_j} = 0$ drop by symmetry. It follows that the moment generating function

$$\mathbf{E}\, e^{t(\mu^* x_i)^2} = 1 + \sum_{k=1}^{\infty} \frac{t^k}{k!} \mathbf{E}(\mu^* x_i)^{2k} \leq 1 + \sum_{k=1}^{\infty} \frac{t^k}{k!} \mathbf{E}\, Z^{2k} = \mathbf{E}\, e^{tZ^2} = \frac{1}{\sqrt{1-2t}}$$

for $0 \leq t < 1/2$. The proof of Theorem 2.4 then tells us that

$$\mathbf{P}\big(T_m > (1+\epsilon) \operatorname{tr} A\big) < e^{-\frac{m}{2}\left(\frac{\epsilon^2}{2} - \frac{\epsilon^3}{3}\right)}$$

for all $\epsilon > 0$. For the lower tail, the same proof tells us

$$\mathbf{P}\big(T_m < (1-\epsilon) \operatorname{tr} A\big) \leq e^{(1-\epsilon)mt} \sum_{j=1}^n \frac{\lambda_j}{\operatorname{tr} A} \prod_{i=1}^m \mathbf{E}\, e^{-t x_{ij}^2},$$

so it remains to bound $\mathbf{E}\, e^{-t x_{ij}^2}$. Following the suggestion of Achlioptas (2001), we can use the numerical estimate $e^{-t} \leq 1 - t + t^2/2$ for $t > 0$ to give

$$\mathbf{E}\, e^{-t x_{ij}^2} \leq 1 - t\, \mathbf{E}\, x_{ij}^2 + \frac{t^2}{2} \mathbf{E}\, x_{ij}^4 \leq 1 - t + \frac{t^2}{2} \mathbf{E}\, Z^4 = 1 - t + \frac{3}{2} t^2$$

for $t > 0$. Plugging in $t = \frac{1}{2} \frac{\epsilon}{1+\epsilon} > 0$ we have

$$\mathbf{P}\big(T_m < (1-\epsilon) \operatorname{tr} A\big) < e^{-\frac{m}{2}\left(\frac{\epsilon^2}{2} - \frac{\epsilon^3}{3}\right)}.$$

The last inequality $(1 - t + \frac{3}{2}t^2)e^{(1-\epsilon)t} \leq e^{-\frac{1}{2}(\frac{\epsilon^2}{2} - \frac{\epsilon^3}{3})}$ can be verified via series expansion and the assistance of a computer algebra system. A union bound gives the result. ■

2.3.1 Application To Schatten-p Norm Computations

Many common computations involving matrices ask in some detail about the size of the matrices involved, usually in the form of a norm. For example, matrix completion problems like those presented in Candès and Recht (2009) rely on minimizing the Schatten-1 norm of a matrix, which is also known as the nuclear norm. The ubiquitous truncated singular value decomposition finds the closest low rank matrix under the Schatten-2 and Schatten-∞ norms, also known as the Frobenius and operator/spectral norms, respectively. Simply computing these quantities is therefore fundamental, yet as we will see for large matrices this is challenging. Before we can talk about computing these norms, it is necessary to define them.

Definition 2.3. Let $\boldsymbol{A} \in \mathbb{R}^{d\times d}$ be positive semi-definite and $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$ be the eigenvalues (which are the same as the singular values) of the matrix $\boldsymbol{A}$. For $p > 1$ the Schatten-p norm of $\boldsymbol{A}$ is defined as

$$\|\boldsymbol{A}\|_p = \Big(\sum_{i=1}^{d} \lambda_d^p\Big)^{1/p}.$$

For $p = \infty$, a point-wise limit gives the spectral norm

$$\|\boldsymbol{A}\|_\infty = \max_{1\leq i\leq d} \lambda_d.$$

The normal procedure for computing $\|\boldsymbol{A}\|_p$ can be seen in Algorithm 1. Note that for simplicity's sake all algorithms will compute $\|\boldsymbol{A}\|_p^p$ instead of $\|\boldsymbol{A}\|_p$; to find the Schatten norm from this quantity one can just spend ~ 17 flops computing the p-th root. The eigenvalue computation can be completed using (Golub and Van Loan, 2012: Alg 8.3.3), for example. This n^3 dependence is too slow for large matrices; on a 1.4GHz computer the computation time for an arbitrary $\boldsymbol{A}$ is at least a day for $n = 50,000$ and $p = 4$.

Luckily, our trace estimation schemes present an easy way to drop this dependence from n^3 to $n^2 p$. See Algorithm 2 for details. Note that (Woodruff, 2014: Thm 69) (relayed from Li et al. (2014)) gives a less-precise bound for this

Algorithm 1: Naive Schatten norm computation.

Data: A positive semi-definite matrix $\boldsymbol{A} \in \mathbb{R}^{d\times d}$ and an integer $p = 1, 2, \ldots < \infty$.
Result: The Schatten-p norm $\|\boldsymbol{A}\|_p^p$ using $\sim \frac{4}{3}d^3 + dp$ flops.
Overwrite $\boldsymbol{A}$ with $\operatorname{diag}(\lambda_1, \ldots, \lambda_d)$ using $\sim 4n^3/3$ flops.
Compute $\boldsymbol{A} \leftarrow \boldsymbol{A}^p$ using $d(p-1)$ flops.
Compute $\operatorname{tr}\boldsymbol{A} = \sum_{i=1}^n \boldsymbol{A}_{ii}$ using $d-1$ flops.
Return $\|\boldsymbol{A}\|_p^p = \operatorname{tr}\boldsymbol{A}$.

same algorithm. In particular, Li et al. (2014) doesn't allow for variable failure probability δ without resorting to multiple runs of the algorithms, and the constant is much looser at 40 instead of 18 for the failure probability they specify. Also, this version of the algorithm uses far fewer bits of randomness than the Gaussian probing vectors used in Li et al. (2014). Experimental results are given in Section 2.5.

Note that if we wish to get relative error in terms of $\|\boldsymbol{A}\|_p$ instead of $\|\boldsymbol{A}\|_p^p$, the fact that $(1+\epsilon)^{1/p} \asymp 1 + \frac{\epsilon}{p}$ effectively removes the dependence on p. Indeed, as p increases, *fewer* floating point oprations are needed asymptotically. This makes intuitive sense because of the success of the power method in computing $\|\boldsymbol{A}\|_p$.

2.4 Lower Bounds for Hutchinson-Type Estimators

As promised repeatedly, we can exhibit lower bounds which show that our results from Section 2.3 regarding Hutchinson-type estimators are exceedingly tight. The following theorem relies weakly on Theorem 2.3 to say that any Hutchinson-type estimator needs $\Omega(\frac{1}{\epsilon^2}\log\frac{1}{\delta})$ queries to $\boldsymbol{A}$ in order to achieve an ϵ-relative error approximation to $\operatorname{tr}\boldsymbol{A}$ with probability at least $1-\delta$. Recall that W is the Lambert-W function, satisfying $W(x) = \log x - \log\log x + o(1) = \Theta(\log x)$ as $x \to \infty$. (Hoorfar and Hassani, 2007)

Theorem 2.6. Fix $0 < \delta < \frac{1}{10}$. For every Hutchinson-type estimator T_m and every positive semi-definite matrix $\boldsymbol{A} \in \mathbb{R}^{d\times d}$ there exists an $\epsilon_0 > 0$ so that

$$\mathbf{P}\big(|T_m - \operatorname{tr}\boldsymbol{A}| > \epsilon \operatorname{tr}\boldsymbol{A}\big) > \delta$$

whenever $0 < \epsilon < \epsilon_0$ and $m = \lfloor \frac{\operatorname{Var}(\boldsymbol{x}^*\boldsymbol{A}\boldsymbol{x})}{\operatorname{tr}(\boldsymbol{A})^2}\frac{1}{\epsilon^2}W(\frac{2/\pi}{\delta^2})\rfloor = \Theta(\frac{1}{\epsilon^2}\log\frac{1}{\delta})$. Since there exists an $\boldsymbol{A}$ with $\operatorname{Var}(\boldsymbol{x}^*\boldsymbol{A}\boldsymbol{x}) > 0$, this sample complexity dependence on δ

Algorithm 2: Randomized Schatten norm computation.

Data: A positive semi-definite matrix $\boldsymbol{A} \in \mathbb{R}^{d\times d}$ and an integer $p = 1, 2, \ldots < \infty$. User-supplied accuracy parameter $0 < \epsilon < 1/2$ and failure probability $0 < \delta < 1$.

Result: The Schatten-p norm $\|\boldsymbol{A}\|_p^p$ up to ϵ-relative error using $\sim 6\frac{p}{\epsilon^2}d^2 \log\frac{2}{\delta}$ flops. If $\boldsymbol{A}$ is sparse, $\sim 6\frac{p}{\epsilon^2}\,\textsc{nnz}(\boldsymbol{A}) \log\frac{2}{\delta}$ flops are used. $d\lceil\frac{6}{\epsilon^2}\log\frac{2}{\delta}\rceil$ bits of randomness are used.

Compute $m \leftarrow \lceil\frac{6}{\epsilon^2}\log\frac{2}{\delta}\rceil$ using ~ 20 flops.
Sample $\boldsymbol{R} \in \{-1, 1\}^{d\times m}$ so that $\boldsymbol{R}_{ij} = \pm 1$ with equal probability.
foreach $1 \le i \le \lfloor p/2 \rfloor$ **do**
  Compute $\boldsymbol{R} \leftarrow \boldsymbol{A}\boldsymbol{R}$ using $2d^2m - md$ flops.
end
if *p is odd* **then**
  Compute $\boldsymbol{S} \leftarrow \boldsymbol{A}\boldsymbol{R}$ using $2d^2m - md$ flops.
  Return $\frac{1}{m}\sum_{i=1}^{d}\sum_{j=1}^{m}\boldsymbol{S}_{ij}\boldsymbol{R}_{ij}$ using $2md$ flops.
else
  Return $\frac{1}{m}\sum_{i=1}^{d}\sum_{j=1}^{m}\boldsymbol{R}_{ij}^2$ using $2md$ flops.
end

and ϵ is uniform across all Hutchinson-type estimators.

Proof. If $Z \sim \mathcal{N}(0, 1)$ is a standard normal random variable then

$$\mathbf{P}\big(|Z| > t\big) = 2\,\mathbf{P}(Z > t) > \sqrt{\frac{2}{\pi}}\frac{t}{t^2+1}e^{-t^2} \ge \frac{1}{\sqrt{2\pi}}\frac{1}{t}e^{-t^2}$$

when $t \ge 1$. (Cook, 2009) Setting the right hand side to 2δ and solving gives

$$\mathbf{P}\big(|Z| > 2^{-1/2}\sqrt{W(\pi^{-1}\delta^{-2})}\big) > 2\delta$$

whenever $2^{-1/2}\sqrt{W(\pi^{-1}\delta^{-2})} \ge 1$ or $0 < \delta \le (e\sqrt{2\pi})^{-1}$. This is satisfied when $0 < \delta < \frac{1}{10}$. If we fix $m = \lfloor\frac{\mathrm{Var}(\boldsymbol{x}^*\boldsymbol{A}\boldsymbol{x})}{\mathrm{tr}(\boldsymbol{A})^2}\frac{1}{\epsilon^2}W(\frac{2/\pi}{\delta^2})\rfloor$ and write $\sigma^2 = \mathrm{Var}(\boldsymbol{x}^*\boldsymbol{A}\boldsymbol{x}) > 0$,

$$\begin{aligned}\mathbf{P}\big(|T_m - \mathrm{tr}\,\boldsymbol{A}| > \epsilon\,\mathrm{tr}\,\boldsymbol{A}\big) &= \mathbf{P}\left(\frac{\sqrt{m}}{\sigma}|T_m - \mathrm{tr}\,\boldsymbol{A}| > \frac{\sqrt{m}}{\sigma}\epsilon\,\mathrm{tr}\,\boldsymbol{A}\right)\\ &\ge \mathbf{P}\left(\frac{\sqrt{m}}{\sigma}|T_m - \mathrm{tr}\,\boldsymbol{A}| > \frac{1}{\sqrt{2}}\sqrt{W(\pi^{-1}\delta^{-2})}\right).\end{aligned}$$

Conveniently, the latter expression converges to $\mathbf{P}(|Z| \geq 2^{-1/2}\sqrt{W(\pi^{-1}\delta^{-2})})$ as $m \to \infty$, (Wasserman, 2013: Thm 5.10) and so

$$\lim_{\epsilon \to 0} \mathbf{P}\big(|T_m - \operatorname{tr} \boldsymbol{A}| > \epsilon \operatorname{tr} \boldsymbol{A}\big) > 2\delta.$$

This implies the existence of an $\epsilon_0 > 0$ so that $0 < \epsilon < \epsilon_0$ ensures $\mathbf{P}(|T_m - \operatorname{tr} \boldsymbol{A}| > \epsilon \operatorname{tr} \boldsymbol{A}) > \delta$ under our relation defining m. ■

2.4.1 Applications to Specific Estimators

For any input vector with sufficiently sub-Gaussian entries, Theorem 2.5 ensures that $m \geq \frac{12}{\epsilon^2(3-2\epsilon)} \log \frac{2}{\delta}$ matrix-vector products are needed to ensure that the corresponding trace estimate T_m satisfies ϵ-relative error with probability at least $1 - \delta$. Theorem 2.6 guarantees that this dependence on ϵ and δ is optimal for the class of Hutchinson-type estimators. (See Section 2.4.2 for extensions to more generic trace estimators.) By restricting ϵ to be smaller and smaller, the leading constant results in an asymptotic upper bound on m of $\frac{4}{\epsilon^2} \log \frac{2}{\delta}$. The following application of Theorem 2.6 ensures that this leading 4 is indeed optimal for all our example estimators; even though the 2 in the $\log \frac{2}{\delta}$ term could be closer to $\sqrt{2/\pi} \approx 0.79788$, this constant could be factored into a lower order term and hence we ignore it.

Corollary 2.1. There exists a rank-one positive semi-definite matrix $\boldsymbol{A} \in \mathbb{R}^{d \times d}$ and $\epsilon_0 > 0$ so that the Gaussian estimator G_m needs $m \geq \lfloor \frac{2}{\epsilon^2} W(\frac{2/\pi}{\delta^2}) \rfloor$ to achieve

$$\mathbf{P}\big(|G_m - \operatorname{tr} \boldsymbol{A}| > \epsilon \operatorname{tr} \boldsymbol{A}\big) < \delta$$

for any $0 < \epsilon < \epsilon_0$. If we know $\delta < 1/66$ this means that we need $m \geq \frac{4}{\epsilon^2} \log \frac{\sqrt{2/\pi}}{\delta} - \frac{2}{\epsilon^2} \log \log \frac{2/\pi}{\delta^2} - 1$.

Proof. Let Z be a standard normal random variable, $\boldsymbol{\mu}$ be any unit vector, and $\boldsymbol{A} = \boldsymbol{\mu}\boldsymbol{\mu}^*$. This result follows simply from applying Theorem 2.6 by realizing that

$$\operatorname{Var}(\boldsymbol{g}^* \boldsymbol{A} \boldsymbol{g}) = \operatorname{Var}((\boldsymbol{\mu}^* \boldsymbol{g})^2) = \operatorname{Var}(Z^2) = 2, \qquad \operatorname{tr} \boldsymbol{A} = \operatorname{tr}(\boldsymbol{\mu}^* \boldsymbol{\mu}) = \|\boldsymbol{\mu}\|_2^2 = 1,$$

and looking up the lower bound on the Lambert-W function from Hoorfar and Hassani (2007). ■

The proof for the classical Hutchinson estimator is a tad more complicated, but is easily generalizable to the other sparser vectors we saw applicable to Theorem 2.5.

Corollary 2.2. There exists a rank-one positive semi-definite matrix $\boldsymbol{A} \in \mathbb{R}^{d\times d}$ and $\epsilon_0 > 0$ so that the classical Hutchinson estimator H_m with Rademacher probing vectors needs $m \geq \lfloor \frac{2-\frac{2}{d}}{\epsilon^2} W(\frac{2/\pi}{\delta^2}) \rfloor$ to achieve

$$\mathbf{P}\big(|H_m - \operatorname{tr} \boldsymbol{A}| > \epsilon \operatorname{tr} \boldsymbol{A}\big) < \delta$$

for any $0 < \epsilon < \epsilon_0$. If we know $\delta < 1/66$ this means that we need $m \geq \frac{4-\frac{4}{d}}{\epsilon^2} \log \frac{\sqrt{2/\pi}}{\delta} - \frac{2-\frac{2}{d}}{\epsilon^2} \log\log \frac{2/\pi}{\delta^2} - 1$.

Proof. Let $\boldsymbol{\mu} = \mathbf{1}/\sqrt{d} \in \mathbb{R}^d$ be the constant unit vector and $\boldsymbol{A} = \boldsymbol{\mu}\boldsymbol{\mu}^*$. Then for the Rademacher vector $\boldsymbol{x} \sim \text{Uniform}\{\pm 1\}^d$

$$\boldsymbol{x}^*\boldsymbol{A}\boldsymbol{x} = \left(\sum_{i=1}^{n} \frac{\boldsymbol{x}_i}{\sqrt{d}}\right)^2 = \frac{1}{d}\sum_{ij} \boldsymbol{x}_i \boldsymbol{x}_j.$$

We already know $\mathbf{E}\, \boldsymbol{x}^*\boldsymbol{A}\boldsymbol{x} = 1$, but a simple counting argument can show us that

$$\begin{aligned}
\mathbf{E}(\boldsymbol{x}^*\boldsymbol{A}\boldsymbol{x})^2 &= \frac{1}{d^2}\sum_{ijk\ell} \mathbf{E}\, \boldsymbol{x}_i\boldsymbol{x}_j\boldsymbol{x}_k\boldsymbol{x}_\ell \\
&= \frac{1}{d^2}\left|\left\{i,j,k,\ell : \begin{smallmatrix} i=j\neq k=\ell \\ i=k\neq j=\ell \\ i=\ell\neq k=j \end{smallmatrix}\right\}\right| + \frac{\mathbf{E}\, \boldsymbol{x}_1^4}{d}\big|\{i,j,k,\ell : i=j=k=\ell\}\big| \\
&= 3 - \tfrac{3-\mathbf{E}\, \boldsymbol{x}_1^4}{d}
\end{aligned}$$

so

$$\operatorname{Var}(\boldsymbol{x}^*\boldsymbol{A}\boldsymbol{x}) = 3 - \frac{3-\mathbf{E}\, \boldsymbol{x}_1^4}{d} - 1 = 2 - \frac{3-\mathbf{E}\, \boldsymbol{x}_1^4}{d} = 2 - \frac{2}{d}.$$

Since $\operatorname{tr} \boldsymbol{A} = 1$, the result follows from applying Theorem 2.6 and the Lambert-W lower bound from Hoorfar and Hassani (2007). ∎

Essentially the same argument gives the tight lower bound on the constant for our sparse trace estimators as well.

Corollary 2.3. There exists a rank-one positive semi-definite matrix $\boldsymbol{A} \in \mathbb{R}^{d\times d}$ and $\epsilon_0 > 0$ so that the sparse Hutchinson estimator H_m with probing vectors taking values $\pm\sqrt{3}$ with probability $1/3$ and zero otherwise needs $m \geq \lfloor \frac{2}{\epsilon^2} W(\frac{2/\pi}{\delta^2}) \rfloor$ to achieve

$$\mathbf{P}\big(|H_m - \operatorname{tr} \boldsymbol{A}| > \epsilon \operatorname{tr} \boldsymbol{A}\big) < \delta$$

for any $0 < \epsilon < \epsilon_0$. If we know $\delta < 1/66$ this means that we need $m \geq \frac{4}{\epsilon^2} \log \frac{\sqrt{2/\pi}}{\delta} - \frac{2}{\epsilon^2} \log\log \frac{2/\pi}{\delta^2} - 1$.

Corollary 2.4. There exists a rank-one positive semi-definite matrix $\boldsymbol{A} \in \mathbb{R}^{d\times d}$ and $\epsilon_0 > 0$ so that the sparse Hutchinson estimator H_m with probing vectors taking values $\pm\sqrt{2}$ with probability $1/2$ and zero otherwise needs $m \geq \lfloor \frac{2-\frac{1}{d}}{\epsilon^2} W(\frac{2/\pi}{\delta^2}) \rfloor$ to achieve

$$\mathbf{P}\big(|H_m - \operatorname{tr} \boldsymbol{A}| > \epsilon \operatorname{tr} \boldsymbol{A}\big) < \delta$$

for any $0 < \epsilon < \epsilon_0$. If we know $\delta < 1/66$ this means that we need $m \geq \frac{4-\frac{2}{d}}{\epsilon^2} \log \frac{\sqrt{2/\pi}}{\delta} - \frac{2-\frac{1}{d}}{\epsilon^2} \log\log \frac{2/\pi}{\delta^2} - 1$.

2.4.2 Related Work: Generic Trace Estimation Lower Bounds

While we won't delve into this in detail, it's important to recognize that we haven't closed the book on trace estimation just yet. We have presented Hutchinson-type estimation schemes which are the best one could reasonably hope for *among this class of estimators*. There is a natural question of whether there exists some (potentially adaptive and terribly difficult to compute) procedure to estimate the trace of a matrix which only needs on the order of $\frac{1}{\epsilon} \log \frac{1}{\delta}$ or $\frac{1}{\epsilon^2} \log\log \frac{1}{\delta}$ queries of a matrix $\boldsymbol{A}$ in the form of matrix-vector products to compute an ϵ-relative error trace estimate with probability at least $1 - \delta$. Such a procedure would need to be randomized, as evidenced by Section 2.1, but our lower bounds from Section 2.4 don't eliminate this possibility.

It is open whether this is the case for the generic matrix-vector product oracle $O : \boldsymbol{x} \mapsto \boldsymbol{A}\boldsymbol{x}$ we have considered thus far, but Wimmer et al. (2014) showed that all estimators in Table 2.1 are optimal in their dependence on ϵ and δ among all estimation procedures which have access to a more restricted quadratic form oracle $O : \boldsymbol{x} \mapsto \boldsymbol{x}^*\boldsymbol{A}\boldsymbol{x}$. Their results are as follows.

Theorem 2.7 (Wimmer et al. Thm 1)**.** If we consider estimators for $\operatorname{tr} \boldsymbol{A}$ that pre-decide a distribution over queries $(\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_m)$ as well as weights $(w_1, w_2, \ldots, w_n)$ and output

$$T_m = \sum_{i=1}^{m} w_i O(\boldsymbol{r}_i) = \sum_{i=1}^{m} w_i \boldsymbol{r}_i^* \boldsymbol{A} \boldsymbol{r}_i,$$

the minimum variance estimator for which $\mathbf{E}\, T_m = \operatorname{tr} \boldsymbol{A}$ uniformly on $\boldsymbol{M}_n$ is achieved by sampling $\{\boldsymbol{r}_i\}$ as a collection of m orthogonal unit vectors and outputting

$$T_m^\star = \frac{n}{m} \sum_{i=1}^{m} O(\boldsymbol{r}_i)$$

Theorem 2.8 (Wimmer et al. Thm 2)**.** Any possibly nonlinear or adaptive estimator for the trace of a matrix $\boldsymbol{A}$ that sequentially submits random queries $\boldsymbol{r}_i$ to $O : \boldsymbol{r}_i \mapsto \boldsymbol{r}_i^* \boldsymbol{A} \boldsymbol{r}_i$ after seeing the previous $i - 1$ queries needs $\Omega(1/\epsilon)$ queries to achieve ϵ mean squared error uniformly across all matrices with Frobenius norm 1.

Theorem 2.9 (Wimmer et al. Thm 3)**.** Any possibly nonlinear or adaptive estimator for the trace of a matrix $\boldsymbol{A}$ that sequentially submits random queries $\boldsymbol{r}_i$ to $O : \boldsymbol{r}_i \mapsto \boldsymbol{r}_i^* \boldsymbol{A} \boldsymbol{r}_i$ after seeing the previous $i - 1$ queries needs $\Omega(\frac{1}{\epsilon^2} \log \frac{1}{\delta})$ queries to output an estimator T_m that satisfies

$$\mathbf{P}\big(|T - \operatorname{tr} \boldsymbol{A}| > \epsilon \operatorname{tr} \boldsymbol{A}\big) \leq \delta$$

for any rank-one positive definite matrix $\boldsymbol{A}$.

The proofs of these theorems are tedious and extending them to the generic matrix-vector product oracle will be left as future work. The main insight Wimmer et al. (2014) use is that the trace of a matrix $\boldsymbol{A}$ is always the same as the trace of a matrix $\boldsymbol{U}\boldsymbol{A}\boldsymbol{U}^*$ where $\boldsymbol{U}$ is orthogonal. As a result, they exchange the randomness-provided problem of estimating the trace of $\boldsymbol{A}$ to the statistical problem of estimating the trace of a random matrix $\boldsymbol{U}\boldsymbol{A}\boldsymbol{U}^*$ where $\boldsymbol{U}$ is sampled according to the Haar measure. The decades of work proving minimax lower bounds in the statistics community then becomes helpful and gives the desired results.

2.5 Experimental Results

Now we put the algorithms for estimating traces and Schatten norms to the test in a series of empirical trial. Tables 2.2, 2.3, and 2.4 compare all the trace estimators referenced in Table 2.1 for a couple different artificial matrices. Table 2.2 uses a random pentadiagonal matrix generated as an inner product $\boldsymbol{B}^*\boldsymbol{B}$ of a tridiagonal matrix $\boldsymbol{B}$ with independent standard normal entries; this matrix is the sparsest of all those evaluated against. Table 2.3 uses the all-ones matrix that was used as to create lower bounds for the estimators

Estimator	d	$\text{NNZ}(\boldsymbol{A})$	Expected Relative Error		
			$\epsilon = 0.5$	$\epsilon = 0.2$	$\epsilon = 0.1$
hutch	1,000	4,994	**0.0050**	**0.0024**	**0.0012**
gauss	1,000	4,994	0.0097	0.0043	0.0022
sphalf	1,000	4,994	0.0078	0.0033	0.0018
spthird	1,000	4,994	0.0093	0.0043	0.0023

Table 2.2 Empirical computation of expected relative error for various estimators computing the trace of $\boldsymbol{A}$ as a pentadiagonal matrix constructed as the product $\boldsymbol{A} = \boldsymbol{B}^*\boldsymbol{B}$ where $\boldsymbol{B}$ is a tridiagonal matrix with independent standard Gaussian entries. Here, we fix our desired failure probability $\delta = 0.5$ and vary $\epsilon \in \{0.5, 0.2, 0.1\}$. We use Theorem 2.5 to calculate a recommended number of samples to achieve the given error bound with at least $1 - \delta = 0.5$ probability, and report the empirical expected relative error in this case. Each element of this table is independently calculated using $n = 1,000$ trials. The estimators sphalf and spthird are the trace estimators from Table 2.1 which are half- and third-sparse in expectation, respectively.

in Section 2.4.1. Finally, Table 2.4 considers a rank-100 randomly generated matrix as an outer product of Gaussian matrices. For all of these experiments we fix the desired failure probability to $\delta = 0.5$ and consider the expected relative error as we vary ϵ from 0.5 to 0.2 and then to 0.1. For more details see the description of Table 2.2.

This bound ensures that the *median* relative error is less than ϵ. These experiments validate whether the distribution of relative errors from these estimators is skewed towards zero, indicating a good estimate or conservative bound, or whether the error is skewed past ϵ, indicating a bad estimator to use in practice. Indeed, the latter case would show that failure of the estimator (which can occur half of the time here) would result in severely poor computational estimate of the trace. The former would say that any failure of these estimators would seldom return an estimate that was *grossly* inaccurate.

Luckily, our computational evidence suggests across the board that when using the parameter guidance of Theorem 2.5 in the artificial situations considered, all sub-Gaussian trace estimators won't return wildly inaccurate estimates when they fail. The empirical estimated relative error was never more than 50% of our bound on the median of the relative error. This tightness is achieved in Table 2.3, by the same matrix we used to construct

Estimator	d	$\text{NNZ}(\boldsymbol{A})$	Expected Relative Error		
			$\epsilon = 0.5$	$\epsilon = 0.2$	$\epsilon = 0.1$
hutch	1,000	1,000,000	**0.1897**	0.0902	0.0454
gauss	1,000	1,000,000	0.1911	0.0887	0.0460
sphalf	1,000	1,000,000	0.1956	0.0897	0.0453
spthird	1,000	1,000,000	0.1961	**0.0839**	**0.0450**

Table 2.3 Empirical computation of expected relative error for various estimators computing the trace of $\boldsymbol{A} = \mathbf{1}\mathbf{1}^*$. Remaining setup is the same as Table 2.2.

Estimator	d	$\text{NNZ}(\boldsymbol{A})$	Expected Relative Error		
			$\epsilon = 0.5$	$\epsilon = 0.2$	$\epsilon = 0.1$
hutch	1,000	1,000,000	**0.0193**	**0.0089**	0.0049
gauss	1,000	1,000,000	0.0220	0.0094	0.0049
sphalf	1,000	1,000,000	0.0203	0.0091	**0.0048**
spthird	1,000	1,000,000	0.0205	0.0096	**0.0048**

Table 2.4 Empirical computation of expected relative error for various estimators computing the trace of $\boldsymbol{A} = \boldsymbol{B}\boldsymbol{B}^*$ where $\boldsymbol{B} \in \mathbb{R}^{d\times 100}$ has independent standard Gaussian entries. Remaining setup is the same as Table 2.2.

the lower bounds in Section 2.4.1.

From the proof of Theorem 2.5, it is natural to ask whether specific input distributions result in better computational results on average across certain types of matrices. Indeed, by reducing to the Gaussian case, we would expect query distributions using Rademacher entries with *much* smaller higher moments to give tighter concentration in practice. It turns out that this intuition is evident throughout our results; the Hutchinson estimator of the trace performed the best in six of the nine artificial experiments. Most of the time, performance ranking was dictated precisely by how fast the moments of the query distribution grows: Rademacher entries have the smallest moments, then the half-sparse $\{-\sqrt{2}, 0, \sqrt{2}\}$ distribution, then the third-sparse $\{-\sqrt{3}, 0, \sqrt{3}\}$, before we reach the bounding Gaussian case. What's more, the Hutchinson estimator performs best for sparse matrices, while the Gaussian estimator tends to perform best for the dense, higher rank matrices of Table 2.4. These differences are seen with the more nuanced bounds

Matrix	d	$\text{NNZ}(\boldsymbol{A})$	Expected $\text{tr}\,\boldsymbol{A}$ Error			
			hutch	gauss	sphalf	spthird
apache1	80,800	542,184	**0.0004**	0.0007	0.0005	0.0006
cvxbqp1	50,000	349,968	**0.0004**	0.0006	0.0005	0.0006
gridgena	48,962	512,084	**0.0002**	0.0005	0.0004	0.0005
jnlbrng1	40,000	199,200	**0.0003**	0.0006	0.0004	0.0006
minsurfo	40,806	203,622	**0.0003**	0.0006	0.0005	0.0006
obstclae	40,000	197,608	**0.0002**	0.0005	0.0004	0.0005
torsion1	40,000	197,608	**0.0002**	0.0005	0.0003	0.0004
wathen100	30,401	471,601	**0.0005**	0.0008	0.0007	0.0008
wathen120	36,441	565,761	**0.0005**	0.0008	0.0007	0.0008

Table 2.5 Empirical computation of expected relative error for various trace estimators. We use all matrices from the GHS_psdef group of the UF Sparse Matrix Collection (see Davis and Hu (2011)) which have less than a million nonzero entries and are strictly positive definite. We fix our desired failure probability $\delta = 0.5$ and $\epsilon = 0.2$, using Theorem 2.5 to calculate a recommended number of samples to achieve the given error bound with at least $1 - \delta = 0.5$ probability. We report the empirical expected relative error in this case, computed over $n = 150$ independent trials for each element of the table.

explored in Roosta-Khorasani and Ascher (2015) in Theorem 2 and Theorem 3, where the diagonal dominance can theoretically improve Hutchinson estimator convergence while high stable rank improves convergence for the Gaussian estimator.

To ensure that these results for artificial matrices are maintained when we apply these algorithms to real world data, we also tested the four estimators against a collection of relatively large, sparse matrices from the UF Sparse Matrix Collection (see Davis and Hu (2011)). These positive definite matrices are collected from real world examples of finite element modeling. See Table 2.5 for details. For computational simplicity, we set $\delta = 0.5$ as before but left $\epsilon = 0.2$ uniformly. As we could suspect from our artificial results above, we can see that the Hutchinson estimator performs uniformly better than the other matrices, probably due to the sparsity in this collection. Interestingly, though, all four estimators return extremely strong empirical relative error – all below 1/1000 even though the desired relative error bound gives a median relative error 200 times larger than this.

To sanity-check the performance of the almost-equivalent Algorithm 1

Matrix	d	$\text{NNZ}(\boldsymbol{A})$	Expected $\|\boldsymbol{A}\|_2^2$ Error			
			hutch	gauss	sphalf	spthird
apache1	80,800	542,184	**0.0005**	0.0009	0.0007	0.0008
cvxbqp1	50,000	349,968	**0.0007**	0.0009	**0.0007**	0.0009
gridgena	48,962	512,084	**0.0003**	0.0006	0.0004	0.0005
jnlbrng1	40,000	199,200	**0.0005**	0.0007	**0.0005**	0.0007
minsurfo	40,806	203,622	**0.0006**	0.0008	0.0007	0.0008
obstclae	40,000	197,608	**0.0003**	0.0006	0.0005	0.0006
torsion1	40,000	197,608	**0.0004**	0.0005	**0.0004**	0.0005
wathen100	30,401	471,601	**0.0011**	0.0014	**0.0011**	0.0013
wathen120	36,441	565,761	**0.0011**	0.0012	0.0012	0.0012

Table 2.6 Empirical computation of expected relative error for various estimators computing the Schatten-2 norm by Algorithm 1. Remaining setup is the same as Table 2.5, though the relative error reported is in terms of the *squared* Schatten-2 norm instead of the actual norm, as reflected in Algorithm 1.

for Schatten-p norm computation, we test that algorithm against the same UF Sparse Matrix Collection group for computing the Schatten-2 norm. That prognosis is exceptional as well, with all estimators furnishing empirical relative error in the squared Schatten-2 norm that is at most 2/1000. The Hutchinson estimator again performs uniformly the best, though this time the half-sparse queries meet the same empirical performance for about half of the matrices. This might be because the matrix we are estimating the trace of, $\boldsymbol{A}^2$, is potentially much less sparse than $\boldsymbol{A}$ and as a result the associated benefits of the Hutchinson estimator may be lost.

Chapter 3

Results on Preconditioner Selection

Direct algorithms like Gaussian elimination are reliable and standard when one is trying to solve a generic system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ to machine precision (Golub and Van Loan, 2012: Ch. 3). When the system becomes large and sparse, or if one is willing to accept an approximate solution, iterative methods like the conjugate gradients algorithm can become attractive. With iterative methods, our algorithm constructs a sequence of vectors $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$ which converge (hopefully, quickly) to the vector $\boldsymbol{x}$. For example (Trefethen and Bau III, 1997: Thm. 38.5), the conjugate gradients algorithm works for systems with positive definite $\boldsymbol{A}$ and can produce a vector $\boldsymbol{x}_t$ after t iterations with

$$\|\boldsymbol{x}_t - \boldsymbol{x}\|_{\boldsymbol{A}} \leq 2\left(\frac{\sqrt{\kappa(\boldsymbol{A})} - 1}{\sqrt{\kappa(\boldsymbol{A})} + 1}\right)^t \|\boldsymbol{x}\|_{\boldsymbol{A}}.$$

Here, $\|\boldsymbol{z}\|_{\boldsymbol{A}} = \sqrt{\boldsymbol{z}^*\boldsymbol{A}\boldsymbol{z}}$ is the norm induced by $\boldsymbol{A}$, and $\kappa(\boldsymbol{A}) = \frac{\lambda_{\max}(\boldsymbol{A})}{\lambda_{\min}(\boldsymbol{A})} \geq 1$ is the condition number of $\boldsymbol{A}$. The number of iterations to achieve some fixed accuracy is then $O(\sqrt{\kappa(\boldsymbol{A})})$ and each iteration takes time about equal to the time it takes to compute $\boldsymbol{A}\boldsymbol{z}$ for a vector $\boldsymbol{z}$.

Even though the conjugate gradients method can appear attractive at first, many applications have poor conditioning which makes the method useless on a practical level. To resolve this issue, one can construct a cheap approximation $\boldsymbol{M}$ to the matrix $\boldsymbol{A}$, called a preconditioner, and instead solve the system $\boldsymbol{M}^{-1}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{M}^{-1}\boldsymbol{b}$. If $\boldsymbol{M}$ is a faithful approximation, $\boldsymbol{M}^{-1}\boldsymbol{A}$ should be close to the identity and hence have good conditioning. In the context of conjugate gradients, the equivalent system is in reality

$(\boldsymbol{M}^{-1/2}\boldsymbol{A}\boldsymbol{M}^{-1/2})(\boldsymbol{M}^{1/2}\boldsymbol{x}) = \boldsymbol{M}^{-1/2}\boldsymbol{b}$, but the same intuition carries through. This technique is heavily used, for example resulting in the celebrated nearly-linear time solvers for Laplacian systems (see Vishnoi et al. (2013).) See (Golub and Van Loan, 2012: Sec. 11.5) for more background on preconditioning.

A preconditioner $\boldsymbol{M}$ is helpful if it reduces the number of conjugate gradients iterations enough to offset the cost of constructing the preconditioner plus the additional cost of taking the iterations (effectively an extra computation of the form $\boldsymbol{M}^{-1}\boldsymbol{z}$ per iteration.) That framing makes preconditioner selection seem straightforward, but in reality finding suitable preconditioners is a challenging problem and an open research area as seen in Benzi (2002). For instance, even if we have a couple candidate preconditioners $\boldsymbol{M}_1, \boldsymbol{M}_2, \ldots, \boldsymbol{M}_n$ for our problem ready to use and assume that they add the same amount of time to compute each iteration, it is unclear how one would go about estimating which preconditioner would reduce the iteration count the most without actually solving a system with each preconditioner or doing a comparable amount of work. This task is the focus of the present work.

Prior Art

Current methods used for forecasting preconditioner quality are not robust across all situations of interest and as a result cannot be used in an automated manner. The simplest criterion is that a preconditioner $\boldsymbol{M}$ ought to be an 'accurate' approximation, in the sense that $\|\boldsymbol{M} - \boldsymbol{A}\|_{\mathsf{F}}$ is small. It turns out that for symmetric M-matrices, this accuracy criterion is a useful proxy for the number of conjugate gradient iterations necessary to solve the preconditioned system in $\boldsymbol{A}$. This point was theoretically confirmed by Axelsson and Eijkhout (1990), who noticed that the condition number $\kappa(\boldsymbol{M}^{-1}\boldsymbol{A})$ can be bounded in terms of $\|\boldsymbol{M}^{-1}\|_{\mathsf{F}}\|\boldsymbol{M} - \boldsymbol{A}\|_{\mathsf{F}}$. The accuracy criterion was heavily tested on an empirical level in Duff and Meurant (1989).

Even in this setting, though, there exist accurate real-world preconditioners that give a poor iteration count because $\|\boldsymbol{M}^{-1}\|_{\mathsf{F}}$ is very large (Benzi et al., 1999). Since $\|\boldsymbol{M}^{-1}\|_{\mathsf{F}}$ presumably[1] requires computing $\boldsymbol{M}^{-1}$ even though in general we only have access to $\boldsymbol{M}^{-1}$ via matrix-vector products, this was deemed impractical (Benzi, 2002). To detect this so-called instability in $\boldsymbol{M}^{-1}$, Chow and Saad (Chow and Saad, 1997) proposed estimating the ℓ^∞ operator norm of $\boldsymbol{M}^{-1}$ as $\|\boldsymbol{M}^{-1}\boldsymbol{e}\|_\infty$, where $\boldsymbol{e}$ is the vector of all-ones. For incomplete

[1] One can rephrase Algorithm 3 to create a practical algorithm for computing $\|\boldsymbol{M}^{-1}\|_{\mathsf{F}}$.

LU factorization preconditioners, if this is is large relative to the size of the smallest pivot, it can predict instability in the conjugate gradients method.

The quantity known as 'preconditioner stability,' $\|\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A}\|_{\mathsf{F}}$, is in general the most reliable indicator of preconditioner performance. This is especially true among many non-symmetric problems or problems which are far from diagonally dominant (Benzi et al., 1999). Unfortunately, prior work has suggested that computing preconditioner stability is 'impractical' (Benzi, 2002) for effectively the same reason as why $\|\boldsymbol{M}^{-1}\|_{\mathsf{F}}$ was deemed impractical to compute.

Contributions

The core contribution of this chapter is the realization that randomized sketching methods make completely practical the computation of a quantity previously thought to be infeasible to compute. In addition to this primary method for computing preconditioner stability, we have provided a number of other results which are also deserving of note:

- We prove that no practical deterministic algorithm, in a meaningful sense, could possibly be used to estimate preconditioner stability.
- We provide an algorithm which can provably select a preconditioner of approximately minimal stability among n candidate preconditioners using computational resources equivalent to computing about $n \log n$ steps of the conjugate gradients algorithm.
- By making an anti-concentration assumption about the candidate preconditioners, we are able to provide a theoretical speedup to the initial preconditioner selection method which largely decouples the runtime dependence between the number of preconditioners n and the desired accuracy.
- Using our initial preconditioner selection algorithm, we create the first (to the best of our knowledge) method for preconditioning in kernel regression which never gives a worse number of iterations than using no preconditioner in standard tests.

It is important to point out that while our motivation for this method and experiments consider positive definite systems and preconditioners, our methods work equally well with arbitrary matrices $\boldsymbol{A}$ and preconditioners $\boldsymbol{M}$.

Overview

Section 3.1 motivates the need for a randomized algorithm for stability estimation with theory, responds with a sketching-based solution, and uses it to create and analyze a method for preconditioner selection. This theory is empirically confirmed in Section 3.2 where we apply the primary preconditioner selection algorithm from Section 3.1 to solving generic real-world systems (Section 3.2.1) and creating more robust preconditioning methods for kernel regression (Section 3.2.2.)

3.1 The Algorithm

This section forms the meat of this chapter. In Section 3.1.1 we show that the only algorithms which can possibly estimate preconditioner stability must be randomized. The natural follow-up question of whether randomization can indeed work is answered in the affirmative in Section 3.1.2, where we show that a slight adaptation of a well-known sketching-based algorithm for computing Schatten norms perfectly fits our realistic access model to our preconditioner $\boldsymbol{M}$ and matrix $\boldsymbol{A}$. Once we have a good estimator of preconditioner stability, a natural method for selecting the candidate preconditioner with minimal stability criterion presents itself in Section 3.1.3. It turns our that our algorithm can be trivially parallelized, and a testament to this fact is given in Section 3.1.3. In Section 3.1.4 we take advantage of highly informative results from the literature on trace estimation to provide useful approximation guarantees and runtime bounds for the previously presented algorithms. Using these bounds, we include a theoretical speedup on our preconditioner selection algorithm which helps when there is a somewhat clear winner in Section 3.1.4. Section 3.1.4 wraps up our conversation in this area by proving that the bounds included in our preconditioner estimation and initial preconditioner selection algorithms are tight even to their leading order constants. We also prove that no randomized algorithm for estimating preconditioner stability could possibly do better asymptotically by relying on similar results from the trace estimation literature.

3.1.1 Randomization is Necessary to Compute Preconditioner Stability

This paper provides a simple randomized algorithm which can accurately estimate the preconditioner stability $\|\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A}\|_{\mathsf{F}}$ in time faster than running

a constant number of iterations of preconditioned conjugate gradients with the matrix $\boldsymbol{A}$ and preconditioner $\boldsymbol{M}$. Through incorporating randomness, however, we must accept that the algorithm fails with some probability. This failure probability can be made arbitrarily small, but it would still be advantageous (for example, in mission-critical applications) to provide a deterministic algorithm for the same task, so long as it attained the same approximation guarantees. The purpose of this section is to crush that latter hope, and the following theorem does just that. Note that the proof is analogous to that of Theorem 2.1 in Chapter 2.

Theorem 3.1. Fix some $0 \leq \epsilon < 1$. Suppose we have a deterministic algorithm $\textsc{Alg}(\boldsymbol{A}, \boldsymbol{M})$ which takes as input an arbitrary positive semi-definite matrix $\boldsymbol{A} \in \mathbb{F}^{d \times d}$ and positive definite matrix $\boldsymbol{M} \in \mathbb{F}^{d \times d}$, and returns an estimate

$$(1-\epsilon)\|\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A}\|_{\mathsf{F}} \leq \textsc{Alg}(\boldsymbol{A}, \boldsymbol{M}) \leq (1+\epsilon)\|\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A}\|_{\mathsf{F}}$$

after sequentially querying and observing matrix vector multiplies of the form $(\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A})\boldsymbol{q}_i = \boldsymbol{q}_i - \boldsymbol{M}^{-1}(\boldsymbol{A}\boldsymbol{q}_i)$ for $i = 1, 2, \ldots, k$ where k is a universal constant depending only on d and ϵ. Then $k \geq d$.

Proof. Take $\boldsymbol{M} = \boldsymbol{I}$ for the remainder of the proof. Suppose to the contrary that $k = d - 1$ suffices to compute $\textsc{Alg}(\boldsymbol{A}, \boldsymbol{M})$. Let $\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_{d-1}$ be the query vectors used by the algorithm in the case that $(\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A})\boldsymbol{q}_i$ always returns $\boldsymbol{0}$. Write $\boldsymbol{P}$ for the orthogonal projection onto $\operatorname{span}\{\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_{d-1}\}$. Both of the positive semi-definite matrices $\boldsymbol{A} = \boldsymbol{I}$ and $\boldsymbol{A} = \boldsymbol{P}$ will return $(\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{I})\boldsymbol{q}_i = (\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{P})\boldsymbol{q}_i = \boldsymbol{0}$ uniformly over $i = 1, 2, \ldots, d-1$, and thus since the algorithm is deterministic the estimated stabilities $\textsc{Alg}(\boldsymbol{I}, \boldsymbol{M}) = \textsc{Alg}(\boldsymbol{P}, \boldsymbol{M})$ are equal. But $\boldsymbol{P} \neq \boldsymbol{I}$ since $\boldsymbol{P}$ was an orthogonal projection onto a subspace of dimension strictly less than d, and hence

$$0 < (1-\epsilon)\|\boldsymbol{I} - \boldsymbol{P}\|_{\mathsf{F}} \leq \textsc{Alg}(\boldsymbol{P}, \boldsymbol{M}) = \textsc{Alg}(\boldsymbol{I}, \boldsymbol{M}) \leq (1+\epsilon)\|\boldsymbol{I} - \boldsymbol{I}\|_{\mathsf{F}} = 0$$

by our approximation guarantee. This contradiction ensures that we must take $k \geq d$. ∎

Of course, using $k = d$ queries suffices to achieve no error at all, and so the above lower bound is tight:

$$\|\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A}\|_{\mathsf{F}} = \left(\sum_{i=1}^{d} \|(\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A})\boldsymbol{e}_i\|_2^2\right)^{1/2} \tag{3.1}$$

where $\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_d$ is any orthonormal basis for $\mathbb{F}^d$. Also, note that the condition that $\boldsymbol{A}$ and $\boldsymbol{M}$ be positive semi-definite gives a stronger result than if they were allowed to be arbitrary matrices.

In order to put Theorem 3.1 into better context, though, recall that the dominant cost of an iteration of preconditioned conjugate gradients (Golub and Van Loan, 2012: Alg. 11.5.1) is (a) computing $\boldsymbol{A}\boldsymbol{y}$ for a vector $\boldsymbol{y}$, and (b) computing $\boldsymbol{M}^{-1}\boldsymbol{z}$ for a vector $\boldsymbol{z}$. To leading order, this is the same number of floating point operations as computing $(\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A})\boldsymbol{q}$ via $\boldsymbol{q} - \boldsymbol{M}^{-1}(\boldsymbol{A}\boldsymbol{q})$ for a vector $\boldsymbol{q}$, and so Theorem 3.1 says roughly that in the time it takes to even approximate $\|\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A}\|_{\mathsf{F}}$ deterministically, one can solve a system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ exactly (at least in exact arithmetic) by running the conjugate gradients algorithm for d iterations. Since our whole goal of computing $\|\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A}\|_{\mathsf{F}}$ is to forecast how well $\boldsymbol{M}$ would do as a preconditioner for solving the system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, this means that any deterministic algorithm for this task is effectively useless. To tie loose ends, we conclude by noting that matrix-vector product access to $\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A}$ is indeed a reasonable computational model to prove lower bounds on algorithm performance because preconditioners $\boldsymbol{M}$ are in some sense defined by the fact we only have access to $\boldsymbol{M}^{-1}$ via matrix-vector products. Similarly, the conjugate gradients method would only be practical because computing matrix vector multiplies $\boldsymbol{A}\boldsymbol{y}$ are practical.

The above qualitative corollary gives a strong theoretical backing to the common refrain that preconditioner stability is impractical to compute (Benzi, 2002: Sec. 3.2.2). Moreover, it shows that the only possible schemes for computing preconditioner stability in a practical manner must be randomized. Such a reasonable randomized method is presented in the next section.

3.1.2 Computing Preconditioner Stability via Randomization

Now we will show that, unlike the deterministic case, randomization makes it entirely practical to compute preconditioner stability. To see why this is intuitive, let $\boldsymbol{q} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ be a standard Gaussian vector. Then

$$\begin{aligned}
\|\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A}\|_{\mathsf{F}}^2 &= \operatorname{tr}\big((\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A})^*(\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A})\big) && (3.2)\\
&= \operatorname{tr}\big((\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A})^*(\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A})\,\mathbf{E}\,\boldsymbol{q}\boldsymbol{q}^*\big) && (3.3)\\
&= \mathbf{E}\operatorname{tr}\big(\boldsymbol{q}^*(\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A})^*(\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A})\boldsymbol{q}\big) && (3.4)\\
&= \mathbf{E}\,\|(\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A})\boldsymbol{q}\|_2^2 && (3.5)
\end{aligned}$$

Algorithm 3: Stab($\boldsymbol{A}, \boldsymbol{M}, k$): Estimates the stability of the preconditioner $\boldsymbol{M}$ in $\sim 3dk + kT_m + 2k\,\text{NNZ}(\boldsymbol{A})$ floating point operations when $\mathbb{F} = \mathbb{R}$ or $\sim 6dk + kT_m + 4\,\text{NNZ}(\boldsymbol{A})$ flops when $\mathbb{F} = \mathbb{C}$, where T_m is the number of flops needed to compute $\boldsymbol{M}^{-1}\boldsymbol{b}$ for an arbitrary $\boldsymbol{b} \in \mathbb{F}^d$.

Data: A matrix $\boldsymbol{A} \in \mathbb{F}^{d\times d}$, preconditioner $\boldsymbol{M} \in \mathbb{F}^{d\times d}$, and an accuracy parameter $k \in \{1, 2, \dots\}$.
Result: A estimate of the preconditioner stability $\|\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A}\|_{\mathsf{F}}$.
Form a matrix $\boldsymbol{Q} = [\boldsymbol{q}_1, \dots, \boldsymbol{q}_k]$ with independent columns $\boldsymbol{q}_i \sim \mathcal{N}(\boldsymbol{0}, \frac{1}{k}\boldsymbol{I}_d)$.
Construct the sketch $\boldsymbol{S} = (\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A})\boldsymbol{Q}$ via its columns $\boldsymbol{q}_i - \boldsymbol{M}^{-1}(\boldsymbol{A}\boldsymbol{q}_i)$.
Return $\|\boldsymbol{S}\|_{\mathsf{F}}$.

by the linearity of expectation, the cyclic property of the trace, and the fact that $\mathbf{E}\,\boldsymbol{q}\boldsymbol{q}^* = \boldsymbol{I}$. Thus, if $\boldsymbol{q}_i$ are independent standard normal vectors for all $i = 1, 2, \dots, k$, the Monte-Carlo squared stability estimate

$$S^2 = \frac{1}{k}\sum_{i=1}^{k} \|(\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A})\boldsymbol{q}_i\|_2^2 \to \|\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A}\|_{\mathsf{F}}^2 \tag{3.6}$$

almost surely as $k \to \infty$ by the strong law of large numbers. We can rewrite the above estimator as

$$S^2 = \sum_{i=1}^{k} \|(\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A})\tfrac{\boldsymbol{q}_i}{\sqrt{k}}\|_2^2 = \|(\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A})\boldsymbol{Q}\|_{\mathsf{F}}^2 \tag{3.7}$$

where $\boldsymbol{Q}$ is a matrix with independent and identically distributed elements $\boldsymbol{Q}_{ii} \sim \mathcal{N}(0, \frac{1}{k})$. This stability estimation algorithm for $S = \sqrt{S^2} \approx \|\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A}\|_{\mathsf{F}}$ is given as Algorithm 3.

It is important to note that the foundations of the above algorithm are not novel. It is mathematically equivalent to applying the trace estimators in (Roosta-Khorasani and Ascher, 2015) to the matrix $(\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A})^*(\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A})$ and then taking the square root. It is also a simplification of the Schatten-2 norm estimator in (Woodruff, 2014: Thm. 69) (relayed from (Li et al., 2014)) applied to $\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A}$. The reason we include Algorithm 3 is not because of its mathematical novelty but because of its observational novelty: sketching algorithms using the matrix-vector multiply access model are a perfect fit for

interrogating the matrices $\boldsymbol{M}^{-1}$ and $\boldsymbol{A}$ in the context of conjugate gradients, since this kind of access to $\boldsymbol{M}^{-1}$ and $\boldsymbol{A}$ are precisely what make the conjugate gradients algorithm practical.

Of course, the presentation thus far does not help us choose how large the accuracy parameter k should be. If the variance of the estimate S is large, for example, we would reasonably expect k to necessarily be very large to get a decent stability estimate. By construction, though, the standard deviation

$$\sigma(S) = \sqrt{\operatorname{Var}(S^2)} \leq \sqrt{\mathbf{E}\, S^2} = \frac{1}{\sqrt{k}} \|\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A}\|_{\mathsf{F}}, \tag{3.8}$$

and so as long as $\mathbf{E}\, S$ is quite close to $\sqrt{\mathbf{E}\, S^2} = \|\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A}\|_{\mathsf{F}}$ then $k = O(\frac{1}{\epsilon^2})$ will ensure that the algorithm returns an estimate for $\|\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A}\|_{\mathsf{F}}$ within a multiplicative factor $1 \pm \epsilon$ by Chebyshev's inequality (Vershynin, 2018: Cor. 1.25). A rigorous bound will be shown in Section 3.1.4, tightening an analogous result presented in (Roosta-Khorasani and Ascher, 2015) in the constant factor.

3.1.3 Randomized Algorithm for Selecting the 'Best' Preconditioner

Ignoring the issue of selecting k, once we have a practical way to compute preconditioner stability, a trivial algorithm for picking the preconditioner among n candidates $\boldsymbol{M}_1, \boldsymbol{M}_2, \ldots, \boldsymbol{M}_n$ presents itself. Namely, we can compute estimates $S_i \approx \|\boldsymbol{I} - \boldsymbol{M}_i^{-1}\boldsymbol{A}\|_{\mathsf{F}}$ for $i = 1, 2, \ldots, n$ and then just return the preconditioner $\boldsymbol{M}_i$ for which S_i is minimized. This is presented as Algorithm 4. As we mentioned in the previous section, theoretical advice on how to pick k will be given in Section 3.1.4. An improvement to this algorithm in the case there is a clear winner, relying on those analytical bounds, is included in Section 3.1.4.

We note that the sketching matrix $\boldsymbol{Q}$ can be re-used when computing the stability estimates S_j in Algorithm 3. This is done in all our computational experiments in Section 3.2, and reduces the number of normal variables one needs to simulate from n^2k to nk. Reuse does not affect our theoretical upper bound presented in Section 3.1.4.

Parallelization

A convenient aspect of sketching based algorithms like Algorithm 4 is that they can be parallelized extremely easily. For instance, suppose we are

Algorithm 4: Returns an approximately optimal preconditioner in the Tyrtyshkinov sense among n candidates with strictly fewer floating point operations than running k iterations of preconditioned conjugate gradients (Golub and Van Loan, 2012: Alg. 11.5.1) with each of the n preconditioners.

Data: A matrix $\boldsymbol{A} \in \mathbb{F}^{d\times d}$, n candidate preconditioners $\boldsymbol{M}_1$, $\boldsymbol{M}_2$,...,$\boldsymbol{M}_n \in \mathbb{F}^{d\times d}$, and an accuracy parameter $k \in \{1, 2, \ldots\}$.
Result: A preconditioner $\boldsymbol{M}_i$ which approximately minimizes the stability criterion $\|\boldsymbol{I} - \boldsymbol{M}_j^{-1}\boldsymbol{A}\|_{\mathsf{F}}$ over $j = 1, \ldots, n$.
Compute a stability estimate $S_j = \mathrm{Stab}(\boldsymbol{A}, \boldsymbol{M}_j, k)$ for each $j = 1, 2, \ldots, n$.
Return an arbitrary $\boldsymbol{M}_i$ with $S_i = \min_{1\le j\le n} S_j$.

trying to pick the Tyrtyshkinov optimal preconditioner among n candidates $\boldsymbol{M}_1, \boldsymbol{M}_2, \ldots, \boldsymbol{M}_n$, and have n processors P_j, $j = 1, 2, \ldots, n$, which have access to $\boldsymbol{A}$ and $\boldsymbol{M}_j$, respectively. Then we can compute each stability estimate S_j in parallel at processor P_j. Ignoring communication costs (which are a genuine concern in practice,) this would bring the runtime of the algorithm down to computing k steps of preconditioned conjugate gradients with $\boldsymbol{A}$ and the most computation-intensive (in terms of matrix-vector multiply access to $\boldsymbol{M}^{-1}$) preconditioner $\boldsymbol{M}_j$.

Taken to the extreme, one could similarly parallelize Algorithm 4 over nk processors P_{ij}, $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, k$, assuming each P_{ij} had access to $\boldsymbol{A}$ and the candidate preconditioner $\boldsymbol{M}_j$. Each processor P_{ij} would need to compute and return $s_{ij} = \|(\boldsymbol{I} - \boldsymbol{M}_j^{-1}\boldsymbol{A})\boldsymbol{q}_i\|_2^2$ where $\boldsymbol{q}_i$ is an independently sampled standard normal vector. Then in parallel for all $i = 1, 2, \ldots, n$ processor P_{i1} could compute $S_i^2 = \frac{1}{k}\sum_{j=1}^{k} s_{ij}$, at which point we could use processor P_{11} to compute i such that S_i^2 is minimal and return the corresponding $\boldsymbol{M}_i$. Ignoring communication costs again, this algorithm take fewer floating point operations than running *one* iteration of preconditioned conjugate gradients with $\boldsymbol{A}$ and the most computation-intensive preconditioner $\boldsymbol{M}_j$, plus k flops that were used for computing the S_i.

3.1.4 Approximation Guarantees and Runtime Bounds

In the above exposition, we have largely ignored the choice of the accuracy parameter k. In this section we will fill that gap in knowledge. To start, we will relay the following theorem, which says that to estimate preconditioner stability up to a $1 \pm \epsilon$ multiplicative factor with failure probability at most δ, one may take $k = O(\frac{1}{\epsilon^2} \log \frac{1}{\delta})$ in Algorithm 3, which is (in contrast to the deterministic case) completely independent of the underlying dimension.

Theorem 3.2. Let $\boldsymbol{M}$ and $\boldsymbol{A}$ be arbitrary matrices in $\mathbb{F}^{d \times d}$ where $\boldsymbol{M}$ is invertible. If ϵ and δ are positive and less than one, taking $k \geq \frac{12}{\epsilon^2(3-2\epsilon)} \log \frac{2}{\delta}$ ensures that the estimate $\mathrm{Stab}(\boldsymbol{A}, \boldsymbol{M}, k)$ satisfies

$$\sqrt{1-\epsilon}\, \|\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A}\|_{\mathsf{F}} \leq \mathrm{Stab}(\boldsymbol{A}, \boldsymbol{M}, k) \leq \sqrt{1+\epsilon}\, \|\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A}\|_{\mathsf{F}}.$$

with probability at least $1 - \delta$. In particular, if $\epsilon \leq 1/2$, then the simpler condition $k \geq \frac{6}{\epsilon^2} \log \frac{1}{\delta}$ ensures this same approximation guarantee.

Proof. Follows directly from Theorem 2.4 in Chapter 2. ■

Using Theorem 3.2 we are able to prove an approximation guarantee for Algorithm 4 via a union bound. In particular, to achieve an ϵ-multiplicative approximation to the best of n candidate preconditioners with probability at least $1 - \delta$ we can take $k = O(\frac{1}{\epsilon^2} \log \frac{n}{\delta})$. This dependence on n is quite weak and since in realistic applications we would only expect to have at most, say, twenty candidate preconditioners the necessary k is effectively constant, again independent of the underlying dimensionality.

Theorem 3.3. Let $\boldsymbol{A} \in \mathbb{F}^{d \times d}$ be an arbitrary matrix, and $\boldsymbol{M}_1, \boldsymbol{M}_2, \ldots, \boldsymbol{M}_n \in \mathbb{F}^{d \times d}$ be invertible candidate preconditioners for $\boldsymbol{A}$. If ϵ and δ are positive and less than one, taking $k \geq \frac{12}{\epsilon^2(3-2\epsilon)} \log \frac{2n}{\delta}$ ensures that the preconditioner $\boldsymbol{M}_i$ returned by Algorithm 4 satisfies

$$\|\boldsymbol{I} - \boldsymbol{M}_i^{-1}\boldsymbol{A}\|_{\mathsf{F}} \leq \sqrt{\frac{1+\epsilon}{1-\epsilon}} \min_{1 \leq j \leq n} \|\boldsymbol{I} - \boldsymbol{M}_j^{-1}\boldsymbol{A}\|_{\mathsf{F}}.$$

In particular, if $\epsilon < 1/2$ the simpler condition $k \geq \frac{11}{\epsilon^2} \log \frac{2n}{\delta}$ ensures

$$\|\boldsymbol{I} - \boldsymbol{M}_i^{-1}\boldsymbol{A}\|_{\mathsf{F}} \leq (1+\epsilon) \min_{1 \leq j \leq n} \|\boldsymbol{I} - \boldsymbol{M}_j^{-1}\boldsymbol{A}\|_{\mathsf{F}}.$$

Proof. Start by fixing any $j \in \{1, 2, \ldots, n\}$. If we take $k \geq \frac{12}{\epsilon^2(3-2\epsilon)} \log \frac{2n}{\delta}$, Theorem 3.2 ensures that

$$\sqrt{1-\epsilon}\, \|\boldsymbol{I} - \boldsymbol{M}_j^{-1}\boldsymbol{A}\|_{\mathsf{F}} \leq \mathrm{Stab}(\boldsymbol{A}, \boldsymbol{M}_j, k) \leq \sqrt{1+\epsilon}\, \|\boldsymbol{I} - \boldsymbol{M}_j^{-1}\boldsymbol{A}\|_{\mathsf{F}}, \tag{3.9}$$

except with probability at most $\frac{\delta}{n}$. In particular, if we unfix j the probability at least one of the $\mathrm{Stab}(\boldsymbol{A}, \boldsymbol{M}_j, k)$ does not satisfy Equation 3.9 is at most $\sum_{j=1}^{n} \frac{\delta}{n} = \delta$ by a union bound. (Note that we did not need independence of the estimates $\mathrm{Stab}(\boldsymbol{A}, \boldsymbol{M}_j, k)$ here; this is why reusing the sketching matrix $\boldsymbol{Q}$ is valid.) Thus with probability at least $1 - \delta$ all estimates $\mathrm{Stab}(\boldsymbol{A}, \boldsymbol{M}_j, k)$ satisfy Equation 3.9 simultaneously.

Write $\boldsymbol{M}_i$ for the candidate preconditioner returned by Algorithm 4, and write $\boldsymbol{M}_\star$ for a candidate preconditioner which satisfies

$$\|\boldsymbol{I} - \boldsymbol{M}_\star^{-1}\boldsymbol{A}\|_{\mathsf{F}} = \min_{1 \leq j \leq n} \|\boldsymbol{I} - \boldsymbol{M}_j^{-1}\boldsymbol{A}\|_{\mathsf{F}}. \tag{3.10}$$

Then since the *estimate* of the stability of $\boldsymbol{M}_i$ was at most that of $\boldsymbol{M}_\star$ by minimality, the simultaneous bounds of Equation 3.9 give

$$\sqrt{1-\epsilon}\|\boldsymbol{I} - \boldsymbol{M}_i^{-1}\boldsymbol{A}\|_{\mathsf{F}} \leq \mathrm{Stab}(\boldsymbol{A}, \boldsymbol{M}_i, k) \leq \mathrm{Stab}(\boldsymbol{A}, \boldsymbol{M}_\star, k) \leq \sqrt{1+\epsilon}\|\boldsymbol{I} - \boldsymbol{M}_\star^{-1}\boldsymbol{A}\|_{\mathsf{F}}$$

except with probability at most δ. Rearranging the inequality gives the desired result after substituting Equation 3.10.

The final simplified bound results from the scalar inequality $\sqrt{\frac{1+\epsilon}{1-\epsilon}} \leq 1 + \frac{4}{3}\epsilon$ when $0 \leq \epsilon < 2/5$ and simple algebraic manipulation. ∎

The Constant in Theorem 3.2 is Tight

Most of the theory presented in this chapter relies on Theorem 3.2 to create more sophisticated bounds. Since Algorithm 3 is at its core a repurposing of a trace estimator using only matrix vector products, the work (Wimmer et al., 2014) applies and ensures that no randomized, adaptive algorithm for estimating the stability $\|\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A}\|_{\mathsf{F}}^2 = (\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A})^*(\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A})$ could possibly use asymptotically fewer matrix-vector multiplies so long as the algorithm only has access to $\|(\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A})\boldsymbol{q}\|_2$ for query vectors $\boldsymbol{q}$. In this sense, Algorithm 3 is optimal.

The theoretically-inclined practitioner, however, also cares about knowing the optimality of our *analysis* in Theorem 3.2. The following Theorem says that our analysis in Theorem 3.2 is asymptotically tight even up to the

leading effective constant $12/(3-2\epsilon)$ which tends to 4 for small ϵ. In the proof, $W(x) = \log x - \log\log x + o(1) = \Theta(\log x)$ as $x \to \infty$ is the Lambert-W function (Hoorfar and Hassani, 2007) as in Chapter 2. Note that the proof is analogous to Theorem 2.6.

Theorem 3.4. Fix some $0 < \delta < 1/10$. For any underlying dimension d, there exists a positive semi-definite matrix $\boldsymbol{A} \in \mathbb{F}^{d\times d}$, positive definite matrix $\boldsymbol{M} \in \mathbb{F}^{d\times d}$, and some $\epsilon_0 > 0$ so that for any $0 < \epsilon < \epsilon_0$, taking $k = \lfloor \frac{4}{\epsilon^2}\log\frac{1}{\sqrt{8\pi}\delta} - \frac{2}{\epsilon^2}\log\log\frac{1}{\sqrt{8\pi}\delta}\rfloor$ guarantees the stability estimate $\mathrm{Stab}(\boldsymbol{A},\boldsymbol{M},k)$ returned by Algorithm 3 fails to satisfy the equation

$$\sqrt{1-\epsilon}\|\boldsymbol{I}-\boldsymbol{M}^{-1}\boldsymbol{A}\|_{\mathsf{F}} \le \mathrm{Stab}(\boldsymbol{A},\boldsymbol{M},k) \le \sqrt{1+\epsilon}\|\boldsymbol{I}-\boldsymbol{M}^{-1}\boldsymbol{A}\|_{\mathsf{F}}$$

with probability at least δ.

Proof. If $Z \sim \mathcal{N}(0,1)$ is a standard normal random variable then

$$\mathbf{P}(|Z| \ge t) = 2\,\mathbf{P}(Z > t) > \sqrt{\frac{2}{\pi}}\frac{t}{t^2+1}e^{-t^2/2} \ge \frac{1}{\sqrt{2\pi}}\frac{1}{t}e^{-t^2/2} \tag{3.11}$$

by (Gordon, 1941) for all $t \ge 1$. Setting the right hand side of Inequality 3.11 to 2δ and solving gives

$$\mathbf{P}\big(|Z| \ge \sqrt{W(8^{-1}\pi^{-1}\delta^{-2})}\big) > 2\delta \tag{3.12}$$

whenever $\sqrt{W(8^{-1}\pi^{-1}\delta^{-2})} \ge 1$, which is satisfied when $0 < \delta \le 1/10$.

Now let $\boldsymbol{A} = \boldsymbol{I} - \boldsymbol{e}_1\boldsymbol{e}_1^*$ and $\boldsymbol{M} = \boldsymbol{I}$, where $\boldsymbol{e}_1$ is the first standard basis vector. We can observe that

$$\|(\boldsymbol{I}-\boldsymbol{M}^{-1}\boldsymbol{A})\boldsymbol{q}\|_2^2 = \|\boldsymbol{e}_1(\boldsymbol{e}_1^*\boldsymbol{q})\|_2^2 = \boldsymbol{q}_1^2 \sim \chi^2 \tag{3.13}$$

if $\boldsymbol{q}$ is a standard Gaussian vector. In particular, the standard deviation of $\|(\boldsymbol{I}-\boldsymbol{M}^{-1}\boldsymbol{A})\boldsymbol{q}\|_2^2$ is $\sigma = \sqrt{2}$. Thus since $\mathrm{Stab}(\boldsymbol{A},\boldsymbol{M},k)$ is a sample average of independent copies of these random variables, fixing $k = \lfloor \frac{2}{\epsilon^2}W(\frac{1}{8\pi\delta^2})\rfloor$ ensures

$$\begin{aligned}
&\mathbf{P}(|\,\mathrm{Stab}(\boldsymbol{A},\boldsymbol{M},k)^2 - 1| > \epsilon) && (3.14)\\
&\quad = \mathbf{P}(\tfrac{\sqrt{k}}{\sigma}|\,\mathrm{Stab}(\boldsymbol{A},\boldsymbol{M},k)^2 - 1| \ge \tfrac{\sqrt{k}\epsilon}{\sigma}) && (3.15)\\
&\quad \ge \mathbf{P}\big(\tfrac{\sqrt{k}}{\sigma}|\,\mathrm{Stab}(\boldsymbol{A},\boldsymbol{M},k)^2 - 1| \ge \sqrt{W(8^{-1}\pi^{-1}\delta^{-2})}\big) && (3.16)\\
&\quad \to \mathbf{P}\big(|Z| \ge \sqrt{W(8^{-1}\pi^{-1}\delta^{-2})}\big) > 2\delta && (3.17)
\end{aligned}$$

by (Vershynin, 2018: Thm. 1.3.2) and Equation 3.12 as $k \to \infty$. This implies the existence of an ϵ_0 so that $0 < \epsilon < \epsilon_0$ ensures the relation

$$\sqrt{1-\epsilon}\|\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A}\|_{\mathsf{F}} \leq \mathrm{Stab}(\boldsymbol{A}, \boldsymbol{M}, k) \leq \sqrt{1+\epsilon}\|\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A}\|_{\mathsf{F}} \tag{3.18}$$

fails with probability *at least* δ under our relation defining k. The simpler condition on k given in the statement of this result follows from the bound $W(x) \geq \log(x) - \log\log(x)$ for all $x \geq e$ from (Hoorfar and Hassani, 2007: Thm. 2). ■

An Improvement in the Presence of a Clear Winner

Algorithm 4 is extremely easy to implement and works well in practice, as we shall see in Section 3.2. Nevertheless, if we are selecting between preconditioners where some are clearly worse than the optimal preconditioner in terms of stability, our method seems excessive. Intuitively, we should be able to tell that terrible preconditioners will not be optimal with very rudimentary information. Algorithm 5 presents such a revision to Algorithm 4, iteratively refining the stability estimates we have and filtering out any preconditioners as soon as we can be confident they will not be optimal. Note that the algorithm crucially relies on the bounds from Section 3.1.4.

We can prove that Algorithm 5 is actually an improvement over Algorithm 4 by making an anti-concentration assumption about the input stabilities.

Theorem 3.5. Let $\boldsymbol{A} \in \mathbb{F}^{d\times d}$ be an arbitrary matrix, $\boldsymbol{M}_1, \boldsymbol{M}_2, \ldots, \boldsymbol{M}_n \in \mathbb{F}^{d\times d}$ be invertible candidate preconditioners for $\boldsymbol{A}$, $0 < \epsilon < 1/2$, and $0 < \delta < 1$. Denoting $i^\star \in \arg\min_{1\leq j\leq n} \|\boldsymbol{I} - \boldsymbol{M}_j^{-1}\boldsymbol{A}\|_{\mathsf{F}}$, we will write

$$F(t) = \frac{1}{n}\left|\left\{j : j \in \{1, 2, \ldots, n\} \text{ and } \frac{\|\boldsymbol{I} - \boldsymbol{M}_j^{-1}\boldsymbol{A}\|_{\mathsf{F}}}{\|\boldsymbol{I} - \boldsymbol{M}_{i^\star}^{-1}\boldsymbol{A}\|_{\mathsf{F}}} \leq 1 + t\right\}\right|$$

for the (shifted) cumulative distribution function of the input relative stabilities. If $F(t) \leq ct$ uniformly over $t \in [\epsilon/2, 2]$ for some positive constant c, then Algorithm 5 returns a preconditioner $\boldsymbol{M}_i$ satisfying

$$\|\boldsymbol{I} - \boldsymbol{M}_i^{-1}\boldsymbol{A}\|_{\mathsf{F}} \leq \sqrt{\frac{1+\epsilon}{1-\epsilon}} \min_{1\leq j\leq n} \|\boldsymbol{I} - \boldsymbol{M}_j^{-1}\boldsymbol{A}\|_{\mathsf{F}}.$$

with probability at least $1 - \delta$ using strictly fewer floating point operations than running $24n(1 + \frac{2c}{\epsilon}) \log \frac{2n}{\delta} + 24n(1 + \frac{2c}{\epsilon}) \log\log_2 \frac{2}{\epsilon}$ iterations of the

Algorithm 5: An improvement to Algorithm 4 when there is a relatively clear winner among the candidate preconditioners.

Data: A matrix $\boldsymbol{A} \in \mathbb{F}^{d\times d}$, n candidate preconditioners $\boldsymbol{M}_1$, $\boldsymbol{M}_2$,...,$\boldsymbol{M}_n \in \mathbb{F}^{d\times d}$, an accuracy parameter $0 < \epsilon < \frac{1}{2}$ and an acceptable failure probability $0 < \delta < 1$.

Result: A preconditioner $\boldsymbol{M}_i$ for which the stability criterion $\|\boldsymbol{I} - \boldsymbol{M}_i^{-1}\boldsymbol{A}\|_{\mathsf{F}}$ is an ϵ-multiplicative approximation to the minimum possible among the candidate preconditioners, except with probability at most δ.

$\epsilon_{\mathsf{cur}} \leftarrow 1$
$P \leftarrow \{1, 2, \ldots, n\}$
$T \leftarrow \lceil \log_2 \frac{1}{\epsilon} \rceil$
for $t = 1, 2, \ldots, T$ **do**
  $\epsilon_{\mathsf{cur}} \leftarrow \epsilon_{\mathsf{cur}}/2$
  $k \leftarrow \frac{6}{\epsilon_{\mathsf{cur}}^2} \log \frac{2T|P|}{\delta}$
  $S_i \leftarrow \mathrm{Stab}(\boldsymbol{A}, \boldsymbol{M}_i, k)$ for all $i \in P$
  $i^\star = \arg\min_{i\in P} S_i$
  $P \leftarrow \left\{ i \in P : S_i \leq S_{i^\star} \sqrt{\frac{1+\epsilon_{\mathsf{cur}}}{1-\epsilon_{\mathsf{cur}}}} \right\}$
end
Return $\boldsymbol{M}_{i^\star}$

preconditioned conjugate gradients algorithm in $\boldsymbol{A}$ with the most expensive preconditioner $\boldsymbol{M}_j$ in terms of the number of floating point operations required to compute $\boldsymbol{M}_j^{-1}\boldsymbol{y}$ for input vectors $\boldsymbol{y}$.

Proof. The same Bonferroni-correction argument from the proof of Theorem 3.3 ensures that

$$\sqrt{1-\epsilon_{\mathsf{cur}}}\,\|\boldsymbol{I} - \boldsymbol{M}_i^{-1}\boldsymbol{A}\|_{\mathsf{F}} \leq S_i \leq \sqrt{1+\epsilon_{\mathsf{cur}}}\,\|\boldsymbol{I} - \boldsymbol{M}_i^{-1}\boldsymbol{A}\|_{\mathsf{F}}, \tag{3.19}$$

simultaneously for all $i \in P$ over the course of the algorithm, except with probability at most δ. The rest of the proof will only rely on property 3.19, so everything we say will hold with this same probability.

If $i_t^\star$ is the $i^\star$ set in step t of the algorithm and $i^\star \in \arg\min_{1\leq i\leq n} \|\boldsymbol{I} - \boldsymbol{M}_i^{-1}\boldsymbol{A}\|_{\mathsf{F}}$ is in P before the filtering at the end of step t, Equation 3.19 implies

$$S_{i^\star} \leq \sqrt{1+\epsilon_{\mathsf{cur}}}\|\boldsymbol{I}-\boldsymbol{M}_{i^\star}^{-1}\boldsymbol{A}\|_{\mathsf{F}} \leq \sqrt{\frac{1+\epsilon_{\mathsf{cur}}}{1-\epsilon_{\mathsf{cur}}}}\sqrt{1-\epsilon_{\mathsf{cur}}}\|\boldsymbol{I}-\boldsymbol{M}_{i_t^\star}^{-1}\boldsymbol{A}\|_{\mathsf{F}} \leq \sqrt{\frac{1+\epsilon_{\mathsf{cur}}}{1-\epsilon_{\mathsf{cur}}}} S_{i_t^\star}.$$

Thus, since $i^\star \in P$ initially, we know by induction that $i^\star \in P$ throughout the process of the entire algorithm. Now consider P in the final step $t = T$ of Algorithm 5. Since $i^\star \in P$,

$$\sqrt{1-\epsilon_{\mathsf{cur}}}\|\boldsymbol{I} - \boldsymbol{M}_{i_T^\star}^{-1}\boldsymbol{A}\|_{\mathsf{F}} \leq S_{i_T^\star} \leq S_{i^\star} \leq \sqrt{1+\epsilon_{\mathsf{cur}}}\|\boldsymbol{I} - \boldsymbol{M}_{i^\star}^{-1}\boldsymbol{A}\|_{\mathsf{F}}. \tag{3.20}$$

Rearranging and realizing that $\epsilon_{\mathsf{cur}} = 2^{-\lceil \log_2 \frac{1}{\epsilon} \rceil} \leq 2^{-\log_2 \frac{1}{\epsilon}} = \epsilon$ at $t = T$ gives our desired approximation guarantee.

Now we will exhibit the runtime bound by bounding $|P|$ at each step of Algorithm 5. We claim that $|P| \leq 4cn2^{-t}$ for all $t = 1, 2, \ldots, T$. To see this, note that if a candidate preconditioner $\boldsymbol{M}_j$ is retained after filtering in any step t of the algorithm,

$$\|\boldsymbol{I} - \boldsymbol{M}_j^{-1}\boldsymbol{A}\|_{\mathsf{F}} \leq \frac{S_j}{\sqrt{1-\epsilon_{\mathsf{cur}}}} \leq \frac{\sqrt{1+\epsilon_{\mathsf{cur}}}}{1-\epsilon_{\mathsf{cur}}} S_{i_t^\star} \leq \frac{1+\epsilon_{\mathsf{cur}}}{1-\epsilon_{\mathsf{cur}}}\|\boldsymbol{I} - \boldsymbol{M}_{i^\star}^{-1}\boldsymbol{A}\|_{\mathsf{F}}. \tag{3.21}$$

Thus the number of elements in P just after step t in the algorithm is at most $nF(4\epsilon_{\mathsf{cur}}) \leq 4cn2^{-t}$ since $\frac{1+x}{1-x} \leq 1 + 4x$ for $0 \leq x \leq 1/2$. Our runtime bound follows from the sum

$$\sum_{t=1}^{T-1} |P_t| \frac{6}{(2^{-t})^2} \log \frac{2T|P_t|}{\delta} \leq \sum_{t=1}^{T-1} 4cn2^{-t} \tfrac{6}{(2^{-t})^2} \log \tfrac{2nT}{\delta} \leq 24cn2^T \log \tfrac{2nT}{\delta} \tag{3.22}$$

where P_t is the set P during iteration t of the algorithm. This gives the number of matrix-vector multiplies of the form $(\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A})\boldsymbol{q}$ used by the algorithm *after* the first step. To see the final form, add on the $24n \log \frac{2nT}{\delta}$ multiplies done during the first iteration $t = 1$ and plug in $T = \lceil \log_2 \frac{1}{\epsilon} \rceil \leq \log_2 \frac{1}{\epsilon} + 1 = \log_2 \frac{2}{\epsilon}$. ■

The anti-concentration condition in Theorem 3.5 intuitively asserts that the stabilities of the preconditioners do not cluster around the minimal stability. This is satisfied, for example, if at most some number d of the candidate preconditioners have stability within a multiplicative factor 3 of the optimal stability. The resulting constant $c = \frac{2d}{n\epsilon}$ gives an asymptotic runtime bound for Algorithm 5 of $O(n \log \frac{n}{\delta} + n \log\log \frac{1}{\epsilon} + \frac{d}{\epsilon^2} \log \frac{n}{\delta} + \frac{d}{\epsilon^2} \log\log \frac{1}{\epsilon})$, decoupling the linear dependence in n with the polynomial accuracy dependence on $1/\epsilon^2$. Such an improvement is serious when n is moderately large; while this example is contrived many other distributions on input data satisfy the assumptions of Theorem 3.5 with the same constant c.

Of course, one would hope that Algorithm 5 does not perform terribly when the input data assumptions made in Theorem 3.5 are not satisfied. For example, this would happen when all preconditioners have extremely similar performance, to the point that even our target accuracy ϵ cannot distinguish their stabilities. Luckily, a constant $c = \frac{2}{\epsilon}$ always works in Theorem 3.5, so Algorithm 5 never suffers more than a multiplicative $O(\log\log\frac{1}{\epsilon})$ increase over Algorithm 4 in number of floating point operations needed to select a preconditioner.

3.2 Experiments

This paper takes the working hypothesis that preconditioner stability is a good proxy for the performance of the preconditioned conjugate gradients algorithm, and runs with it to create and theoretically verify algorithms to select optimal preconditioners under this metric. The present section will jointly test the good-proxy hypothesis and our algorithms by evaluating Algorithm 4 empirically in a number of realistic settings. In particular, we will see how well Algorithm 4 can select the candidate preconditioner which minimizes the number of conjugate gradients iterations required to achieve some fixed approximation quality.

3.2.1 Experiments with Sparse Systems

First we attempt a generic experiment on a collection of real-world sparse linear systems and simple preconditioners. For the target system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, we fix a sampled $\boldsymbol{b} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ for the entire experiment. The positive definite matrices $\boldsymbol{A}$ are taken from the SuiteSparse/University of Florida Sparse Matrix Collection (Davis and Hu, 2011). We include all matrices from the `Boeing` and `GHS_psdef` groups which have between 100,000 and 2,250,000 non-zero entries and are strictly positive definite.

We include nine candidate preconditioners for Algorithm 4 to select between. All of the candidate preconditioners are block diagonal. This choice was made to get around some existence and algorithmic issues that accompany other common preconditioners like incomplete Cholesky factorizations (Benzi, 2002). The first candidate preconditioner is the trivial preconditioner $\boldsymbol{I}$, which is equivalent to using no preconditioner at all. The preconditioner $\boldsymbol{D}_\ell$ denotes a block-diagonal pinching/truncation of the matrix $\boldsymbol{A}$ with block size ℓ. The preconditioner $\boldsymbol{R}_\ell$ is the same block-diagonal pinching, but performed after a Reverse Cuthill-McKee ordering

Matrix	Conjugate Gradients Iterations With Various Preconditioners								
`apache1`	3,538	3,513	3,286	3,283	3,270	3,265	3,269	3,710	3,693
`crystm01`	122	54	39	34	30	27	27	24	21
`crystm02`	138	54	38	35	34	29	30	24	24
`crystm03`	143	54	38	34	33	30	29	25	24
`cvxbqp1`	16,424	11,337	11,338	11,332	11,331	11,330	11,328	10,148	10,353
`gridgena`	3,658	3,542	2,659	2,572	2,504	2,504	2,479	2,892	2,863
`jnlbrng1`	139	131	126	126	125	125	125	130	130
`minsurfo`	94	88	64	63	63	62	62	88	88
`msc10848`	—	5,659	3,791	3,028	2,793	2,656	2,628	2,192	2,092
`obstclae`	66	65	49	48	47	47	47	65	65
`oilpan`	48,291	28,065	12,804	8,167	5,476	4,992	4,127	4,757	4,433
`torsion1`	66	65	49	48	47	47	47	65	65
`wathen100`	327	45	44	44	44	44	44	42	42
`wathen120`	378	45	45	45	44	44	44	43	43

Table 3.1 This table reports the number of iterations taken by the conjugate gradients algorithm to report an approximate solution $\tilde{\boldsymbol{x}}$ to the linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ for specified test matrices $\boldsymbol{A}$, a constant sampled standard normally distributed $\boldsymbol{b} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$, and various candidate preconditioners.

of the matrix (Cuthill and McKee, 1969). To ensure uniqueness and clarity, blocking is performed by taking the matrix $\boldsymbol{A} \in \mathbb{F}^{d \times d}$ and constructing a block diagonal matrix $\boldsymbol{M}$ with blocks of the form $\boldsymbol{A}(m\ell : \min\{d, (m+1)\ell\}, m\ell : \min\{d, (m+1)\ell\})$ for $m = 0, 1, 2, \ldots$. Since $\boldsymbol{A}$ is positive definite, the resulting preconditioners $\boldsymbol{M}$ are also positive definite (Bhatia, 2009: Ex. 2.2.1.(viii)).

In Table 3.1 we present the number of iterations the preconditioned conjugate gradients algorithm took for each test matrix and preconditioner pair. The algorithm was run until the approximate solution $\tilde{\boldsymbol{x}}$ satisfied $\|\boldsymbol{A}\tilde{\boldsymbol{x}} - \boldsymbol{b}\|_2 \leq 10^{-9}\|\boldsymbol{b}\|_2$. The number of iterations was capped 50,000. Entries in Table 3.1 achieving this artificial stopping criterion are overwritten with '—'. The conjugate gradients algorithm applied to the matrices `bcsstk36`, `bcsstk38`, `msc23052`, and `vanbody` did not converge with any candidate preconditioner, so they are omitted in Table 3.1.

Observe that even though larger block sizes ℓ ought to create better approximations of the original matrix, there are situations when smaller block sizes result in fewer conjugate gradients iterations. Similarly, there are some situations when the original ordering of the data is preferable over the Reverse Cuthill-McKee ordering, and vice-versa. As a result, it is unclear

a-priori which preconditioner one should choose to solve the linear system, and this is why someone might wish to use Algorithm 4 to automate that choice.

We test this use of Algorithm 4 under two parameter settings $k = 10$ and $k = 50$. Algorithm 4 is run for 1,000 independent trials for each matrix-preconditioner-k pairing. After the fact, we compare the number of iterations of preconditioned conjugate gradients would be necessary when using the recommendation of Algorithm 4 relative to the minimal number of iterations possible if we knew in advance how many iterations each preconditioner would use.

Results

The results of our generic real-world-use experiment are presented in Table 3.2. Every cell is an approximation ratio, i.e. the number of iterations an algorithm for selecting preconditioners took divided by the minimal number of iterations possible using our set of candidate preconditioners. As such, an entry of 1.00 is optimal and represents the minimal-number-of-iterations preconditioner being correctly selected. The column 'Worst-Case' reports the approximation ratio if one deterministically selected the maximal-number-of-iterations preconditioner in each setting. The column 'Random' reports the expected approximation ratio if one were to select a candidate preconditioner from Table 3.1 uniformly at random. The columns corresponding to Algorithm 4 gives statistics of the empirical distribution of approximation ratios seen over the 1,000 independent trial runs of the method.

A clear take-away from Table 3.2 is that Algorithm 4 performs admirably in practice for selecting preconditioners. For 10 of the 14 test matrices reported, setting $k = 10$ always picks the optimal preconditioner for the problem across every one of the 1,000 trials. If we take $k = 50$, this happens for 11 of the 14 test matrices. Moreover, even when the accuracy parameter $k = 10$ the returned preconditioner never needs more than 15% more iterations than the optimal choice. For the practitioner, such a 15% increase in iteration count in a trade for robustness would in most cases be completely acceptable. As such, Algorithm 4 appears to be useful for real-world problems when selecting preconditioners.

Of course, one might wonder if taking k to be even larger would result in approximation ratios concentrating more uniformly at the ideal 1.00 mark. Unfortunately, this won't happen in general, and is where the good-proxy

Matrix	Worst-Case	Random	Algorithm 4 Approximation Ratio					
			$k = 10$			$k = 50$		
			Min	Mean	Max	Min	Mean	Max
`apache1`	1.14	1.05	1.00	1.00	1.00	1.00	1.00	1.00
`crystm01`	5.81	2.00	1.00	1.00	1.00	1.00	1.00	1.00
`crystm02`	5.75	1.88	1.00	1.00	1.00	1.00	1.00	1.00
`crystm03`	5.96	1.90	1.00	1.00	1.00	1.00	1.00	1.00
`cvxbqp1`	1.62	1.15	1.12	1.12	1.12	1.12	1.12	1.12
`gridgena`	1.48	1.15	1.00	1.00	1.01	1.00	1.00	1.00
`jnlbrng1`	1.11	1.03	1.04	1.04	1.04	1.04	1.04	1.04
`minsurfo`	1.52	1.20	1.00	1.00	1.00	1.00	1.00	1.00
`msc10848`	23.90	3.97	1.00	1.00	1.00	1.00	1.00	1.00
`obstclae`	1.40	1.18	1.00	1.00	1.00	1.00	1.00	1.00
`oilpan`	11.70	3.26	1.00	1.09	1.15	1.07	1.08	1.15
`torsion1`	1.40	1.18	1.00	1.00	1.00	1.00	1.00	1.00
`wathen100`	7.79	1.79	1.00	1.00	1.00	1.00	1.00	1.00
`wathen120`	8.79	1.89	1.00	1.00	1.00	1.00	1.00	1.00

Table 3.2 This table summarizes the performance of Algorithm 4 for each matrix in Table 3.1, reporting statistics of the empirical number of iterations given by the algorithm compared to picking the worst-possible preconditioner (in terms of number of CG iterations) or choosing arbitrarily at random. Since the conjugate gradients algorithm did not converge for the matrix `msc10848` with no preconditioner, the 'Worst-Case' and 'Random' columns are *lower* bounds for their true values in that row only.

hypothesis is put to the test. For the `oilpan` matrix, increasing k from 10 to 50 *raises* the best-seen approximation ratio given by Algorithm 4 from 1.00 to 1.07. Increasing k causes the preconditioner returned by Algorithm 4 to concentrate further around the true minimal-stability preconditioner (see Theorem 3.3), and so this implies that the preconditioner stability criterion itself is not perfect and will not in general forecast the exact preconditioner resulting in the minimal number of conjugate gradients iterations. As the previous paragraph details, though, stability is quite a good proxy for iteration count in spite of this flaw.

3.2.2 Experiments with Kernel Regression Preconditioners

This section will show that Algorithm 4 can turn two simple preconditioners for the standard kernel regression problem into a robust, state-of-the-art preconditioning method. As a corollary of this investigation, we exhibit how Algorithm 4 performs well in situations when the 'minimal accuracy' criterion for selecting preconditioners fails, something left unanswered in the previous experiment.

A Quick Review

Kernel regression is a common statistical technique for nonlinear regression. In this setting, we have a dataset $\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_d, y_d)\}$ consisting of $\boldsymbol{x}_i \mapsto y_i$ mappings from Euclidean space $\mathbb{R}^d$ to the real line $\mathbb{R}$. We wish to find coefficients $\boldsymbol{\alpha} \in \mathbb{R}^d$ so that the functional mapping

$$\boldsymbol{x} \mapsto f(\boldsymbol{x}) = \sum_{i=1}^{d} \boldsymbol{\alpha}_i k(\boldsymbol{x}, \boldsymbol{x}_i) \tag{3.23}$$

faithfully represents the empirical mapping in the sense that $f(\boldsymbol{x}_i) \approx y_i$. In general, $k(\boldsymbol{x}, \boldsymbol{y})$ is just required to be a positive definite function (kernel), but in our experiment, we will only use the squared exponential kernel $k(\boldsymbol{x}, \boldsymbol{y}) = \exp(-\frac{\|\boldsymbol{x}-\boldsymbol{y}\|_2^2}{2\ell^2})$, parametrized by the length-scale ℓ which controls the derivative of the model $f(\boldsymbol{x})$. The coefficients $\boldsymbol{\alpha}$ are found by solving the system

$$\boldsymbol{\alpha} = (\boldsymbol{K} + \sigma_n^2 \boldsymbol{I})^{-1} \boldsymbol{y} \tag{3.24}$$

where the positive definite Gram matrix $\boldsymbol{K}_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$, the output vector $\boldsymbol{y} = (y_1, y_2, \ldots, y_d)$, and the noise standarad deviation $\sigma_n > 0$ is used for regularization so that the model $f(\boldsymbol{x})$ fits well on out-of-sample data. In almost all kernel regression problems, $\boldsymbol{K}$ and hence $\boldsymbol{K} + \sigma_n^2 \boldsymbol{I}$ are dense. We will ignore the issue of actually selecting the parameters σ_n and ℓ in this experiment. See (Williams and Rasmussen, 2006) for more background on this model and associated inference procedure.

Related Work

This experiment will test a preconditioning procedure for solving the linear system $(\boldsymbol{K} + \sigma_n^2 \boldsymbol{I})\boldsymbol{\alpha} = \boldsymbol{y}$ via conjugate gradients. There has been a recent interest in this general iterative framework for kernel regression (Avron et al.,

2017; Cutajar et al., 2016; Rudi et al., 2017). Much of the work has focused on developing viable preconditioners since in general Gram matrices $\boldsymbol{K}$ can be poorely conditioned, which results in poor conditioning for the raw system in $\boldsymbol{K} + \sigma_n^2 \boldsymbol{I}$ unless the noise standard deviation σ_n is unnaturally large.

The work of Cutajar et al. (Cutajar et al., 2016) does some initial leg-work in this area, proposing eight candidate preconditioners. These preconditioners include a block-diagonal approximation of $\boldsymbol{K} + \sigma_n^2 \boldsymbol{I}$, adding a larger regularizer σ_n^2 and solving recursively, a Nyström approximation of the Gram matrix using $\sqrt{n}$ datapoints as inducing points chosen uniformly at random, a coupling of the Nyström approximation with a block-diagonal approximation, or replacing $\boldsymbol{K}$ with an optimal low-rank factorization which can be computed via a randomized SVD (Halko et al., 2011) or the Lanczos method (Golub and Van Loan, 2012: Sec. 10.1). Both (Cutajar et al., 2016) and the work (Avron et al., 2017) of Avron et al. use the the Fourier features method of Rahimi and Recht (Rahimi and Recht, 2008) to create a preconditioner which replaces $\boldsymbol{K}$ with a sketched version $\tilde{\boldsymbol{K}}$. The latter paper (Avron et al., 2017) also proposes using the TensorSketch method of (Pagh, 2013) for creating a sketched preconditioner when using the polynomial kernel $k(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^* \boldsymbol{y})^q$, though unfortunately the necessary sketching dimension is exponential in q. The work of Rudi et al. (Rudi et al., 2017) also uses the Nyström-based preconditioner like (Cutajar et al., 2016), combining it with other computational tricks.

The problem with the above works is illustrated perfectly in Figure 1 of (Cutajar et al., 2016). For every known preconditioner among the works who report this statistic, there exist parameter settings for which using no preconditioner results in fewer iterations than using the preconditioner when solving for $\boldsymbol{\alpha}$ via conjugate gradients. As such, these schemes are not robust, and it is unclear how one would choose a performant preconditioner in practice.

Two Simple Geometrically Driven Preconditioners

Here we detail the two candidate preconditioners which we will use in our experiments. They both utilize a geometrically-motivated reordering of the data to achieve superior performance to the preconditioners of (Cutajar et al., 2016) in certain areas of the parameter space.

The first preconditioner is a simple block diagonal pinching of a reordering of the data. The kernel regression model under the squared exponential kernel effectively asserts that points nearby in ℓ^2 ought to

have similar outputs y. If the input data is highly clustered in ℓ^2, our model then ought to largely ignore points from different clusters when considering a point in some cluster. The first preconditioning algorithm turns this 'ought to' statement directly into an approximation of the Gram matrix $\boldsymbol{K}$. We first cluster the data $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_d\}$ in ℓ^2 via the `k-means` or `k-means++` (Arthur and Vassilvitskii, 2007) algorithm with $c = \lceil\sqrt{d}\rceil$ clusters, constructing a permutation matrix $\boldsymbol{P}$ that places points in the same cluster next to each other. At this point, we precondition the re-ordered system $(\boldsymbol{PKP}^* + \sigma_n^2\boldsymbol{I})\boldsymbol{P\alpha} = \boldsymbol{Py}$ by creating a block-diagonal pinching of the re-ordered matrix $\boldsymbol{PKP}^*$ where each block corresponds to the points within a cluster. The resulting preconditioner is that pinching $\hat{\boldsymbol{K}}$ plus the true noise term $\sigma_n^2\boldsymbol{I}$.

Assuming the clusters are approximately equal in size, computing the Cholesky factorization of the preconditioner takes $O(n^2)$ floating point operations and computing $\boldsymbol{M}^{-1}\boldsymbol{z}$ for a vector $\boldsymbol{z}$ takes $O(n^{1.5})$ floating point operations. Computing a matrix-vector product of the form $\boldsymbol{Kz}$ takes $\Theta(n^2)$ floating point operations since $\boldsymbol{K}$ is dense, so this preconditioner won't raise the per-iteration complexity over regular conjugate gradients. Of course, usability assumes the `k-means` algorithm converges quickly, but in practice this is not an issue. Moreover, if we fix the resulting sparsity pattern of the preconditioner, this preconditioner exactly minimizes the accuracy $\|\boldsymbol{M} - \boldsymbol{PKP}^* + \sigma_n^2\boldsymbol{I}\|_\mathsf{F}$ over all matrices with the same sparsity pattern. Since the identity matrix $\boldsymbol{I}$ also has this sparsity pattern, we would always choose this preconditioner over the identity matrix if using the accuracy criterion.

The second preconditioner is a slightly more complex version of the first. After computing the permuted matrix $\boldsymbol{PKP}^*$, we compute a truncated rank-r approximation $\boldsymbol{U}\Lambda\boldsymbol{U}^*$ of $\boldsymbol{PKP}^*$ where $\Lambda \in \mathbb{R}^{r\times r}$ is diagonal and $\boldsymbol{U} \in \mathbb{R}^{d\times r}$ has orthonormal columns. At this point we compute the same block diagonal pinching $\tilde{\boldsymbol{E}}$ of the error in approximation $\boldsymbol{E} = \boldsymbol{PKP}^* - \boldsymbol{U}\Lambda\boldsymbol{U}^*$. The resulting preconditioner is then $\boldsymbol{U}\Lambda\boldsymbol{U}^* + \tilde{\boldsymbol{E}} + \sigma_n^2\boldsymbol{I}$. If r is a constant, we can solve systems in this preconditioner using the Woodbury identity (Golub and Van Loan, 2012: Sec. 2.1.4) in $O(n^2)$ floating point operations under the same assumption that the cluster sizes are approximately equal. Computing the low-rank factorization takes $O(n^2)$ floating point operations using either the Implicitly-Restarted Lanczos method (Sorensen, 1997) or a Randomized SVD (Halko et al., 2011), though for higher ranks r the latter method is preferable. In sum, then, this more sophisticated preconditioner does not raise the per-iteration asymptotic complexity of conjugate gradients so long as r is constant. Like the first preconditioner with no low-rank approximation term,

the low-rank approximation-based preconditioner is always more accurate than the identity matrix $\boldsymbol{I}$, and so the accuracy criterion would always pick this preconditioner over no preconditioner.

Observe that our approaches to these two preconditioners combines some intuition from (Cutajar et al., 2016) with geometric insight to create preconditioners that should intuitively be more representative of the underlying problem.

Experimental Design

We consider three datasets `Concrete`, `Power`, and `Protein`, which are identically the same as in (Cutajar et al., 2016). The `Concrete` dataset consists of $d = 1{,}029$ data points in $\mathbb{R}^8$. The `Power` dataset consists of $d = 9{,}567$ data points in $\mathbb{R}^4$. The `Protein` dataset consists of $d = 45{,}729$ data points in $\mathbb{R}^9$.

For each of these datasets, and each pair of parameters chosen from $\ell \in \{10^{-3}, 10^{-2}, \ldots, 10^{2}\}$ and $\sigma_n^2 \in \{10^{-2}, 10^{-4}, 10^{-6}\}$, we construct a kernel system $(\boldsymbol{K} + \sigma_n^2 \boldsymbol{I})\boldsymbol{\alpha} = \boldsymbol{y}$. This system is solved using conjugate gradients with (a) no preconditioner, (b) the geometric preconditioner with no low-rank approximation, and (c) the geometric preconditioner with a rank $r = 25$ low-rank approximation. We also solve the system using the preconditioner chosen by one run of Algorithm 4 among (a) no preconditioner, (b) the purely block-diagonal geometric preconditioner, and (c) the rank $r = 25$ low-rank approximation-based geometric preconditioner, using an accuracy parameter $k = 10$. We also attempt using Algorithm 4 with the same $k = 10$ if we restrict the choice to the two geometric preconditioners, ruling out the use of no preconditioner. In solving these systems, we record the number of conjugate gradients iterations needed to achieve an residual norm of $10^{-5}\sqrt{d}$ as in (Cutajar et al., 2016); a relative tolerance of $10^{-15}\|\boldsymbol{y}\|_2$ is also specified, though this is vacuous in comparison to the absolute tolerance. The solver is stopped after 10,000 iterations if the residual has not converged to within tolerance by then. The low-rank approximations are computed via `ARPACK` (Lehoucq et al., 1998) with a tolerance parameter of 10^{-5}.

Results

Figure 3.1 illustrates the relative improvement different preconditioning schemes have over using no preconditioner for each dataset and parameter combination. Each cell gives the logarithm of the ratio of the preconditioned

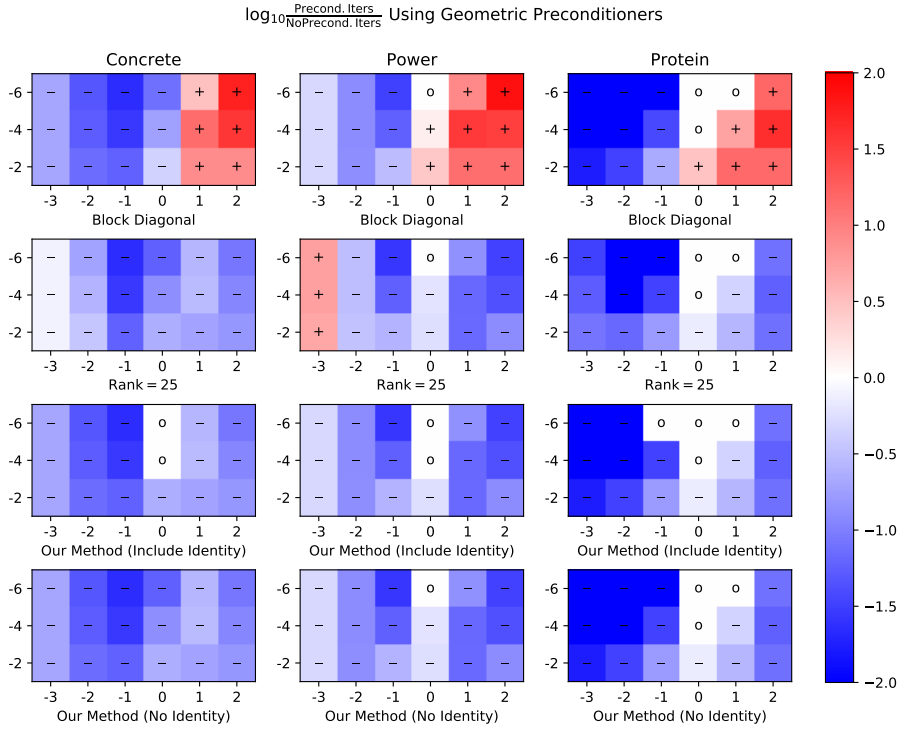

Figure 3.1 This figure presents the relative improvement of using our proposed preconditioners, or the one automatically chosen by Algorithm 4, with respect to using no preconditioner at all. Each individual matrix corresponds to a specific preconditioner and dataset pair. Each row gives the value of $\log \sigma_n^2$ used in the experiment, whereas each column corresponds to $\log \ell$. The absence of red cells in the result matrices corresponding to 'Our Method' indicates significant improvement over the results in (Cutajar et al., 2016).

conjugate gradients iterations to the non-preconditioned conjugate gradients iterations, i.e. the order of magnitude of the improvement granted by using the preconditioner. Accordingly, negative values (blue or '–') represent improvement through using the preconditioner, while positive values (red or '+') correspond to the preconditioned system requiring *more* iterations than using no preconditioner at all. Five of the cells for the `Protein` dataset with the purely block diagonal geometric preconditioner have relative improvements of more than two orders of magnitude. Another three preconditioners using a low-rank approximation with the `Protein` dataset have this property. In spite of this, we restrict the visual range of the

plot from −2 to 2 to allow Figure 3.1 to be compared easily to the identical presentation in Figure 1 of (Cutajar et al., 2016). No cell values exceed 2.

First we comment purely on the performance of the two geometrically motivated preconditioners. The main take-away is that the geometric permutation based on the `k-means` algorithm appears to truly help in creating a faithful preconditioner. As evidence, we can point to the fact that the simple geometric block-diagonal preconditioner gives, for five different parameter settings with the `Protein` dataset, a relative improvement better than every single preconditioner-parameters-dataset pair in (Cutajar et al., 2016). Phrased differently, at these parameter settings the number of iterations drops from 189, 111, 2,345, 618, and 10,000 (did not converge) to 1, 1, 3, 3, and 94 iterations, respectively. Moreover, the geometric preconditioner using a low-rank approximation for the `Concrete` dataset always outperforms using no preconditioner, something no preconditioner proposed in (Cutajar et al., 2016) can do. These improvements are genuine and stark, and again achieved by an extremely simple method just by relying on geometry.

Of course, one can rightfully point out that the block diagonal pinching is not robust as a preconditioner, just like many methods from (Cutajar et al., 2016). This is true; the block diagonal approximation works well for small length scales ℓ, as in these circumstances dependencies $\boldsymbol{K}_{ij}$ between far away data points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are shrunk, resulting in a genuine clustering of the underlying data where the intuition we used in justifying the preconditioner carries through. For large ℓ, the block diagonal preconditioner performs poorly because the matrix $\boldsymbol{K}$ looks more uniform and doesn't have a genuine clustered structure. Luckily, the more sophisticated preconditioners with added rank-25 terms perform well in precisely this regime, as the low-rank term can capture uniform structure in the Gram matrix $\boldsymbol{K}$. While this complicated preconditioner is not perfect, it is more robust to parameter changes than the analogous SVD-based preconditioner from (Cutajar et al., 2016). Between our two candidate preconditioners, at least one provides a performance boost over non-preconditioned conjugate gradients for every dataset and parameter setting chosen. Such a claim cannot be said about any pair of preconditioners in (Cutajar et al., 2016).

Since we have two quality preconditioners, each performing admirably in opposing parameter regimes, we might hope to get the best of both worlds by forecasting via Algorithm 4 which one will perform better than using no preconditioner and solving the system with that resulting preconditioner. This approach does quite well, as we can see in Figure 3.1. While Algorithm

4 does not always pick the best preconditioner in terms of minimizing the number of conjugate gradients iterations, it never selects a preconditioner which performs worse than using no preconditioner. That said, a preconditioner resulting in an exactly minimal number of iterations is chosen over 80% of the time if the 'use no preconditioner' option is included, and over 40% of the time the preconditioner ranking induced by our stability estimates exactly corresponds to the ranking induced by the true iteration count. If we exclude the 'use no preconditioner' option, which corresponds to an a-priori understanding that at least one of the geometric preconditioners works well, the former statistic jumps from 80% to an impressive 98.1%. This 'all blue' plot which represents a robust preconditioner regime can not be found using the techniques of (Cutajar et al., 2016). Moreover, the algorithm was able to return the advice 'use no preconditioner' in the face of uncertainty instead of suggesting the use of a poor preconditioner. This fact alone is highly desirable for the practitioner.

To confirm the importance of this chapter, it is necessary to show that our method performs well when the computationally simple accuracy method does not. As mentioned when detailing the construction of these preconditioners, the accuracy criterion would never choose the 'use no preconditioner' option over one of the geometric preconditioner. If we were just looking at the purely block-diagonal geometric preconditioner versus the 'use no preconditioner option', the accuracy criterion would result in a poor preconditioner (higher number of iterations than possible) exactly a third of the time with the `Concrete` dataset. Of these times that the accuracy method fails, the estimated stability criterion succeeds exactly half of the time. For the `Power` dataset, the accuracy method fails 44.4% of the time, but our estimated stability criterion succeeds in a quarter of these cases. While this behavior is not universal, it indicates that our method can be a crucial help when standard tools fail.

Finally, it is important to point out that in this setting, Algorithm 4 performed computation commensurate with taking 30 steps of conjugate gradients in total. Since in over half of the parameter-dataset pairs the non-preconditioned conjugate gradients algorithm took more than five times this number of iterations, and our method can in most situations reduce that full-solution cost significantly, this initial cost is acceptable.

Chapter 4

Conclusion

We have dived deep into the specific task of computing the trace of large, implicit matrices by using randomized algorithms, and outlined a number of incremental and novel contributions to the field. Our most important contributions in this area are:

- Showing that the best known sample complexity for trace estimation extends to any sub-Gaussian query distribution,
- Proving extremely tight lower bounds for the sample complexity of Hutchinson-type estimators, which tell us the constants in our current upper bounds are essentially optimal, and
- Proving why deterministic algorithms can't estimate traces.

By utilizing those results, we are able to create the first known feasible algorithms for computing preconditioner stability in the realm of the conjugate gradients algorithm. Our primary contributions here are:

- Proving that the preconditioner stability $\|\boldsymbol{I} - \boldsymbol{M}^{-1}\boldsymbol{A}\|_{\mathsf{F}}$ is impractical to even approximate deterministically,
- Showing that, nevertheless, sketching based randomized algorithms are entirely practical for computing this quantity,
- Building upon our stability estimation algorithm to give an easy-to-implement randomized algorithm which provably finds the 'best' of n candidate preconditioners in the time it takes to compute about $n \log n$ steps of conjugate gradients,

- Creating a theoretical improvement when there is a clearly optimal preconditioner by taking advantage of an anti-concentration assumption about the candidate stabilities, and, finally,
- Using our preconditioner selection algorithm to create a state-of-the-art preconditioned solver for kernel regression systems with relatively little leg-work.

Thus, our thesis is both practically helpful and of theoretical importance to both the scientific computing and machine learning/statistics communities.

Our work raises some important theoretical questions which would be ripe for future work. Most notably, it would be helpful if one could prove that preconditioner stability is truly a good proxy for the number of iterations the conjugate gradients algorithm will use. Such an analysis could take the form of a convergence guarantee for conjugate gradients which depends on the stability criterion instead of the usual condition number criterion. We can imagine proceeding towards this goal by analytically relating the stability to the condition number. On the other hand, it would be interesting to see if one could find a concrete example of a matrix and preconditioner for which the stability criterion fails wildly at predicting the number of iterations the conjugate gradients algorithm will take.

Bibliography

Dimitris Achlioptas. Database-friendly random projections. In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 274–281. ACM, 2001.

David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

Haim Avron. Counting triangles in large graphs using randomized matrix trace estimation. In *Workshop on Large-scale Data Mining: Theory and Applications*, volume 10, pages 10–9, 2010.

Haim Avron and Sivan Toledo. Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix. *Journal of the ACM (JACM)*, 58(2):8, 2011.

Haim Avron, Kenneth L Clarkson, and David P Woodruff. Faster kernel ridge regression using sketching and preconditioning. *SIAM Journal on Matrix Analysis and Applications*, 38(4):1116–1138, 2017.

Owe Axelsson and Victor Eijkhout. Vectorizable preconditioners for elliptic difference equations in three space dimensions. In *Advances in Parallel Computing*, volume 1, pages 299–321. Elsevier, 1990.

Michele Benzi. Preconditioning techniques for large linear systems: a survey. *Journal of computational Physics*, 182(2):418–477, 2002.

Michele Benzi, Daniel B Szyld, and Arno Van Duin. Orderings for incomplete factorization preconditioning of nonsymmetric problems. *SIAM Journal on Scientific Computing*, 20(5):1652–1670, 1999.

Rajendra Bhatia. *Positive definite matrices*, volume 16. Princeton university press, 2009.

Emmanuel J Candès and Benjamin Recht. Exact matrix completion via convex optimization. *Foundations of Computational mathematics*, 9(6):717, 2009.

Edmond Chow and Yousef Saad. Experimental study of ilu preconditioners for indefinite matrices. *Journal of Computational and Applied Mathematics*, 86 (2):387–414, 1997.

John D Cook. Upper and lower bounds for the normal distribution function, 2009.

Kurt Cutajar, Michael Osborne, John Cunningham, and Maurizio Filippone. Preconditioning kernel matrices. In *International Conference on Machine Learning*, pages 2529–2538, 2016.

Elizabeth Cuthill and James McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 1969 24th national conference*, pages 157–172. ACM, 1969.

Timothy A Davis and Yifan Hu. The University of Florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)*, 38(1):1, 2011.

Iain S Duff and Gerard A Meurant. The effect of ordering on preconditioned conjugate gradients. *BIT Numerical Mathematics*, 29(4):635–657, 1989.

Gene H Golub and Charles F Van Loan. *Matrix computations*, volume 3. JHU Press, 2012.

Robert D Gordon. Values of mills' ratio of area to bounding ordinate and of the normal probability integral for large values of the argument. *The Annals of Mathematical Statistics*, 12(3):364–366, 1941.

Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288, 2011.

Abdolhossein Hoorfar and Mehdi Hassani. Approximation of the Lambert-*W* function and hyperpower function. *Research report collection*, 10(2), 2007.

Michael F Hutchinson. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communications in Statistics-Simulation and Computation*, 19(2):433–450, 1990.

Richard B Lehoucq, Danny C Sorensen, and Chao Yang. *ARPACK users' guide: solution of large-scale eigenvalue problems with implicitly restarted Arnoldi methods*, volume 6. Siam, 1998.

Yi Li, Huy L Nguyen, and David P Woodruff. On sketching matrix norms and the top singular vector. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 1562–1581. Society for Industrial and Applied Mathematics, 2014.

Ping Ma, Michael W Mahoney, and Bin Yu. A statistical perspective on algorithmic leveraging. *The Journal of Machine Learning Research*, 16(1): 861–911, 2015.

Rasmus Pagh. Compressed matrix multiplication. *ACM Transactions on Computation Theory (TOCT)*, 5(3):9, 2013.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in neural information processing systems*, pages 1177–1184, 2008.

Farbod Roosta-Khorasani and Uri Ascher. Improved bounds on sample size for implicit matrix trace estimators. *Foundations of Computational Mathematics*, 15(5):1187–1212, 2015.

Alessandro Rudi, Luigi Carratino, and Lorenzo Rosasco. Falkon: An optimal large scale kernel method. In *Advances in Neural Information Processing Systems*, pages 3888–3898, 2017.

Danny C Sorensen. Implicitly restarted arnoldi/lanczos methods for large scale eigenvalue calculations. In *Parallel Numerical Algorithms*, pages 119–165. Springer, 1997.

Lloyd N Trefethen and David Bau III. *Numerical linear algebra*, volume 50. Siam, 1997.

Roman Vershynin. *High-dimensional probability: An introduction with applications in data science*, volume 47. Cambridge University Press, 2018.

Nisheeth K Vishnoi et al. $\boldsymbol{Lx} = \boldsymbol{b}$. *Foundations and Trends® in Theoretical Computer Science*, 8(1–2):1–141, 2013.

Larry Wasserman. *All of statistics: a concise course in statistical inference*. Springer Science & Business Media, 2013.

Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT Press Cambridge, MA, 2006.

Karl Wimmer, Yi Wu, and Peng Zhang. Optimal query complexity for estimating the trace of a matrix. In *International Colloquium on Automata, Languages, and Programming*, pages 1051–1062. Springer, 2014.

David P Woodruff. Sketching as a tool for numerical linear algebra. *Foundations and Trends® in Theoretical Computer Science*, 10(1–2):1–157, 2014.