Randomized Algorithms for Matrix Computations

ACM 204 / Caltech / Winter 2020

Prof. Joel A. Tropp

Notes prepared by Dr. Richard Kueng et al.

Typeset on April 21, 2021

Cite as:
Joel A. Tropp, *ACM 204: Randomized Algorithms for Matrix Computations*, Caltech CMS Lecture Notes 2020-01, Pasadena, March 2020.

Available from
https://resolver.caltech.edu/CaltechAUTHORS:20210421-101607288

Contents

Preface

Randomized algorithms date to the earliest days of numerical computation when Monte Carlo methods were developed to estimate complicated integrals. If you open a textbook on matrix computations, however, you will see that randomization historically played a marginal role in numerical linear algebra. In the last two decades, this situation has changed dramatically, and randomized algorithms have taken their place as a central part of numerical linear algebra. The goal of this course is to introduce you to some of the most effective techniques in this emerging area.

Some history

Beginning in the early 1980s, researchers began to appreciate that randomized algorithms can lead to simple, powerful, and provable algorithms for core linear problems. One of the first examples appears in Dixon's paper [Dix83], which demonstrates that we can estimate the largest eigenvalue of a symmetric matrix by initializing the power method with a random vector. This result is valid even when the matrix has no spectral gap; in contrast, the classical analysis of the power method fails completely in this setting. Later, Kuczyński & Woźniakowski [KW92] showed that randomized Lanczos methods can achieve a similar feat.

In the late 1980s, Girard [Gir89] proposed an efficient randomized algorithm for estimating the trace of a matrix that can only be accessed via matrix–vector multiplication. Hutchinson [Hut90] proposed a variant of Girard's method that has been influential. Golub & Meurant [GM10] showed how to use these techniques to estimate matrix trace functions, a problem with a wide range of applications.

In the late 1990s, several theoretical computer scientists proposed randomized algorithms for low-rank matrix approximation [FKV04; Pap+00]. Algorithms researchers quickly realized that other linear algebra computations might benefit from randomization. The papers [DKM06a; DKM06b; DKM06c] contain a systematic treatment of approximate matrix multiplication and low-rank matrix approximation. The paper [Sar06] studies randomized methods for least-squares problems. These early

efforts have been intellectually influential.

Around the same time, numerical analysts began to design more effective randomized algorithms for low-rank matrix approximation [MRT06] and least-squares computations [RT08]. This work led to the development of robust randomized algorithms for SVD and CUR decomposition [HMT11]. It has also led to efficient techniques for solving overdetermined least-squares problems [AMT10].

Over the last decade, randomized algorithms for matrix computation have found wide use in scientific computing, machine learning, and other settings. This field is currently quite active and there remain many opportunities for further advances.

This course

ACM 204 is a graduate course on randomized algorithms for matrix computations. It was taught for the first time in Winter 2020.

The course begins with Monte Carlo algorithms for trace estimation. This is a relatively simple setting that allows us to explore how randomness can be used for matrix computations. We continue with a discussion of the randomized power method and the Lanczos method for estimating the largest eigenvalue of a symmetric matrix. For these algorithms, the randomized starting point regularizes the trajectory of the iterations. The Lanczos iteration and randomized trace estimation fuse together in the stochastic Lanczos quadrature method for estimating the trace of a matrix function.

Then we turn to Monte Carlo sampling methods for matrix approximation. This approach is justified by the matrix Bernstein inequality, a powerful tool for matrix approximation. As a simple example, we develop sampling methods for approximate matrix multiplication.

In the next part of the course, we study random linear embeddings. These are random matrices that can reduce the dimension of a dataset while approximately preserving its geometry. First, we treat Gaussian embeddings in detail, and then we discuss structured embeddings that can be implemented using fewer computational resources. Afterward, we describe several ways to use random embeddings to solve over-determined least-squares problems.

We continue with a detailed treatment of the randomized SVD algorithm, the most widely used technique from this area. We give a complete *a priori* analysis with detailed error bounds. Then we show how to modify this algorithm for the streaming setting, where the matrix is presented as a sequence of linear updates. Last, we show how to develop an effective algorithm for selecting influential columns and rows from a matrix to obtain skeleton or CUR factorizations.

The next section of the course studies kernel matrices that arise in high-dimensional data analysis. We discuss positive-definite kernels and outline the computational issues associated with solving linear algebra problems involving kernels. We introduce random feature approximations and Nyström approximations based on randomized sampling. This area is still not fully developed.

The last part of the course gives a complete presentation of the sparse Cholesky algorithm of Kyng & Sachdeva [KS16], including a full proof of correctness.

These notes

The Winter 2020 edition of ACM 204 is the first instantiation of a course on randomized matrix computations. At a high level, the course is organized along the same lines as the survey [MT20]. In contrast to the survey, the course contains full proofs of the major results. Some parts of the class are modeled on the short course [Tro19]; the material on sparse Cholesky has been omitted from these notes because it has been carefully documented in the existing notes.

The course notes were transcribed from the lectures by the students as part of their coursework, so they reflect the actual content of the course. The notes have been closely edited by Dr. Richard Kueng, with additional light editing by the instructor.

There is no warranty about the correctness of these notes. Furthermore, material is not necessarily accompanied by appropriate citations to the literature.

Prerequisites

Prerequisites for this course are linear analysis (ACM 107), mathematical probability (ACM 117), and, ideally, some familiarity with numerical linear algebra (ACM 106). Nevertheless, this class is going to be largely self-contained. Experience with high-dimensional probability (ACM 217) is useful, but that class has not been taught for a while. Moreover, students are expected to do some programming in the exercises.

Additional references

This presentation of the course material is drawn primarily from papers by the instructor:

- [HMT11] Halko et al., "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions," 2008–2011.
- [Tro12] Tropp, "User-friendly tail bounds for sums of random matrices," 2010–2012.
- [Tro15] Tropp, *An introduction to matrix concentration inequalities*, 2012–2015.
- [Tro19] Tropp, *Matrix concentration and computational linear algebra*, 2019.
- [MT20] Martinsson & Tropp, "Randomized numerical linear algebra: Foundations and algorithms," 2019–2020.

Most of the ideas in this course have a long history. The papers cited above and the individual lectures (sometimes) provide more detailed background information.

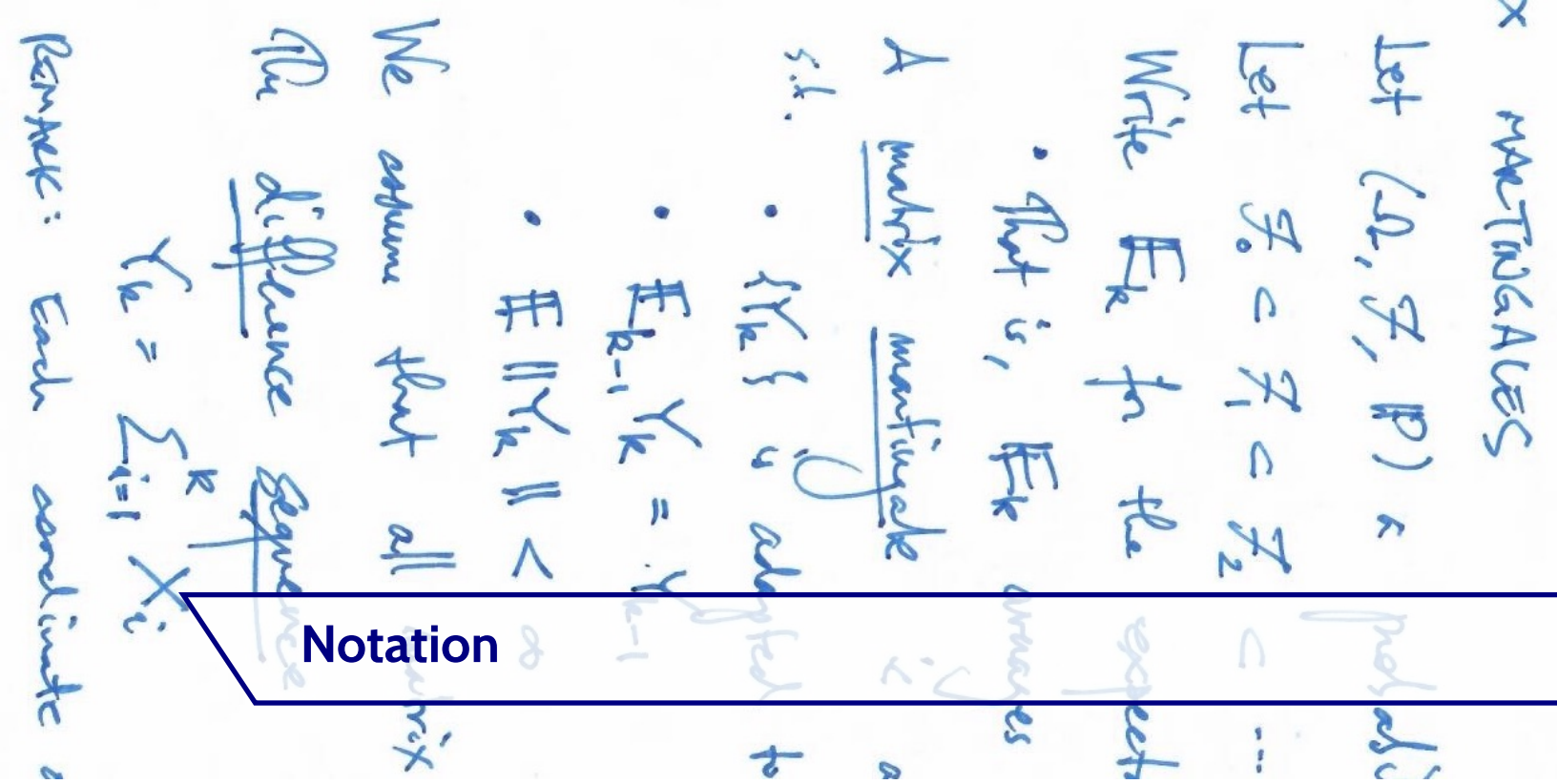
Scribes

These notes were prepared with the assistance of students and postdocs who participated in the course: Dmitry Burov, Po-Chih Chen, Yifan Chen, Nikola Kovachki, Riley Murray, Richard Kueng, Zongyi Li, Chung-Yi Lin, Fariborz Salehi, Jiace Sun, Oguzhan Teke, Jing Yu, Shumao Zhang, and Ziyun Zhang. Richard Kueng prepared the complete set of notes from the individual lectures. Many thanks are due to them for their care and diligence. All remaining errors are the fault of the instructor.

Joel A. Tropp
Steele Family Professor of Applied & Computational Mathematics
California Institute of Technology

jtropp@cms.caltech.edu
http://users.cms.caltech.edu/~jtropp

Pasadena, California
March 2020



Notation

I have selected notation that is common in the linear algebra and probability literature. I have tried to been consistent in using the symbols that are presented below. There are some minor variations in different lectures, including the letter that indicates the dimension of a matrix and the indexing of sums.

Linear algebra

We work in a real or complex linear space. The letters d and n (and occasionally others) are used to denote the dimension of this space, which is always finite. For example, we write $\mathbb{R}^d$ or $\mathbb{C}^n$. We may write $\mathbb{F}$ to refer to either field, or we may omit the field entirely if it is not important.

We use the delta notation for standard basis vectors: $\boldsymbol{\delta}_i$ has a one in the ith coordinate and zeros elsewhere. The vector $\mathbf{1}$ has ones in each entry. The dimension of these vectors is determined by context.

The symbol * denotes the (conjugate) transpose of a vector or a matrix. We equip $\mathbb{F}^d$ with the standard inner product $\langle \boldsymbol{x}, \boldsymbol{y} \rangle := \boldsymbol{x}^* \boldsymbol{y}$. The inner product generates the Euclidean norm $\|\boldsymbol{x}\|^2 := \langle \boldsymbol{x}, \boldsymbol{x} \rangle$.

We write $\mathbb{H}_d(\mathbb{F})$ for the real-linear space of $d \times d$ self-adjoint matrices with entries in the field $\mathbb{F}$. Recall that a matrix is self-adjoint when $\boldsymbol{A} = \boldsymbol{A}^*$. The symbols $\mathbf{0}$ and $\mathbf{I}$ denote zero matrix and the identity matrix; their dimensions are determined by context or by an explicit subscript.

We equip the space $\mathbb{H}_d$ with the trace inner product $\langle \boldsymbol{X}, \boldsymbol{Y} \rangle := \operatorname{tr}(\boldsymbol{XY})$, which generates the Frobenius norm $\|\boldsymbol{X}\|_{\mathrm{F}}^2 := \langle \boldsymbol{X}, \boldsymbol{X} \rangle$. The map $\operatorname{tr}[\cdot]$ returns the trace of a square matrix; we instate the convention that nonlinear functions bind before the trace.

The spectral theorem states that every self-adjoint matrix $\boldsymbol{A} \in \mathbb{H}_n$ admits a *spectral resolution*:

spectral resolution

$$\boldsymbol{A} = \sum\nolimits_{i=1}^{m} \lambda_i \boldsymbol{P}_i \quad \text{where} \quad \sum\nolimits_{i=1}^{m} \boldsymbol{P}_i = \mathbf{I}_n \quad \text{and} \quad \boldsymbol{P}_i \boldsymbol{P}_j = \delta_{ij} \boldsymbol{P}_i.$$

Here, $\lambda_1, \dots, \lambda_m$ are the distinct (real) eigenvalues of $\boldsymbol{A}$. The range of the orthogonal projector $\boldsymbol{P}_i$ is the invariant subspace associated with λ_i. We have written δ_{ij} for the Kronecker delta.

The maps $\lambda_{\min}(\cdot)$ and $\lambda_{\max}(\cdot)$ return the minimum and maximum eigenvalues of a self-adjoint matrix. The ℓ_2 operator norm $\|\cdot\|$ of a self-adjoint matrix satisfies the relation

$$\|\boldsymbol{A}\| \coloneqq \max\left\{|\lambda_{\max}(\boldsymbol{A})|, |\lambda_{\min}(\boldsymbol{A})|\right\} \quad \text{for } \boldsymbol{A} \in \mathbb{H}_n.$$

A self-adjoint matrix is *positive semidefinite (psd)* if its eigenvalues are nonnegative; a self-adjoint matrix is *positive definite (pd)* if its eigenvalues are positive. The symbol $\preccurlyeq$ refers to the psd order: $\boldsymbol{A} \preccurlyeq \boldsymbol{H}$ if and only if $\boldsymbol{H} - \boldsymbol{A}$ is psd.

positive semidefinite (psd)
positive definite (pd)

We can define a *standard matrix function* for a self-adjoint matrix using the spectral resolution. For an interval $I \subseteq \mathbb{R}$ and for a function $f : I \to \mathbb{R}$,

standard matrix function

$$\boldsymbol{A} = \sum\nolimits_{i=1}^{m} \lambda_i \boldsymbol{P}_i \quad \text{implies} \quad f(\boldsymbol{A}) = \sum\nolimits_{i=1}^{m} f(\lambda_i) \boldsymbol{P}_i.$$

Implicitly, we assume that the eigenvalues of the matrix $\boldsymbol{A}$ lie within the domain I of the function f. When we apply a real function to a self-adjoint matrix, we are always referring to the associated standard matrix function. In particular, we often encounter powers, exponentials, and logarithms.

We write $\mathbb{M}_n(\mathbb{F})$ for the linear space of $n \times n$ matrices over the field $\mathbb{F}$. We also define the linear space $\mathbb{M}^{m\times n}(\mathbb{F})$ of $m \times n$ matrices over the field $\mathbb{F}$. We can extend the trace inner-product and Frobenius norm to this setting:

$$\langle \boldsymbol{B}, \boldsymbol{C} \rangle \coloneqq \operatorname{tr}(\boldsymbol{B}^*\boldsymbol{C}) \quad \text{and} \quad \|\boldsymbol{B}\|_{\mathrm{F}}^2 \coloneqq \langle \boldsymbol{B}, \boldsymbol{B} \rangle \quad \text{for } \boldsymbol{A}, \boldsymbol{B} \in \mathbb{M}^{m\times n}.$$

The symbol $\|\cdot\|$ always refers to the ℓ_2 operator norm.

A square matrix $\boldsymbol{Q} \in \mathbb{M}_n$ that satisfies $\boldsymbol{Q}^*\boldsymbol{Q} = \mathbf{I}_n$ is called *orthogonal / unitary* in the real / complex case. A tall, rectangular matrix $\boldsymbol{B} \in \mathbb{M}^{m\times n}$ with $n \le m$ that satisfies $\boldsymbol{B}^*\boldsymbol{B} = \mathbf{I}_n$ is called *orthonormal*; this terminology is common in the numerical literature. More generally, a rectangular matrix $\boldsymbol{B} \in \mathbb{M}^{m\times n}$ is called a *partial isometry* if $\boldsymbol{B}^*\boldsymbol{B}$ is an orthogonal projector.

orthogonal / unitary
orthonormal
partial isometry

We write span for the linear hull of a family of vectors. The operators range$(\cdot)$ and null$(\cdot)$ extract the range and null space of a matrix. The operator † extracts the pseudoinverse.

Probability

The map $\mathbb{P}\{\cdot\}$ returns the probability of an event. The operator $\mathbb{E}[\cdot]$ returns the expectation of a random variable taking values in a linear space. We only include the brackets when it is necessary for clarity, and we impose the convention that nonlinear functions bind before the expectation.

The symbol $\sim$ means "has the distribution." We abbreviate (statistically) *independent and identically distributed (iid)*. Named distributions, such as NORMAL and UNIFORM, are written with small capitals.

independent and identically distributed (iid)

We say that a random vector $\boldsymbol{x} \in \mathbb{F}^n$ is *centered* when $\mathbb{E}[\boldsymbol{x}] = \mathbf{0}$. A random vector is *isotropic* when $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^*] = \mathbf{I}_n$. A random vector that is both centered and isotropic is *standardized*.

centered
isotropic
standardized

An important property of the standard normal distribution, which we use heavily, is the fact that it is rotationally invariant. If $\boldsymbol{x} \sim \text{NORMAL}(\mathbf{0}, \mathbf{I})$, then $\boldsymbol{Q}\boldsymbol{x}$ is also standard normal for every fixed matrix $\boldsymbol{Q}$ that is orthogonal (in the real case) or unitary (in the complex case).

Order notation

We use the familiar order notation from computer science. The symbol $\Theta(\cdot)$ refers to asymptotic equality. The symbol $O(\cdot)$ refers to an asymptotic upper bound.

Numerics

To understand numerical methods, you must implement and test numerical methods. To do so, you need a testbed of matrices to evaluate the performance of the methods in different circumstances.

Numerics 0.1 (Bedrock). We are going to make some synthetic matrices that will serve as running examples. Write code that generates each of the following matrix classes for appropriate parameters. Each matrix is parameterized by the field ($\mathbb{F} = \mathbb{R}$ or $\mathbb{F} = \mathbb{C}$), the dimension n and, sometimes, a rank parameter R.

1 **(Laplacian).** A standard second-order discretization of the differential operator $Lu = -u''$ for $u \in C^2[0,1]$ with boundary conditions $u(0) = u(1) = 0$ on a grid with $h = 1/(n+1)$ yields a symmetric tridiagonal matrix $\boldsymbol{L} \in \mathbb{H}_n$ with diagonal entries equal to $2/h^2$ and super- and sub-diagonal entries equal to $-1/h^2$. Write code to construct $\boldsymbol{L}$ for any given value of n. (*) What are the eigenvalues of $\boldsymbol{L}$?

2 **(Inverse Laplacian).** Write code that solves $\boldsymbol{Lu} = \boldsymbol{f}$ by computing a Cholesky decomposition of $\boldsymbol{L}$ and performing triangular elimination. (You may use the Matlab `chol` and `backslash` commands, or equivalents.) (*) What are the eigenvalues of $\boldsymbol{L}^{-1}$?

3 **(RBF kernel).** Draw n random points $(\boldsymbol{x}_1, \dots, \boldsymbol{x}_n)$ uniformly from the unit cube $[0,1]^d$. For bandwidth $h > 0$, form the radial basis function kernel matrix

$$k_{ij} = \exp(-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2/2h) \quad \text{for } i, j = 1, \dots, n.$$

Note that the cost of explicitly forming $\boldsymbol{K}$ grows like $O(dn^2)$, so it gets expensive fast. You can also do the same thing for a real dataset, which is more interesting.

4 **(Low-rank + noise).** For an inverse SNR ξ, the matrix takes the form

$$\boldsymbol{A} = \operatorname{diag}(\underbrace{1, \dots, 1}_{R}, 0, \dots, 0) + \xi/(4n)\boldsymbol{G}\boldsymbol{G}^* \in \mathbb{H}_n.$$

The matrix $\boldsymbol{G} \in \mathbb{M}_n$ has iid standard normal entries. Three examples: `LowRankLowNoise` ($\xi = 0.005$), `LowRankMedNoise` ($\xi = 0.05$), `LowRankHiNoise` ($\xi = 0.5$).

5 **(Polynomial decay).** For a decay parameter $p > 0$, the matrix takes the form

$$\boldsymbol{A} = \operatorname{diag}(\underbrace{1, \dots, 1}_{R}, 2^{-p}, 3^{-p} \dots, (n-R+1)^{-p}) \in \mathbb{H}_n.$$

Examples: `PolyDecaySlow` ($p = 0.5$), `PolyDecayMed` ($p = 1$), `PolyDecayFast` ($p = 2$).

6 **(Exponential decay).** For a decay parameter $q > 0$, the matrix takes the form

$$\boldsymbol{A} = \operatorname{diag}(\underbrace{1, \dots, 1}_{R}, 10^{-q}, 10^{-2q}, \dots, 10^{-(n-R)q}) \in \mathbb{H}_n.$$

Examples: `ExpDecaySlow` ($q = 0.01$), `ExpDecayMed` ($q = 0.1$), `ExpDecayFast` ($q = 0.5$).

7 **(Matrix Libraries).** You can also find many example matrices, arising from a variety of applications at SparseSuite (https://sparse.tamu.edu), SNAP (https://snap.stanford.edu), the UCI ML Repository (https://archive.ics.uci.edu/ml/index.php), and StatLib (http://lib.stat.cmu.edu/datasets/). Have a look.

1. Trace Estimation by Sampling

Date: 7 January 2020 **Scribe:** Richard Kueng

We will begin the course with a discussion of one of the simplest problems in numerical linear algebra: computing the trace of a positive-semidefinite matrix. Although this problem may seem trivial, there are important situations where it is challenging to compute the trace explicitly. We will develop a randomized method for estimating the trace that is useful in these circumstances. This technique serves as a building block for more interesting calculations, including error estimation and estimating the trace of a spectral function.

Agenda:

1. Trace estimation
2. Monte Carlo estimates
3. Controlling the trace
4. *A priori* bounds
5. Universality
6. *A posteriori* validation

1.1 Trace estimation

Let $\boldsymbol{A} \in \mathbb{H}_n$ be a psd matrix. The goal of today's lecture is to develop randomized algorithms that can efficiently approximate the trace of the matrix:

psd = positive semidefinite

$$\operatorname{tr}(\boldsymbol{A}) := \sum\nolimits_{i=1}^{n} a_{ii}.$$

This problem deserves some justification. Why can't we just read off the diagonal entries? Sometimes, we should. But it is not always possible.

Suppose that we have access to the matrix $\boldsymbol{A}$ via matrix–vector multiplication: $\boldsymbol{u} \mapsto \boldsymbol{A}\boldsymbol{u}$. It is clear that we can compute the trace exactly by invoking the primitive n times to extract the diagonal of the matrix:

$$\operatorname{diag}(\boldsymbol{A}) \quad = \quad (\boldsymbol{\delta}_1^*(\boldsymbol{A}\boldsymbol{\delta}_1), \quad \boldsymbol{\delta}_2^*(\boldsymbol{A}\boldsymbol{\delta}_2), \quad \dots, \quad \boldsymbol{\delta}_n^*(\boldsymbol{A}\boldsymbol{\delta}_n)).$$

But, if we are willing to accept an approximation for the trace, we might hope to prevent this profligacy. Our aim is to apply the primitive as few times as possible.

In summary, here are our goals:

Computational Problem (Trace estimation). Given a psd matrix $\boldsymbol{A} \in \mathbb{H}_n$, produce an estimate for $\operatorname{tr}(\boldsymbol{A})$.

Computational Primitive (Matrix–vector multiplication). Assume that we can compute $\boldsymbol{u} \mapsto \boldsymbol{A}\boldsymbol{u}$ efficiently for an arbitrary vector $\boldsymbol{u}$.

1.1.1 Why matrix–vector multiplication?

At first, the matrix–vector multiplication primitive may seem mysterious. Nevertheless, it arises in many circumstances in computational mathematics. Here is one example.

Example 1.1 (Trace of the inverse). Suppose that we wish to compute $\operatorname{tr}(\boldsymbol{A}^{-1})$ for a pd matrix $\boldsymbol{A}$. If we can solve the linear system $\boldsymbol{A}\boldsymbol{u} = \boldsymbol{f}$ efficiently, then we can implicitly apply the primitive $\boldsymbol{f} \mapsto \boldsymbol{A}^{-1}\boldsymbol{f}$.

pd = positive definite

For instance, suppose that $\boldsymbol{A}$ is the solution map for a (discretized) differential operator with boundary data $\boldsymbol{f}$. We can often solve the differential equation $\boldsymbol{A}\boldsymbol{u} = \boldsymbol{f}$ just as easily as we can compute a diagonal entry of the inverse map $\boldsymbol{A}^{-1}$. This problem arises in electronic structure calculations. ∎

1.1.2 Applications

Trace estimation originally arose from problems in computational statistics. The algorithms we discuss today were invented in this context.

Application 1.2 (Smoothing splines). Girard [Gir89] considered the problem of performing cross-validation for smoothing splines. We can fit a smoothing spline to data by solving a structured linear system. To identify the best smoothing parameter, we need to determine how the number of degrees of freedom in the spline model varies with the smoothing parameter. The number of degrees of freedom coincides with the trace of a somewhat complicated matrix involving the data and the smoothing parameter. But we can efficiently apply this matrix to an arbitrary vector, just by fitting a spline. Girard designed a randomized trace estimation algorithm to exploit this fact. ∎

There are many modern applications of trace estimation. Examples include electronic structure calculations [Bai+98], seismic inversion [LAH11], PDE-constrained optimization [HCH12], and Gaussian process regression [Don+17]. This catalog is adapted from the paper [Fit+18].

In this course, we will see that randomized trace estimation provides an effective means for estimating the error in a linear algebra computation.

1.2 Monte Carlo trace estimation

In its simplest form, the *Monte Carlo method* is the application of random sampling to estimate integrals. This approach is attributed to Ulam and von Neumann, who invented it during their work on the Manhattan Project in the 1940s. Girard [Gir89] observed that it is possible to develop a Monte Carlo method for trace estimation.

Idea: Design an unbiased random estimate for the trace of a matrix.

It turns out that it is quite easy to implement this vision.

1.2.1 Random vectors

We commence with some important definitions.

Definition 1.3 (Centered, isotropic, standardized). A *centered* random vector $\boldsymbol{\omega} \in \mathbb{F}^n$ satisfies $\mathbb{E}\,\boldsymbol{\omega} = \mathbf{0}$. A random vector $\boldsymbol{\omega}$ is *isotropic* when $\mathbb{E}[\boldsymbol{\omega}\boldsymbol{\omega}^*] = \mathbf{I}_n$. If a random vector is both centered and isotropic, we say that it is *standardized*.

centered
isotropic
standardized

One basic example of a standardized random vector is the standard normal random vector $\boldsymbol{\omega} \sim \textsc{normal}(\mathbf{0}, \mathbf{I}_n)$, also known as a standard Gaussian. Another example is a *Rademacher* random vector $\boldsymbol{\omega} \sim \textsc{uniform}\{\pm 1\}^n$.

$\sim$ means "has distribution"
Rademacher

1.2.2 The randomized trace estimator

Let $\boldsymbol{A} \in \mathbb{H}_n(\mathbb{F})$ be a psd matrix. Draw an isotropic random vector $\boldsymbol{\omega} \in \mathbb{F}^n$, which we call a *test vector*. Form the scalar random variable

test vector

$$X = \boldsymbol{\omega}^*(\boldsymbol{A}\boldsymbol{\omega}) \quad \text{for isotropic } \boldsymbol{\omega}. \tag{1.1}$$

We quickly calculate the expectation of X using linearity and the cyclic property of the trace:

$$\mathbb{E}[X] = \mathbb{E}\,\mathrm{tr}(\boldsymbol{\omega}^*\boldsymbol{A}\boldsymbol{\omega}) = \mathbb{E}\,\mathrm{tr}(\boldsymbol{A}\boldsymbol{\omega}\boldsymbol{\omega}^*) = \mathrm{tr}\left(\boldsymbol{A}\,\mathbb{E}[\boldsymbol{\omega}\boldsymbol{\omega}^*]\right) = \mathrm{tr}(\boldsymbol{A}\mathbf{I}_n) = \mathrm{tr}(\boldsymbol{A}).$$

That is, the random variable X is an *unbiased estimator* for the trace of $\boldsymbol{A}$.

A single sample of X does not usually provide a satisfactory trace estimate because its variance may be large. To resolve this shortcoming, we can average iid copies of the simple estimator.

iid = independent and identically distributed

For a natural number s, the *Monte Carlo trace estimator* with s samples is

Monte Carlo trace estimator

$$\bar{X}_s = \frac{1}{s}\sum\nolimits_{i=1}^{s} X_i \quad \text{for iid } X_i \sim X. \tag{1.2}$$

Using independence, it is straightforward to check that the trace estimator obeys

$$\mathbb{E}[\bar{X}_s] = \mathrm{tr}(\boldsymbol{A}) \quad \text{and} \quad \mathrm{Var}[\bar{X}_s] = \frac{1}{s}\mathrm{Var}[X].$$

In words, the expectation is correct, and the variance decreases in proportion to the number s of samples.

The arithmetic cost of the Monte Carlo trace estimator (1.2) amounts to simulating s random test vectors, invoking the matrix–vector multiplication primitive s times, and computing s inner products at a cost of $O(sn)$. Storage requirements are $O(s + n)$.

The Monte Carlo trace estimator (1.1)–(1.2) should be viewed as the most fundamental method in all of randomized linear algebra.

Exercise 1.4 (Frobenius-norm estimates). Let $\boldsymbol{B} \in \mathbb{F}^{m\times n}$ be a rectangular matrix, equipped with the computational primitive $\boldsymbol{B} \mapsto \boldsymbol{B}\boldsymbol{u}$. Explain how to use the randomized trace estimator to construct an unbiased estimator for $\|\boldsymbol{B}\|_{\mathrm{F}}^2$, the squared Frobenius norm of $\boldsymbol{B}$. What is the variance of the estimator?

1.2.3 Controlling the variance

What is the variance $\mathrm{Var}[X]$ of an individual sample X? It depends on the distribution of the test vector.

Idea: Choose the distribution of the test vector to control the variance.

Let us work out some specific examples.

Example 1.5 (Gaussians). Suppose that the test vector $\boldsymbol{\omega} \sim \text{NORMAL}(\mathbf{0}, \mathbf{I}_n)$. To compute the variance of X, let $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^*$ be an eigenvalue decomposition of the matrix $\boldsymbol{A}$. Rotational invariance of the standard normal distribution implies

$$X = \boldsymbol{\omega}^*\boldsymbol{A}\boldsymbol{\omega} \sim \boldsymbol{\omega}^*\boldsymbol{\Lambda}\boldsymbol{\omega} = \sum\nolimits_{i=1}^{n} \lambda_i \omega_i^2,$$

where $\boldsymbol{\Lambda} = \operatorname{diag}(\lambda_1, \dots, \lambda_n)$ and $\boldsymbol{\omega} = (\omega_1, \dots, \omega_n)$. Using independence of the coordinates of $\boldsymbol{\omega}$, we find that

$$\begin{aligned} \operatorname{Var}[X] &= \operatorname{Var}\left[\sum\nolimits_{i=1}^{n} \lambda_i \omega_i^2\right] = \sum\nolimits_{i=1}^{n} \lambda_i^2 \operatorname{Var}[\omega_i^2] \\ &= 2\sum\nolimits_{i=1}^{n} \lambda_i^2 = 2\|\boldsymbol{A}\|_{\mathrm{F}}^2. \end{aligned} \tag{1.3}$$

We used the fact that a squared standard normal variable has variance two.

To appreciate what this means, make the further estimate

$$\operatorname{Var}[X] = 2\|\boldsymbol{A}\|_{\mathrm{F}}^2 \le 2\|\boldsymbol{A}\| \operatorname{tr}(\boldsymbol{A}). \tag{1.4}$$

This relation holds because $\boldsymbol{A}$ is psd, so its eigenvalues are nonnegative. The inequality (1.4) demonstrates that $\operatorname{Var}[X]$ is bounded in terms of $\operatorname{tr}(\boldsymbol{A})$. As a consequence, we may hope to estimate the trace on a relative scale. More on this later. ∎

Example 1.6 (Rademachers). Suppose that the test vector $\boldsymbol{\omega} \sim \text{UNIFORM}\{\pm 1\}^n$. An elementary computation reveals that

$$\operatorname{Var}[X] = 2\sum\nolimits_{i \neq j} |a_{ij}|^2 < 2\|\boldsymbol{A}\|_F^2.$$

This is strictly smaller than the variance for a Gaussian test vector. The trace estimator based on a Rademacher test vector is called the Hutchinson trace estimator [Hut90]. It has an optimality property (Problem 1.20), and it is particularly useful for strongly diagonally dominant matrices. ∎

Example 1.7 (Uniform on the complex sphere). Assume that the test vector is drawn uniformly from the *complex* sphere with radius $\sqrt{n}$:

$$\boldsymbol{\omega} \sim \text{UNIFORM}(\sqrt{n}\, \mathbb{S}^{n-1}(\mathbb{C})).$$

Computing the variance is considerably more involved. One can show that

$$\operatorname{Var}[X] = \frac{n}{n+1}\left[\|\boldsymbol{A}\|_{\mathrm{F}}^2 - \frac{1}{n}(\operatorname{tr}\boldsymbol{A})^2\right]. \tag{1.5}$$

We have reduced the variance by a factor of two as compared with Example 1.5 or Example 1.6, although we do need to use complex arithmetic.

It turns out that (1.5) is the minimum variance achievable by any Monte Carlo trace estimator of the form (1.1). See Problem 1.23. ∎

Exercise 1.8 (Hutchinson [Hut90]). Complete the calculation from Example 1.6.

Exercise 1.9 (Optimal measurement systems). A finite family $\mathscr{M} = \{\boldsymbol{u}_1, \dots, \boldsymbol{u}_m\} \subset \mathbb{S}^{n-1}(\mathbb{C})$ of *complex* unit vectors is an *optimal measurement system* if

optimal measurement system

$$\frac{1}{m}\sum\nolimits_{i=1}^{m} (\boldsymbol{u}_i^*\boldsymbol{M}\boldsymbol{u}_i)\boldsymbol{u}_i\boldsymbol{u}_i^* = \frac{1}{(n+1)n}\left[\boldsymbol{M} + \operatorname{tr}(\boldsymbol{M})\,\mathbf{I}_n\right] \quad \text{for all } \boldsymbol{M} \in \mathbb{H}_n(\mathbb{C}).$$

Suppose that we draw a random vector $\boldsymbol{u} \sim \text{UNIFORM}(\mathscr{M})$, and construct the test vector $\boldsymbol{\omega} = \sqrt{n}\,\boldsymbol{u}$.

Verify that $\boldsymbol{\omega}$ is isotropic. For psd $\boldsymbol{A} \in \mathbb{H}_n(\mathbb{C})$, check that the variance of the trace estimate $X = \boldsymbol{\omega}^* \boldsymbol{A} \boldsymbol{\omega}$ satisfies

$$\operatorname{Var}[X] = \frac{n}{n+1} \left[\|\boldsymbol{A}\|_{\mathrm{F}}^2 - \frac{1}{n} (\operatorname{tr} \boldsymbol{A})^2 \right]. \tag{1.6}$$

In other words, an optimal measurement system achieves the same bound as a uniformly random vector on the complex sphere.

Exercise 1.10 (Fitzsimons et al. [Fit+18]). Two unitary matrices $\boldsymbol{U}, \boldsymbol{V} \in \mathbb{M}_n$ are called *mutually unbiased bases* when the columns satisfy $|\langle \boldsymbol{u}_i, \boldsymbol{v}_j \rangle| = n^{-1/2}$ for all $i, j = 1, \dots, n$. A family $\{\boldsymbol{U}_1, \dots, \boldsymbol{U}_k\} \subset \mathbb{M}_n$ of unitary matrices is *mutually unbiased* when each pair is mutually unbiased.

mutually unbiased bases

Show that a family of $n + 1$ mutually unbiased bases in $\mathbb{C}^n$ composes an optimal measurement system (Exercise 1.9).

This exercise can be interpreted as a partial derandomization of the trace estimator. It takes only $O(\log n)$ random bits to determine a test vector from this optimal measurement system. In contrast, it takes $O(n)$ random bits to determine a Rademacher random vector.

Exercise 1.11 (Traceless matrices). Suppose that $\boldsymbol{M} \in \mathbb{H}_n$ has trace zero. Consider the Gaussian trace estimator $X = \boldsymbol{\omega}^* \boldsymbol{M} \boldsymbol{\omega}$ where $\boldsymbol{\omega} \sim \textsc{normal}(\mathbf{0}, \mathbf{I}_n)$. What is the expectation of X? What is the variance of X? How do they compare? Do you anticipate that this estimator can give a relative-error approximation of the trace?

1.3 *A priori* bounds

When we design a randomized algorithm for solving a linear algebra problem, we also want to develop some theoretical analysis that justifies deploying it. To obtain these results, we turn to the field of probability theory and, especially, the subfield of high-dimensional probability. In this section, we roll out some classic methods from probability to deduce *a priori* guarantees on the probability that randomized trace estimators succeed.

1.3.1 The intrinsic dimension and the stable rank

The behavior of randomized linear algebra algorithms often depends on the spectral profile of the input matrix; that is, the rate of decay of the eigenvalues in the psd case or the singular values in the general case. We can capture some information about spectral decay using continuous proxies for the rank.

Definition 1.12 (Intrinsic dimension, stable rank). Let $\boldsymbol{A} \in \mathbb{H}_n$ be a psd matrix. The *intrinsic dimension* is the quantity

intrinsic dimension

$$\operatorname{intdim}(\boldsymbol{A}) := \frac{\operatorname{tr}(\boldsymbol{A})}{\|\boldsymbol{A}\|}.$$

The intrinsic dimension of the zero matrix equals zero.

Let $\boldsymbol{B} \in \mathbb{F}^{m \times n}$ be a rectangular matrix. The *stable rank* is the quantity

stable rank

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

The stable rank of the zero matrix equals zero.

For a nonzero matrix $\boldsymbol{A}$, the intrinsic dimension satisfies the inequality $1 \le \operatorname{intdim}(\boldsymbol{A}) \le \operatorname{rank}(\boldsymbol{A})$. The right-hand inequality holds with equality when $\boldsymbol{A}$ is an orthogonal projector. This bound supports the intuition that the intrinsic dimension reflects the number of dimensions in which $\boldsymbol{A}$ is energetic.

For a rectangular matrix, the stable rank has a similar interpretion as the intrinsic dimension. When $\boldsymbol{B}$ is nonzero, its stable rank obeys $1 \le \operatorname{srank}(\boldsymbol{B}) \le \operatorname{rank}(\boldsymbol{B})$.

1.3.2 Nonasymptotic bounds

Now, the simplest probability bound for the Monte Carlo estimator (1.2) stems from Chebyshev's inequality:

$$\mathbb{P}\left\{|\bar{X}_s - \operatorname{tr}(\boldsymbol{A})| \ge t \cdot \operatorname{tr}(\boldsymbol{A})\right\} \le \frac{\operatorname{Var}[X]}{s\,(\operatorname{tr}\boldsymbol{A})^2\, t^2}. \tag{1.7}$$

A key advantage of this result is that it does not depend on the distribution of a sample X, except through its variance.

Example 1.13 (Gaussians). Let us instantiate the probability inequality (1.7) for the special case of a Gaussian test vector (Example 1.5). Inserting the bound (1.4), we arrive at

$$\mathbb{P}\left\{|\bar{X}_s - \operatorname{tr}(\boldsymbol{A})| \ge t \cdot \operatorname{tr}(\boldsymbol{A})\right\} \le \frac{2}{s\,\operatorname{intdim}(\boldsymbol{A})\, t^2}. \tag{1.8}$$

As the intrinsic dimension increases, so the does the probability that we can estimate the trace to within a fixed relative error. This is not surprising: When the intrinsic dimension is large, the trace estimator is averaging over a greater number of energetic dimensions. ▪

Although Chebyshev's inequality is appealing, it also delivers a rather weak bound. We can develop stronger exponential concentration inequalities using the Laplace transform method. Here is one such result.

Theorem 1.14 (Gaussian trace estimator [GTP18]). Let $\boldsymbol{A} \in \mathbb{H}_n(\mathbb{R})$ be a *real* psd matrix. The trace estimator (1.2) based on a standard normal test vector $\boldsymbol{\omega} \sim \textsc{normal}(\mathbf{0}, \mathbf{I}_n)$ obeys the following bounds. For $s \le n$,

$$\mathbb{P}\left\{\bar{X}_s \ge t \cdot \operatorname{tr}(\boldsymbol{A})\right\} \le \exp\left(-\tfrac{1}{2} s \operatorname{intdim}(\boldsymbol{A})(\sqrt{t} - 1)^2\right) \qquad \text{for } t > 1;$$

$$\mathbb{P}\left\{\bar{X}_s \le t \cdot \operatorname{tr}(\boldsymbol{A})\right\} \le \exp\left(-\tfrac{1}{4} s \operatorname{intdim}(\boldsymbol{A})(1 - t)^2\right) \qquad \text{for } t < 1.$$

In other words, the probability of suffering an unusually large or small estimate of the trace is exponentially small in the number s of samples. It also declines exponentially with $\operatorname{intdim}(\boldsymbol{A})$. Compare with the bound (1.8), where these quantities only enter linearly.

Proof sketch. Carefully estimate the moment generating function (mgf) of X using rotational invariance of the Gaussian distribution and the explicit form of the chi-square distribution with one degree of freedom. ■

Exercise 1.15 (Complex Gaussian trace estimator). Extend Theorem 1.14 to a *complex* psd matrix $\boldsymbol{A} \in \mathbb{H}_n(\mathbb{C})$. Each of the exponents features an additional factor of two, so the complex result is *better* than the real result. **Hint:** Using rotational invariance, we can reduce the analysis to the real case.

Problem 1.16 (Gratton and Titley-Peloquin [GTP18]). Prove Theorem 1.14.

1.3.3 Asymptotic bounds and universality

Although linear algebra algorithms are finite procedures, we can still gain some insight into their performance using asymptotic theory. In the case of the trace estimator, we learn that its large-sample behavior only depends on the variance of an individual sample, but not on any fine properties of the distribution. This is an instance of a phenomenon called *universality*. *universality*

Provided that the test vector $\boldsymbol{\omega}$ has one moment (which it must!), the strong law of large numbers (SLLN) implies that

$$\bar{X}_s \to \operatorname{tr}(\boldsymbol{A}) \quad \text{almost surely as } s \to \infty.$$

In words, if we take a sufficiently large number of samples, the trace estimator will tend to the correct answer. Statisticians call this *asymptotic consistency*. *asymptotic consistency*

Provided that the test vector $\boldsymbol{\omega}$ has two moments (which we can and should engineer!), the central limit theorem (CLT) implies that

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

In words, the fluctuations of the trace estimator around its mean roughly follow a Gaussian distribution with variance $\operatorname{Var}[X]/s$. If the sample size is large enough, we can use the Gaussian limit to obtain heuristic information about the behavior of the trace estimator. Statisticians call this *asymptotic normality*. *asymptotic normality*

Warning 1.17 (The curse of Monte Carlo). To achieve a relative error of $\varepsilon > 0$ in the trace estimate $\bar{X}_s$, the number s of samples must obey $s \geq \varepsilon^{-2} \operatorname{Var}[X]$. This scaling is unavoidable because of the central limit theorem. Since ε^{-2} grows rather quickly as $\varepsilon \to 0$, we cannot hope to achieve a very small relative error using a Monte Carlo trace estimator.

The same phenomenon plagues most Monte Carlo sampling estimators. For this reason, numerical analysts have historically been averse to using Monte Carlo methods, except as a final resort. Nevertheless, there are some applications where we are willing to accept low accuracy in exchange for an inexpensive computation. One must contemplate whether this tradeoff is acceptable.

For Monte Carlo methods to be practical, they often need to be coupled with variance reduction techniques. This can be achieved for trace estimation by means of low-rank matrix approximation [GSO17]. ▪

1.4 *A posteriori* validation and the role of statistics

Although *a priori* analysis gives us confidence that the Monte Carlo trace estimator will produce a relative-error approximation of the trace, the theory has limited relevance for practical computation.

First of all, we cannot activate the error bounds for particular matrices because we do not have access to the information required. For instance, we rarely know the intrinsic dimension of the input matrix because it depends explicitly on the trace!

Second, the trace estimator is stochastic. It has a sampling distribution and a probability of failure. We know that it will behave well on average, but we still need to validate its performance each time we use it.

Another way to think about these issues is that we are collecting (random) data about the trace of a matrix, and we wish to infer the actual value of trace and quantify the uncertainty in our estimate. We can achieve these goals using methods from statistics, which is the science of drawing inferences from data.

The role of statistics in randomized NLA

Statistical theory is an ideal match for randomized NLA. Indeed, statistical methods frame hypotheses about the random model that generates the data. Since we design the randomized NLA algorithm, we can be confident that it produces data that satisfies the assumptions required to invoke statistical methodologies.

Miles Lopes [Lop19] has proposed a sweeping program to design statistical methodology for validating randomized NLA algorithms. The approach in this section is inspired by his vision. At this stage, there remain many opportunities for further research.

1.4.1 The sample variance estimator

sample variance

First of all, we can use the data that we have collected to estimate the variance of the trace estimator. To do so, we simply compute the *sample variance* of the individual samples:

$$V_s := \frac{1}{s-1} \sum_{i=1}^{s} (X_i - \bar{X}_s)^2.$$

The sample variance is an unbiased estimator for the variance of a single sample:

$$\mathbb{E}\, V_s = \mathrm{Var}[\bar{X}].$$

Thus, we can approximate the variance of the trace estimator as $\mathrm{Var}[\bar{X}_s] \approx V_s/s$. This simple computation already gives us a much clearer sense about the scale of the errors in our trace computation.

Exercise 1.18 (Schatten 4-norm estimation). Let $\boldsymbol{B} \in \mathbb{F}^{m\times n}$ be a rectangular matrix, equipped with the primitive $\boldsymbol{u} \mapsto \boldsymbol{B}\boldsymbol{u}$. Explain how to use the sample variance of the trace estimator to approximate the Schatten 4-norm of $\boldsymbol{B}$.

1.4.2 Confidence intervals for the trace

confidence interval

To form a clearer picture about the uncertainty in our trace estimate, we can build a (symmetric, Student's t) *confidence interval* for the trace. For a level $\alpha \in (0, 0.5)$,

$$\mathrm{tr}(\boldsymbol{A}) \in \bar{X}_s \pm t_{\alpha,s-1}\sqrt{V_s} \quad \text{with probability about } 1 - 2\alpha.$$

$a \pm e$ denotes the interval $[a - e, a + e]$

We have written $t_{\alpha,s-1}$ for the α quantile of a Student's t-distribution with $s-1$ degrees of freedom. The probability here is over the random choices in the algorithm. To ensure that the confidence interval has the correct coverage, one rule of thumb is to take the number of samples $s \geq 30$ when the level $\alpha \geq 0.025$.

When the number s of samples is very large, it is common to use a normal confidence interval instead. In this case, we replace $t_{\alpha,s-1}$ by the α quantile of a standard normal variable. With the statistical facilities in modern programming languages, the normal approximation is not really necessary.

The construction of the confidence interval is inspired by the asymptotic normality of the Monte Carlo trace estimate. Nevertheless, the confidence interval is only approximate, and it can be misleading or inaccurate.

1.4.3 Bootstrap confidence intervals

the bootstrap

To build better confidence intervals, we recommend using data-driven methods. This section summarizes a procedure, called *the bootstrap*, which resamples the trace data to obtain a proxy for the sampling distribution of the trace estimator. This distribution can be used to perform more reliable inference about the actual value of the trace.

Here is the procedure. Let $\alpha \in (0, 0.5)$ be the level of coverage, and let B be the number of bootstrap replicates.

1 Draw a random sample $\mathfrak{X} = (X_1, \dots, X_s)$ of iid trace samples.
2 Form the Monte Carlo trace estimate $\bar{X}_s$.
3 For each bootstrap replicate $b = 1, \dots, B$:
    1 Draw a sample $X_1^\circ, \dots, X_s^\circ$ uniformly from $\mathfrak{X}$ with replacement.
    2 Compute the replicate Monte Carlo trace estimate $\bar{X}_s^\circ$.
    3 Form the replicate error estimate $e_b = \bar{X}_s^\circ - \bar{X}_s$.
4 Compute the quantiles q_α and $q_{1-\alpha}$ of the errors $(e_1, \dots, e_B)$.
5 Report the bootstrap confidence interval:

$$\operatorname{tr}(\boldsymbol{A}) \in [\bar{X}_s + q_\alpha, \bar{X}_s + q_{1-\alpha}] \quad \text{with probability about } 1 - 2\alpha.$$

The rough idea behind the bootstrap method is that each replicate $\bar{X}_s^\circ$ of the trace estimator can be viewed as a new draw from the distribution of the trace estimator. The errors $e_b = \bar{X}_s^\circ - \bar{X}_s$ serve as approximations for the fluctuations of the distribution around its mean. See [ET93] for background and theory about the bootstrap.

Each trace sample may be expensive, but the bootstrap replicates are very cheap. A simple rule of thumb is to take the number of trace samples $s \geq 30$ when the level $\alpha \geq 0.025$. With modern computational facilities, there is no need to limit the number of bootstrap replicates. For a simple problem like trace estimation, we may choose B in the range $B = 10^3$ up to $B = 10^6$. Using more replicates leads to better estimates of the sampling distribution of the trace estimator, limited only by the fidelity of the original sample.

Algorithm 1.1 contains a summary of the Monte Carlo trace estimation procedure, equipped with a bootstrap confidence interval. We anticipate that this procedure gives reliable results for relatively small samples.

Remark 1.19 (Bootstrap and NLA). The bootstrap was invented by Brad Efron in the late 1970s [Efr79a; Efr79b]. It is one of the earliest methods of computational statistics. Miles Lopes [Lop19] recognized the opportunity to use the bootstrap to validate randomized NLA calculations.

Efron was a mathematics undergraduate at Caltech in the late 1950s!

Problems

Problem 1.20 (Hutchinson [Hut90]). Let $\boldsymbol{A}$ be psd. Among all isotropic distributions for $\boldsymbol{\omega}$ where the coordinates are independent, the Rademacher distribution minimizes $\operatorname{Var}[X]$ when X has the form (1.1).

Problem 1.21 (The real sphere). Let $\boldsymbol{\omega} \sim \textsc{uniform}(\sqrt{n}\, \mathbb{S}^{n-1}(\mathbb{R}))$. Compute the variance $\operatorname{Var}[X]$ of the trace estimator $X = \boldsymbol{\omega}^* \boldsymbol{A} \boldsymbol{\omega}$. **Hint:** Have a look at the paper [Folo1].

Problem 1.22 (The complex sphere). Complete the calculation from Example 1.7. One approach is to verify that

$$\int_{\mathbb{S}^{n-1}(\mathbb{C})} (\boldsymbol{u}^* \boldsymbol{M} \boldsymbol{u})\, \boldsymbol{u}\boldsymbol{u}^*\, \mathrm{d}\boldsymbol{u} = \frac{1}{(n+1)n} [\boldsymbol{M} + \operatorname{tr}(\boldsymbol{M})\, \mathbf{I}] \quad \text{for all } \boldsymbol{M} \in \mathbb{H}_n(\mathbb{C}).$$

where $\mathrm{d}\boldsymbol{u}$ is the Haar distribution on the complex unit sphere. Can you compute the centered fourth moment $\mathbb{E}[(X - \mathbb{E}\, X)^4]$ of the trace estimator for a uniformly random vector on the complex sphere? **Hint:** Look up *Haar integration*; this method allows for quick computation of many challenging integrals.

Problem 1.23 (Richard Kueng). Define the *set of states* in n complex dimensions:

set of states

$$\mathsf{S}_n := \{\boldsymbol{A} \in \mathbb{H}_n(\mathbb{C}) : \operatorname{tr}(\boldsymbol{A}) = 1 \text{ and } \boldsymbol{A} \text{ is psd}\}.$$

Algorithm 1.1 TraceEstimator by Monte Carlo sampling, with a bootstrap confidence interval

Input: Psd input matrix $\boldsymbol{A} \in \mathbb{H}_n$, number s of samples, level α of confidence, number B of bootstrap replicates
Output: Level α confidence interval $[\bar{X}_s + q_\alpha, \bar{X}_s + q_{1-\alpha}]$ for $\operatorname{tr}(\boldsymbol{A})$

function TraceEstimator($\boldsymbol{A}$; s; α; B)
  for $i = 1, \dots, s$ **do** ▷ Initial trace estimate
    Draw isotropic random test vector $\boldsymbol{\omega}_i \in \mathbb{F}^n$
    Form sample $X_i = \boldsymbol{\omega}_i^*(\boldsymbol{A}\boldsymbol{\omega}_i)$
  Aggregate sample $\mathfrak{X} = (X_1, \dots, X_s)$
  Initial trace estimate $\bar{X}_s = s^{-1} \sum_{i=1}^s X_i$

  for $b = 1, \dots, B$ **do** ▷ Bootstrap confidence interval
    Draw $X_1^\circ, \dots, X_s^\circ$ uniformly from $\mathfrak{X}$ with replacement
    Compute replicate trace estimate $\bar{X}_s^\circ = s^{-1} \sum_{i=1}^s X_i^\circ$
    Compute replicate error $e_b = \bar{X}_s^\circ - \bar{X}_s$

  Find quantiles q_α and $q_{1-\alpha}$ of errors $(e_1, \dots, e_B)$
  Return interval $[\bar{X}_s + q_\alpha, \bar{X}_s + q_{1-\alpha}]$

Prove that, among all isotropic distributions for $\boldsymbol{\omega} \in \mathbb{C}^n$,

$$\min_{\boldsymbol{\omega}} \max_{\boldsymbol{A} \in \mathsf{S}_n} \operatorname{Var}[\boldsymbol{\omega}^* \boldsymbol{A} \boldsymbol{\omega}] = \frac{n}{n+1} \left[\|\boldsymbol{A}\|_{\mathrm{F}}^2 - \frac{1}{n} (\operatorname{tr} \boldsymbol{A})^2 \right].$$

Hint: Look up *frame potential*.

Problem 1.24 (Exponential bounds). Suppose that the random test vector $\boldsymbol{\omega}$ has a subgaussian distribution. Use the Hanson–Wright inequality [Ver18, Thm. 6.2.1] to develop an exponential concentration inequality for the trace estimator (1.2). Compare and contrast the result with Theorem 1.14.

Problem 1.25 (Mutually unbiased bases). For each odd prime p, construct a family of $p+1$ mutually unbiased bases in $\mathbb{M}_p$. The following relations from number theory may be helpful. For each primitive pth root of unity $\zeta = \mathrm{e}^{\mathrm{i}2\pi r/p}$, we have the sum formulas

1. $\sum_{j=0}^{p-1} \zeta^{jk} = p\delta_{k0}$ for each integer k, where $\delta_{\cdot\cdot}$ is the Kronecker delta.
2. $|\sum_{j=0}^{p-1} \zeta^{kj^2+\ell j}|^2 = p$ for each integer ℓ and nonzero integer k.

More hints: Look up *Alltop sequence* and *Weyl–Heisenberg group*.

Numerics 1.26 (Amazing Trace). In this problem, we will explore the behavior of random trace estimators, as applied to the inverse Laplacian matrix.

1. Implement the randomized trace estimator $\bar{X}_s$. We will consider test vectors that are (i) drawn uniformly from $\{\boldsymbol{\delta}_1, \dots, \boldsymbol{\delta}_n\}$, (ii) real standard normal, (iii) Rademacher, and (iv) uniform on the complex unit sphere.
2. Fix $n = 1000$. Compute the trace of the inverse Laplacian matrix, either analytically or numerically.
3. For each type of test vector, plot the sampling distribution of the trace estimator $\bar{X}_s$ as a function of the number s of samples; compare with the exact trace. Plot

the sampling distribution of the sample variance as a function of s. How do the four types of test vectors compare?

4 Implement the bootstrap procedure for obtaining a data-driven confidence interval. For a fixed number s of samples, say $s = 30$, compute level-α symmetric confidence intervals for $\alpha = 0.025$ for each of 1000 realizations of the trace estimator. In how many realizations does the confidence interval cover the exact trace? How much do the confidence intervals vary in their center and their length?

Lecture bibliography

[Bai+98] Z. Bai et al. *Computing Partial Eigenvalue Sum in Electronic Structure Calculations*. Technical report. Stanford University, 1998. URL: https://web.cs.ucdavis.edu/~bai/publications/baifaheygolubetal98.pdf.

[Don+17] K. Dong et al. "Scalable Log Determinants for Gaussian Process Kernel Learning". In: *Advances in Neural Information Processing Systems 30*. Edited by I. Guyon et al. Curran Associates, Inc., 2017, pages 6327–6337. URL: http://papers.nips.cc/paper/7212-scalable-log-determinants-for-gaussian-process-kernel-learning.pdf.

[Efr79a] B. Efron. "Bootstrap methods: another look at the jackknife". In: *Ann. Statist.* 7.1 (1979), pages 1–26. ISSN: 0090-5364.

[Efr79b] B. Efron. "Computers and the theory of statistics: thinking the unthinkable". In: *SIAM Rev.* 21.4 (1979), pages 460–480. DOI: 10.1137/1021092.

[ET93] B. Efron and R. J. Tibshirani. *An introduction to the bootstrap*. Volume 57. Monographs on Statistics and Applied Probability. Chapman and Hall, New York, 1993, pages xvi+436. DOI: 10.1007/978-1-4899-4541-9.

[Fit+18] J. K. Fitzsimons et al. "Improved stochastic trace estimators using mutually unbiased bases". In: *Uncertainty in Artificial Intelligence: Proceedings of the Thirty-Fourth Conference*. Edited by A. Globerson and R. Silva. Monterey, CA: AUAI Press, 2018.

[Fol01] G. B. Folland. "How to integrate a polynomial over a sphere". In: *Amer. Math. Monthly* 108.5 (2001), pages 446–448. DOI: 10.2307/2695802.

[GSO17] A. S. Gambhir, A. Stathopoulos, and K. Orginos. "Deflation as a method of variance reduction for estimating the trace of a matrix inverse". In: *SIAM J. Sci. Comput.* 39.2 (2017), A532–A558. DOI: 10.1137/16M1066361.

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

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

[HCH12] E. Haber, M. Chung, and F. Herrmann. "An effective method for parameter estimation with PDE constraints with multiple right-hand sides". In: *SIAM J. Optim.* 22.3 (2012), pages 739–757. DOI: 10.1137/11081126X.

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

[LAH11] T. van Leeuwen, A. Y. Aravkin, and F. Herrmann. "Seismic waveform inversion by stochastic optimization". In: *Intl. J. Geophysics* (2011). Article ID 689041. DOI: 10.1155/2011/689041.

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

[Ver18] R. Vershynin. *High-dimensional probability*. Volume 47. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, Cambridge, 2018, pages xiv+284. DOI: 10.1017/9781108231596.

2. Maximum eigenvalue

Date: 9 January 2020 **Scribe:** Richard Kueng

Agenda:

1. Maximum eigenvalue
2. Sampling estimator?
3. Power method
4. Classical analysis
5. Randomized power method
6. Gap-free analysis
7. Randomized Krylov

In Lecture 1, we considered sampling methods for estimating the trace, where each sample is independent and involves one matrix–vector product. Today, we will develop randomized algorithms for estimating the maximum eigenvalue of a psd matrix. To accomplish this goal, we introduce iterative algorithms that proceed from a random starting vector. In other words, we will apply the matrix *sequentially*. The iteration is necessary to obtain accurate estimates of the maximum eigenvalue, and the randomness helps us derive refined bounds for the performance of algorithms.

2.1 Estimating the maximum eigenvalue

Let $\boldsymbol{A} \in \mathbb{H}_n(\mathbb{R})$ be a *real* psd matrix. We will explore the complex case in the exercises; it is similar in spirit—but even more interesting. Write $\lambda_1 \geq \cdots \geq \lambda_n \geq 0$ for the eigenvalues of $\boldsymbol{A}$, arranged in weakly decreasing order. Our goal is to estimate λ_1 efficiently.

As always in NLA, our ability to access $\boldsymbol{A}$ affects the kinds of algorithms we can design. Parallel with Lecture 1, we interact with $\boldsymbol{A}$ via matrix–multiplication: we can compute $\boldsymbol{u} \mapsto \boldsymbol{A}\boldsymbol{u}$ for any vector $\boldsymbol{u} \in \mathbb{R}^n$. The goal is to limit the number of times we invoke the primitive.

When $\boldsymbol{A}$ is a general psd matrix, presented as an array, the cost of the matrix–multiplication is $O(n^2)$. When $\boldsymbol{A}$ is sparse, the cost of the primitive may be much lower. There are other circumstances (e.g., where $\boldsymbol{A}$ is the solution operator of a differential equation) where we can apply $\boldsymbol{A}$ quickly because of its structure. Another important property of matrix–multiplication is that it is *non-invasive*: algorithms built on this primitive do not modify the matrix $\boldsymbol{A}$.

In summary, here is the problem setup:

Computational Problem (Maximum eigenvalue). Let $\boldsymbol{A} \in \mathbb{H}_n(\mathbb{R})$ be a real psd matrix. Compute $\lambda_{\max}(\boldsymbol{A})$.

Computational Primitive (Matrix–vector multiplication). We can form $\boldsymbol{u} \mapsto \boldsymbol{A}\boldsymbol{u}$ for any vector $\boldsymbol{u} \in \mathbb{R}^n$.

Remark 2.1 (Maximum eigenvalues by linear sampling?). One might initially hope to estimate λ_1 from linear data of the form $(\boldsymbol{A}\boldsymbol{u}_1, \dots, \boldsymbol{A}\boldsymbol{u}_s)$, where the vectors $\boldsymbol{u}_i$ do not depend on the matrix $\boldsymbol{A}$. Unfortunately, it is necessary to take $s \gtrsim \text{const} \cdot n$ samples to accomplish this goal. More precisely, for the worst-case matrix, the number s of samples must be linear in the dimension n to estimate the maximum eigenvalue up to a constant factor with high probability [Woo14, Chap. 6]. This fact rules out the prospect of an efficient linear sampling estimator for the maximum eigenvalue.

2.2 The power method (PM)

The *power method (PM)* attempts to compute λ_1 by means of a dynamical system, i.e., an iterative procedure. *power method (PM)*

Idea: Use iteration to refine the eigenvalue estimate.

2.2.1 Procedure

Choose an initial vector $\boldsymbol{y}_0 = \boldsymbol{\omega} \in \mathbb{R}^n$, and iteratively construct

$$\boldsymbol{y}_k = \frac{\boldsymbol{A}\boldsymbol{y}_{k-1}}{\|\boldsymbol{A}\boldsymbol{y}_{k-1}\|} \quad \text{for each } k = 1, 2, 3, \dots.$$

This is equivalent to defining

$$\boldsymbol{y}_k = \frac{\boldsymbol{A}^k\boldsymbol{\omega}}{\|\boldsymbol{A}^k\boldsymbol{\omega}\|} \quad \text{for each } k = 1, 2, 3, \dots.$$

Indeed, we can just postpone the normalization to the kth iteration.

The sequence $\{\boldsymbol{y}_k\}$ of approximate (maximum) eigenvectors induces a sequence of *maximum eigenvalue estimates*: *maximum eigenvalue estimates*

$$\xi_k = \boldsymbol{y}_k^*\boldsymbol{A}\boldsymbol{y}_k \in [0, \lambda_1].$$

The upper and lower bounds on ξ_k both follow from the Rayleigh variational principle. Define the *relative error* in the eigenvalue approximation: *relative error*

$$\operatorname{err}(\xi_k) := \frac{\lambda_1 - \xi_k}{\lambda_1} \in [0, 1]. \tag{2.1}$$

Our goal is to assess the number k of iterations required to drive the relative error below a threshold.

Exercise 2.2 (Minimum eigenvalues). Explain how one might use the PM to estimate the *minimum* eigenvalue of a symmetric matrix.

Exercise 2.3 (Maximum singular values). Explain how one might use the PM to estimate the maximum *singular* value of a rectangular matrix.

2.2.2 Intuition

Each iterate $\boldsymbol{y}_k$ involves the kth power of the matrix $\boldsymbol{A}$. To appreciate why the PM might work, introduce the spectral resolution to see that

$$\boldsymbol{A} = \sum\nolimits_j \lambda_j \boldsymbol{P}_j \quad \text{implies} \quad \boldsymbol{A}^k = \sum\nolimits_j \lambda_j^k \boldsymbol{P}_j.$$

In other words, powering dramatically amplifies the large eigenvalues, relative to the small eigenvalues. Thus, we anticipate that $\boldsymbol{y}_k = \boldsymbol{A}^k\boldsymbol{\omega}/\|\boldsymbol{A}^k\boldsymbol{\omega}\|$ will align more and more with the invariant subspaces associated with the largest eigenvalues.

Nevertheless, even when the relative error is very small, the eigenvector estimate $\boldsymbol{y}_k$ may not be close to the invariant subspace range$(\boldsymbol{P}_1)$ associated with the largest eigenvalue λ_1. But we can say something:

Exercise 2.4 (Approximate eigenvectors). Suppose that $\operatorname{err}(\xi_k) \le \varepsilon$. Prove that the approximate eigenvector $\boldsymbol{y}_k$ has a large component aligned with the eigenvectors of $\boldsymbol{A}$ with large eigenvalues. More precisely,

$$\|\boldsymbol{P}_{>\lambda}\boldsymbol{y}_k\|^2 \ge 1 - \frac{\varepsilon\lambda_1}{\lambda_1 - \lambda} \quad \text{for } 0 \le \lambda < \lambda_1.$$

We have written $\boldsymbol{P}_{>\lambda}$ for the spectral projector onto the invariant subspace associated with the eigenvalues of $\boldsymbol{A}$ that strictly exceed λ.

2.2.3 Simplifying assumptions

To make the exposition more transparent, we will frame two inessential hypotheses.

1. **Diagonal matrix:** Change coordinates so that $\boldsymbol{A} = \operatorname{diag}(\lambda_1, \dots, \lambda_n)$.
2. **Normalization:** Rescale the matrix so that $\lambda_1 = 1$.

The first assumption is just a matter of working in the orthonormal basis where $\boldsymbol{A}$ is diagonal. The PM algorithm, however, does not know this distinguished basis, and we cannot use knowledge of the basis to construct the starting vector $\boldsymbol{\omega}$. The second assumption does not affect the analysis because the relative error (2.1) is scale-invariant.

2.2.4 Representation of the relative error

To study the behavior of the PM, we begin with an explicit representation of the relative error (2.1). Since the maximum eigenvalue $\lambda_1 = 1$,

$$\begin{aligned}\operatorname{err}(\xi_k) = 1 - \boldsymbol{y}_k^*\boldsymbol{A}\boldsymbol{y}_k &= 1 - \frac{\boldsymbol{\omega}^*\boldsymbol{A}^{2k+1}\boldsymbol{\omega}}{\boldsymbol{\omega}^*\boldsymbol{A}^{2k}\boldsymbol{\omega}} \\ &= \frac{\boldsymbol{\omega}^*\boldsymbol{A}^{2k}\boldsymbol{\omega}}{\boldsymbol{\omega}^*\boldsymbol{A}^{2k}\boldsymbol{\omega}} - \frac{\boldsymbol{\omega}^*\boldsymbol{A}^{2k+1}\boldsymbol{\omega}}{\boldsymbol{\omega}^*\boldsymbol{A}^{2k}\boldsymbol{\omega}} = \frac{\boldsymbol{\omega}^*\boldsymbol{A}^k\,(\mathbf{I} - \boldsymbol{A})\,\boldsymbol{A}^k\boldsymbol{\omega}}{\boldsymbol{\omega}^*\boldsymbol{A}^{2k}\boldsymbol{\omega}}.\end{aligned}$$

Next, use the representation $\boldsymbol{A} = \operatorname{diag}(\lambda_1, \dots, \lambda_n)$ to write out the expression in coordinates:

$$\operatorname{err}(\xi_k) = \frac{\sum_{i=1}^n \omega_i^2 \lambda_i^{2k}(1 - \lambda_i)}{\sum_{i=1}^n \omega_i^2 \lambda_i^{2k}} = \frac{\sum_{i>1} \omega_i^2 \lambda_i^{2k}(1 - \lambda_i)}{\omega_1^2 + \sum_{i>1} \omega_i^2 \lambda_i^{2k}}. \tag{2.2}$$

This expression shows how the eigenvalues and the components of the starting vector $\boldsymbol{\omega} = (\omega_1, \dots, \omega_n)^{\mathsf{T}} \in \mathbb{R}^n$ interact.

2.2.5 Classical analysis of PM

The standard analysis of the power method shows that the vector $\boldsymbol{y}_q$ converges to a maximum eigenvector of the matrix $\boldsymbol{A}$. The rate of convergence depends on the ratio between second and first eigenvalue.

Theorem 2.5 (Convergence of PM). Let $\boldsymbol{A} \in \mathbb{H}_n(\mathbb{R})$ be a *real* psd matrix whose eigenvalues obey $\lambda_1 > \lambda_2 > \lambda_3$ Assume that the first coordinate of the starting vector $\boldsymbol{\omega} \in \mathbb{R}^n$ satisfies $\omega_1 \neq 0$. Then the relative errors (2.1) in the eigenvalue estimates ξ_k produced by the PM have the limit

$$\frac{\operatorname{err}(\xi_{k+1})}{\operatorname{err}(\xi_k)} \to \left(\frac{\lambda_2}{\lambda_1}\right)^2 \quad \text{as } k \to \infty.$$

In particular, $\boldsymbol{y}_k$ converges to the span of the (unique) top eigenvector.

Proof. Use the decomposition (2.2) to rewrite the error ratio as

$$\begin{aligned}\frac{\operatorname{err}(\xi_{k+1})}{\operatorname{err}(\xi_k)} &= \frac{\sum_{i>1} \omega_i^2 \lambda_i^{2(k+1)}(1-\lambda_i)}{\omega_1^2 + \sum_{i>1} \omega_i^2 \lambda_i^{2(k+1)}} \cdot \frac{\omega_1^2 + \sum_{i>1} \omega_i^2 \lambda_i^{2k}}{\sum_{i=2}^n \omega_i^2 \lambda_i^{2k}(1-\lambda_i)} \\ &= \frac{\sum_{i>1} \omega_i^2 \lambda_i^{2(k+1)}(1-\lambda_i)}{\sum_{i>1} \omega_i^2 \lambda_i^{2k}(1-\lambda_i)} \cdot \frac{\omega_1^2 + \sum_{i>1} \omega_i^2 \lambda_i^{2k}}{\omega_1^2 + \sum_{i>1} \omega_i^2 \lambda_i^{2(k+1)}}\end{aligned}$$

The assumption $1 \geq \lambda_1 > \lambda_2 > \lambda_3 \geq \cdots \geq \lambda_n$ ensures that the limit of the first fraction is λ_2^2. The limit of the second fraction is 1 because $\omega_1 \neq 0$ and $\lambda_i < 1$ for each $i > 1$.

The last claim follows from Exercise 2.4. Indeed, $\lambda_2/\lambda_1 < 1$, so the relative errors $\operatorname{err}(\xi_k) \to 0$. ∎

2.3 The randomized power method (RPM)

So, what starting vector should we use in the PM?

Idea: Draw the starting vector for the power method at random.

Numerical analysis manuscripts often recommend that we initialize the PM with a *random* vector $\boldsymbol{\omega} \in \mathbb{R}^n$. This proposal is usually justified by the reasoning that a random starting vector is likely to have a nontrivial component in the direction of the dominant eigenvector of the matrix $\boldsymbol{A}$. That is, for any fixed basis, $\omega_1 \neq 0$ with high probability.

In fact, there are deeper reasons for choosing the starting vector at random, first recognized by Dixon [Dix83] and elaborated by Kuczyński & Woźniakowski [KW92]:

1. It leads to clean nonasymptotic error estimates.
2. We can obtain nontrivial error bounds, even when $\lambda_1 \approx \lambda_2$ so that $\lambda_2/\lambda_1 \approx 1$.

In this section, we will present a version of PM with a random starting vector. We will develop two theoretical results. The first one is a nonasymptotic analog of the classical analysis (Theorem 2.5). The second illustrates how we can obtain error bounds that do not depend on any assumptions about the spectrum of the matrix.

Algorithm 2.1 ApproxMaxEvec via randomized power method.

Input: Psd input matrix $\boldsymbol{A} \in \mathbb{H}_n$ and maximum number k of iterations
Output: Approximate minimum eigenpair $(\xi, \boldsymbol{y}) \in \mathbb{R} \times \mathbb{R}^n$ of the matrix $\boldsymbol{A}$

```
function ApproxMaxEvec(A; k)
    y = randn(n, 1)/√n                    ▷ Random initial vector
    for i = 1, ..., k do
        y = Ay
        y = y/‖y‖                         ▷ Approximate maximum eigenvector of A
    ξ = y*(Ay)                            ▷ Approximate maximum eigenvalue of A
```

2.3.1 Procedure

randomized power method (RPM)

The *randomized power method (RPM)* is just a specialization of the PM (Section 2.2.1) where the starting vector is drawn from a standard normal distribution: $\boldsymbol{\omega} \sim \textsc{normal}(\mathbf{0}, \mathbf{I}_n)$.

This choice of starting vector bears some discussion. For any fixed basis, since the standard normal distribution is rotationally invariant, the coordinates ω_i are independent, real standard normal variables. Thus, it is unrestrictive to work in a basis where $\boldsymbol{A}$ is diagonal.

There is a more subtle motivation for using a rotationally invariant distribution to compute eigenvectors. Since the maximum eigenvalue is a orthogonally invariant function of the matrix, we can show that the RPM achieves the minimum error by starting with a rotationally invariant vector. Since the RPM is scale invariant, we may as well work with the standard normal distribution, which is our favorite. See Section 2.6 for a rigorous statement and proof of this claim.

Algorithm 2.1 presents pseudocode for the RPM with a fixed total number of iterations. Each iteration requires a single matrix–vector multiplication with the input matrix, as well as $O(n)$ additional arithmetic. The overall storage cost is $O(n)$.

2.3.2 RPM, with a spectral gap

Our first theoretical result gives a nonasymptotic bound on the error in the RPM after a fixed number of iterations. Parallel with Theorem 2.5, the statement involves the first two eigenvalues of the matrix. This result is adapted from [KW92].

Theorem 2.6 (RPM: Analysis with a spectral gap). Let $\boldsymbol{A} \in \mathbb{H}_n(\mathbb{R})$ be a *real* psd matrix whose eigenvalues obey $\lambda_1 > \lambda_2$. For each $k = 1, 2, 3, \dots$, the relative error (2.1) in the eigenvalue estimates ξ_k produced by the RPM satisfies

$$\mathbb{E}\,\mathrm{err}(\xi_k) \le \sqrt{\frac{(n-1)\pi}{2}} \left(\frac{\lambda_2}{\lambda_1}\right)^k. \tag{2.3}$$

relative spectral gap

It is also common to state the result in terms of the *relative spectral gap* of the matrix:

$$\gamma := \frac{\lambda_1 - \lambda_2}{\lambda_1}. \tag{2.4}$$

Then (2.3) is equivalent with the bound

$$\mathbb{E}\,\mathrm{err}(\xi_k) \le \sqrt{\frac{(n-1)\pi}{2}}(1-\gamma)^k \le \sqrt{\frac{(n-1)\pi}{2}}\mathrm{e}^{-\gamma k}. \tag{2.5}$$

This formulation is sometimes easier to interpret.

To understand the meaning of Theorem 2.6, note that

$$k \geq \frac{1}{\gamma}\left[\log(\sqrt{\pi n/2}) + \log(1/\varepsilon)\right] \quad \text{implies} \quad \operatorname{err}(\xi_k) \leq \varepsilon.$$

This formula suggests that the RPM algorithm has a *burn-in period* of about $(\log n)/\gamma$ iterations during which it makes minimal progress. Afterward, it converges exponentially fast, reducing the error by a factor of $(1-\gamma) \approx \mathrm{e}^{-\gamma}$ at each iteration. *burn-in period*

We also observe that the number k_ε of iterations that suffices to achieve relative error ε is *logarithmic*: $k_\varepsilon \sim \log(1/\varepsilon)$. For contrast, recall that the Monte Carlo sampling method for estimating the trace requires ε^{-2} samples to bring the relative error below ε. The improved convergence behavior is a hallmark of (some) iterative methods; we cannot achieve it by plain random sampling.

Unfortunately, Theorem 2.6 gives a convergence rate that depends on the spectral gap γ. When the spectral gap is bounded away from zero, the RPM exhibits rapid convergence. On the other hand, when the spectral gap $\gamma \approx 0$, the bounds here are almost vacuous. We will address this shortcoming in Section 2.3.3.

Remark 2.7 (Burn-in). One may wonder whether the appearance of the burn-in period is an essential feature of this result. According to [SAR17], it is. For a worst-case (random) psd matrix, to estimate λ_1 up to a constant factor, we need at least $k \gtrsim (\log n)/(\log(\lambda_1/\lambda_2))$ matrix–vector multiplies with $\boldsymbol{A}$.

Proof of Theorem 2.6

The proof of Theorem 2.6 relies on a standard integral formula, coupled with some familiar Gaussian tail bounds. This analysis is due to your instructor.

Fact 2.8 (Exponential Integral). Let $g \sim \textsc{normal}(0,1)$ be a real standard normal random variable. For any $c > 0$,

$$\mathbb{E}\left[\frac{1}{g^2 + c}\right] = \frac{1}{\sqrt{c}}\mathrm{e}^{c/2}\int_{\sqrt{c}}^{\infty} \mathrm{e}^{-t^2/2}\,\mathrm{d}t \leq \min\left\{\sqrt{\frac{\pi}{2c}}, \frac{1}{c}\right\}.$$

See [Olv+10, Sec. 8.6.4] and [Ver18, Prop. 2.1.2]. ∎

Proof of Theorem 2.6. Let us revisit (2.2). Since we have chosen $\boldsymbol{\omega}$ from a standard normal distribution, its coordinates are independent standard normal variables. The coordinate ω_1 is isolated in the denominator, and it does not appear anywhere else. We invoke Fact 2.8 to integrate it out.

Using independence, we can first take the expectation $\mathbb{E}_{\omega_1}$ with respect to the first coordinate:

$$\mathbb{E}\operatorname{err}(\xi_k) = \mathbb{E}\,\mathbb{E}_{\omega_1}\left[\frac{\sum_{i>1}\omega_i^2\lambda_i^{2k}(1-\lambda_i)}{\omega_1^2 + \sum_{i>1}\omega_i^2\lambda_i^{2k}}\right] \leq \sqrt{\frac{\pi}{2}}\,\mathbb{E}\left[\frac{\sum_{i>1}\omega_i^2\lambda_i^{2k}(1-\lambda_i)}{\left(\sum_{i>1}\omega_i^2\lambda_i^{2k}\right)^{1/2}}\right].$$

The inequality is the first branch of Fact 2.8. Since the same term appears in the numerator and denominator, we can invoke the Cauchy–Schwarz inequality to obtain

$$\begin{aligned}
\mathbb{E}\operatorname{err}(\xi_k) &\leq \sqrt{\frac{\pi}{2}}\,\mathbb{E}\left[\left(\sum\nolimits_{i>1}\omega_i^2\lambda_i^{2k}(1-\lambda_i)^2\right)^{1/2}\right] \\
&\leq \sqrt{\frac{\pi}{2}}\left[\sum\nolimits_{i>1}(\mathbb{E}\,\omega_i^2)\lambda_i^{2k}(1-\lambda_i)^2\right]^{1/2} \\
&\leq \sqrt{\frac{\pi}{2}}\left[(n-1)\max_{i>1}\lambda_i^{2k}(1-\lambda_i)^2\right]^{1/2} \leq \sqrt{\frac{(n-1)\pi}{2}}(1-\gamma)^k.
\end{aligned}$$

The second inequality is Lyapunov's (for the expectation), and the third is Hölder's (for the sum). In the last step, we have used the definition (2.4) of the relative spectral gap. ∎

Complements

The perspicuous reader will have noticed that Theorem 2.6 predicts a convergence factor of (λ_2/λ_1), while the classical analysis in Theorem 2.5 gives a factor $(\lambda_2/\lambda_1)^2$. This might seem like a deficiency of our analysis. In fact, our result is essentially sharp.

Problem 2.9 (Kucziński and Woźniakowski). Instate the hypotheses of Theorem 2.6. Assuming that $\lambda_1 > \lambda_2 > \lambda_3$, prove that

$$\lim_{k\to\infty} \frac{\mathbb{E}\,\mathrm{err}(\xi_k)}{(\lambda_2/\lambda_1)^k} = 1 - \frac{\lambda_2}{\lambda_1}.$$

Explain why the formula in the last display does not contradict Theorem 2.5.

The direct analysis here makes it easy to derive many interesting variants of Theorem 2.6.

Exercise 2.10 (Tropp). In (2.3), prove that we can replace the dimension $n-1$ by the stable rank, $\mathrm{srank}(\boldsymbol{A})$, if we take one additional step of the power method. Can you modify the result further to replace the stable rank by spectral decay quantities that are (perhaps) even smaller?

Exercise 2.11 (Kucziński and Woźniakowski; Tropp). Suppose that $\lambda_1 = \lambda_2 = \cdots = \lambda_m$, where $m > 1$. Explain how to modify the analysis in this section to achieve superior convergence rates for the RPM. The most interesting cases are $m = 2$ and $m = 3$.

Exercise 2.12 (Tropp). Let $\boldsymbol{A} \in \mathbb{H}_n(\mathbb{C})$ be a *complex* psd matrix. Suppose that we start the PM with a complex standard normal vector $\boldsymbol{\omega}$ to obtain a maximum eigenvalue estimate ξ_k. Deduce a bound on the relative error (2.1) in terms of the relative spectral gap.

Problem 2.13 (Kucziński and Woźniakowski; Tropp). Develop a bound on the failure probability $\mathbb{P}\{\mathrm{err}(\xi_k) > \varepsilon\}$ in terms of the spectral gap. **Hint:** Rearrange the terms in the event to remove the ratio, and estimate the moment generating function of the resulting random variable.

2.3.3 RPM, without a spectral gap

We have alluded to the fact that RPM still works, regardless of whether the matrix $\boldsymbol{A}$ actually displays a spectral gap. This result may come as a shock because the spectral gap takes a central place in the classical analysis of the power method. Here, we start to see the utility of developing probabilistic variants of familiar numerical methods.

Theorem 2.14 (RPM: Analysis without a spectral gap). Let $\boldsymbol{A} \in \mathbb{H}_n(\mathbb{R})$ be a *real* psd matrix. For each $k = 1, 2, 3, \ldots$, the relative error (2.1) in the eigenvalue estimates ξ_k produced by the RPM satisfies

$$\mathbb{E}\,\mathrm{err}(\xi_k) \le \frac{1}{k}\left[1 + \log\sqrt{\frac{(n-1)\pi}{2}} + \log k\right]. \tag{2.6}$$

To appreciate the meaning of Theorem 2.14, observe that

$$k \ge \frac{1}{\varepsilon}\left[1 + \log\sqrt{\pi n/2} + \log k\right] \quad \text{implies} \quad \mathbb{E}\,\mathrm{err}(\xi_k) \le \varepsilon.$$

In words, this result suggests that there is a burn-in period of about $\log n$ steps during which the algorithm makes negligible progress. Afterward, the error declines at least as fast as $(\log k)/k$. Incidentally, the term $\log k$ is spurious, and it can be removed by different and more involved arguments.

Theorem 2.14 indicates that the number k_ε of iterations we should perform to attain relative error ε declines roughly like $1/\varepsilon$. This is far worse than the $\log(1/\varepsilon)$ we saw in Theorem 2.6, but it is far better than the $1/\varepsilon^2$ cost of Monte Carlo sampling methods.

Remark 2.15 (These theorems predict different things!). It may initially seem confusing that we have two results that describe two different behaviors of the PM. Both of them are valid, so we can use whichever one makes a stronger prediction for a given use case.

Proof of Theorem 2.14

The argument follows the same lines as Theorem 2.6, but we need to be more careful. The key idea, due to [KW92], is to treat eigenvalues close to one differently from the smaller eigenvalues. The streamlined development here is due to your instructor.

Proof of Theorem 2.14. As in the proof of Theorem 2.6,

$$\mathbb{E}\,\mathrm{err}(\xi_k) = \mathbb{E}\,\mathbb{E}_{\omega_1}\left[\frac{\sum_{i>1}\omega_i^2\lambda_i^{2k}(1-\lambda_i)}{\omega_1^2 + X}\right], \quad \text{where} \quad X := \sum\nolimits_{i>1}\omega_i^2\lambda_i^{2k}.$$

Note that ω_1 is independent from the numerator of the fraction and from X. Use both branches of Fact 2.8 to arrive at the estimate

$$\mathbb{E}\,\mathrm{err}(\xi_k) \le \mathbb{E}\left[\left(\sum\nolimits_{i>1}\omega_i^2\lambda_i^{2k}(1-\lambda_i)\right)\cdot\min\left\{\sqrt{\frac{\pi}{2X}}, \frac{1}{X}\right\}\right]. \tag{2.7}$$

To continue, we split the sum, depending on whether λ_i is small or large.

Introduce a parameter $\beta > 0$ that remains at our disposal. Segregating terms with $\lambda_i \le 1-\beta$ from those with $\lambda_i > 1-\beta$, we find that

$$\begin{aligned}\sum\nolimits_{i>1}\omega_i^2\lambda_i^{2k}(1-\lambda_i) &\le \sum\nolimits_{\lambda_i\le 1-\beta}\omega_i^2\lambda_i^{2k} + \beta\sum\nolimits_{\lambda_i>1-\beta}\omega_i^2\lambda_i^{2k}\\ &\le \sum\nolimits_{\lambda_i\le 1-\beta}\omega_i^2\lambda_i^{2k} + \beta X.\end{aligned}$$

Substitute this relation into (2.7) to conclude that

$$\begin{aligned}\mathbb{E}\,\mathrm{err}(\xi_k) &\le \mathbb{E}\left[\left(\sum\nolimits_{\lambda_i\le 1-\beta}\omega_i^2\lambda_i^{2k} + \beta X\right)\cdot\min\left\{\sqrt{\frac{\pi}{2X}}, \frac{1}{X}\right\}\right]\\ &\le \mathbb{E}\left[\left(\sum\nolimits_{\lambda_i\le 1-\beta}\omega_i^2\lambda_i^{2k}\right)\sqrt{\frac{\pi}{2X}} + \frac{\beta X}{X}\right]\\ &\le \sqrt{\frac{\pi}{2}}\,\mathbb{E}\left[\left(\sum\nolimits_{\lambda_i\le 1-\beta}\omega_i^2\lambda_i^{2k}\right)^{1/2}\right] + \beta\\ &\le \frac{\pi}{2}\left(\sum\nolimits_{\lambda_i\le 1-\beta}\lambda_i^{2k}\right)^{1/2} + \beta.\end{aligned}$$

To pass from the second line to the third line, we used the fact that X is at least as large as the parenthesis. Then we invoked Lyapunov's inequality to draw the expectation inside the square root.

To complete the argument, we make a simple bound on the remaining sum:

$$\mathbb{E}\,\mathrm{err}(\xi_k) \le \sqrt{\frac{(n-1)\pi}{2}}(1-\beta)^k + \beta \le \sqrt{\frac{(n-1)\pi}{2}}\mathrm{e}^{-\beta k} + \beta.$$

Minimize the right-hand side with respect to β to arrive at the advertised result. ∎

Complements

As with Theorem 2.6, we can obtain many variants of Theorem 2.14 by simple modifications of the argument.

Exercise 2.16 (Tropp). In (2.6), show that we can replace the dimension $n - 1$ by the stable rank, $\mathrm{srank}(\boldsymbol{A})$, if we take one additional step of the power method. Can you modify the result further to obtain bounds that depend on smaller spectral decay quantities?

Exercise 2.17 (Kuczyński and Woźniakowski; Tropp). Suppose that λ_1 has multiplicity m, where $m > 1$. Explain how to modify the analysis in this section to achieve superior convergence rates for the RPM. The most interesting cases are $m = 2$ and $m = 3$.

Exercise 2.18 (Tropp). Let $\boldsymbol{A} \in \mathbb{H}_n(\mathbb{C})$ be a *complex* psd matrix. Suppose that we start PM with a complex standard normal vector $\boldsymbol{\omega}$ to obtain a maximum eigenvalue estimate ξ_k. Deduce a bound on the relative error (2.1) that does not depend on the spectral gap γ.

Problem 2.19 (Kuczyński and Woźniakowski). Can you replace the term $\log k$ by a constant in the bound (2.6)?

Problem 2.20 (Kuczyński and Woźniakowski; Tropp). Develop a bound for the failure probability $\mathbb{P}\{\mathrm{err}(\xi_k) > \varepsilon\}$ that does not depend on the spectral gap.

2.4 The randomized Krylov method (RKM)

Numerical analysts *really* do not like the power method. Here are some typical quotations:

> "Unfortunately, although power iteration is famous, it is by no means an effective tool for general use. Except for special matrices, it is very slow."—Trefethen and Bau [TB97, p. 191].

> "The power method is no longer a serious technique for computing eigenvectors."—Parlett [Par98, p. 61].

I think that these objections are, perhaps, too strong. Nevertheless, you should be aware that the power method has major deficiencies that are remedied by more sophisticated algorithms.

Idea: Use all of the vectors computed by the power method to obtain a better approximation of the maximum eigenvalue.

Krylov methods use all the vectors constructed by the PM to obtain an eigenvalue estimate, and they exhibit far better performance. This section gives a short summary of the kinds of results one can obtain for maximum eigenvalue computations using Krylov subspaces. Next time, we will discuss one of the basic implementations of a Krylov method, due to Lanczos.

2.4.1 Procedure

randomized Krylov method (RKM)

Let us outline the *randomized Krylov method (RKM)*. Run the PM on a real psd matrix $\boldsymbol{A} \in \mathbb{H}_n(\mathbb{R})$ with the random starting vector $\boldsymbol{\omega} \sim \textsc{normal}(\mathbf{0}, \boldsymbol{I}_n)$. Implicitly, form the subspace generated by all the vectors that arise in the iteration:

$$\mathsf{K}_{k+1} := \mathsf{K}_{k+1}(\boldsymbol{A}, \boldsymbol{\omega}) := \mathrm{span}\left\{\boldsymbol{\omega}, \boldsymbol{A}\boldsymbol{\omega}, \dots, \boldsymbol{A}^k\boldsymbol{\omega}\right\}. \tag{2.8}$$

The span is called a *Krylov subspace*. It is closely connected with matrix polynomials. Indeed,

Krylov subspace

$$\boldsymbol{u} \in \mathsf{K}_{k+1} \quad \text{if and only if} \quad \boldsymbol{u} = \sum\nolimits_{i=0}^{k} c_i \boldsymbol{A}^i \boldsymbol{\omega} = \varphi(\boldsymbol{A})\boldsymbol{\omega}.$$

In this expression, the c_i are real scalars, and φ is the degree-k polynomial $\varphi(\lambda) = \sum_{i=0}^{k} c_i \lambda^i$. Recall that $\varphi(\boldsymbol{A})$ is the spectral function of $\boldsymbol{A}$ induced by the polynomial φ.

Now, to estimate the maximum eigenvalue of $\boldsymbol{A}$, we maximize the Rayleigh quotient over the Krylov subspace:

$$\xi_k = \max_{\boldsymbol{u} \in \mathsf{K}_{k+1}} \frac{\boldsymbol{u}^* \boldsymbol{A} \boldsymbol{u}}{\|\boldsymbol{u}\|^2} = \max_{\deg \varphi \le k} \frac{\boldsymbol{\omega}^* \boldsymbol{A} \varphi^2(\boldsymbol{A}) \boldsymbol{\omega}}{\boldsymbol{\omega}^* \varphi^2(\boldsymbol{A}) \boldsymbol{\omega}}.$$

The maximization occurs over all polynomials φ with degree at most k. In contrast, the power method makes a similar eigenvalue estimate using the fixed monomial $\varphi(\lambda) = \lambda^k$. Since the Krylov method optimizes over all degree-k polynomials, we anticipate that it may perform better.

Remark 2.21 (Implementation). The description here falls very far short of a practical implementation of the Krylov method. In the next lecture, we will talk more about how to perform the required calculations.

Exercise 2.22 (Invariance). Show that the Krylov subspace is invariant under affine functions of the input matrix: $\boldsymbol{A} \mapsto \alpha \boldsymbol{A} + \beta \mathbf{I}$ for real scalars α, β. Does the Krylov method care whether the input matrix $\boldsymbol{A}$ is psd?

Exercise 2.23 (Minimum eigenvalues). Explain how to modify the Krylov method to estimate the minimum eigenvalue of a psd matrix.

Exercise 2.24 (Maximum singular values). Explain how to modify the Krylov method to estimate the maximum singular value of a rectangular matrix.

2.4.2 RKM, with a spectral gap

We present without proof a result of Kuczyński and Woźniakowski [KW92] that describes the convergence behavior of RKM in the presence of a gap between the first and second eigenvalues.

Theorem 2.25 (RKM: Analysis with a spectral gap). Let $\boldsymbol{A} \in \mathbb{H}_n(\mathbb{R})$ be a *real* psd matrix with m distinct eigenvalues that obey $\lambda_1 > \lambda_2$. For each $k = 1, 2, 3, \dots, m$, the relative error (2.1) in the eigenvalue estimates ξ_k produced by the RKM satisfies

$$\mathbb{E}\, \mathrm{err}(\xi_k) \le 2.589\sqrt{n} \left(1 - 2\sqrt{1 - \frac{\lambda_2}{\lambda_1}}\right)^k \le 2.589\sqrt{n}\, \mathrm{e}^{-2k\sqrt{\gamma}}. \tag{2.9}$$

The relative spectral gap γ is defined in (2.4).

A valuable consequence of Theorem 2.25 is the iteration complexity bound

$$k \ge \frac{1}{\sqrt{\gamma}} \left[\log(2.589\sqrt{n}) + \log(1/\varepsilon)\right] \quad \text{implies} \quad \mathrm{err}(\xi_k) \le \varepsilon.$$

It is also productive to compare Theorem 2.25 with Theorem 2.6. The key difference is that the RKM only pays for the *square root* of the spectral gap γ, while the RPM pays for γ. When $\gamma \approx 0$, this fact points to a very substantial difference in performance. On the other hand, like the RPM, the RKM still suffers from the same burn-in period of about $\log(n)/\gamma$ iterations.

Problem 2.26 (Kucyński and Woźniakowski; Tropp). Prove Theorem 2.25. There is a simple argument that is not so different from the proof of Theorem 2.6, but we need to choose a special polynomial in the last step to attain the strongest guarantees. The right choice is a shifted and scaled Chebyshev polynomial of the first kind.

2.4.3 RKM, without a spectral gap

Next, we state without proof another result of Kuczyński and Woźniakowski [KW92] that describes the error achieved by the RKM, even when there is no gap between the first and second eigenvalues.

Theorem 2.27 (RKM: Analysis without a spectral gap). Let $\boldsymbol{A} \in \mathbb{H}_n(\mathbb{R})$ be a *real* psd matrix. For each $k = 3, 4, 5, \dots$, the relative error (2.1) in the eigenvalue estimates ξ_k produced by the RPM satisfies

$$\mathbb{E}\,\mathrm{err}(\xi_k) \le 2.575 \left[\frac{\log n}{k}\right]^2. \tag{2.10}$$

Theorem 2.27 implies the iteration complexity bound

$$k \ge \frac{1.605 \log(n)}{\sqrt{\varepsilon}} \quad \text{implies} \quad \mathrm{err}(\xi_k) \le \varepsilon.$$

Comparing Theorem 2.27 with Theorem 2.14, we see that RKM only pays for the *square root* of the error tolerance ε, while RPM pays for ε itself. This is a big deal when $\varepsilon \approx 0$. The result also predicts that RKM has a burn-in period of about $\log n$ iterations.

Problem 2.28 (Kucyński and Woźniakowski; Tropp). Prove Theorem 2.27. There is a fairly simple argument that parallels the proof of Theorem 2.14. In this case, when we select a polynomial, the optimal choice is a shifted and scaled Chebyshev polynomial of the second kind.

2.5 Context: First-Order Convex Optimization

It is interesting to compare RPM and RKM with standard first-order optimization algorithms to see how the error declines with the iteration k. Our eigenvalue computation methods perform one matrix–vector multiplication per iteration, while a first-order optimization algorithm performs one gradient evaluation per iteration. See Bubeck [Bub15, p. 13] for a summary.

- For a nonsmooth, Lipschitz convex problem, subgradient descent methods achieve a dimension-free convergence rate of $k^{-1/2}$. This is the same rate attained by Monte Carlo sampling estimators (for the trace).
- For a smooth convex problem, gradient descent methods achieve a convergence rate of k^{-1}. This is the same rate attained by the RPM when there is no spectral gap.
- For a smooth convex problem, accelerated gradient descent methods achieve a convergence rate of k^{-2}. This is the rate attained by the RKM when there is no spectral gap.
- For a smooth and strongly convex problem, gradient descent achieves an exponential convergence rate e^{-ck}, where c reflects the condition number of the optimization problem. This is the rate attained by the RPM in the presence of a spectral gap.

- For a smooth and strongly convex problem, accelerated gradient descent achieves an exponential convergence rate $\mathrm{e}^{-\sqrt{c}\,k}$, where c reflects the condition number. This is the rate attained by RKM in the presence of a spectral gap.

These parallels are not a coincidence; see [Har18]. One might be tempted to treat maximum eigenvector computations as an optimization problem and apply standard optimization algorithms. Nevertheless, it is very hard to beat RKM for maximum eigenvalue computations when we are working with the matrix–vector multiplication primitive.

2.6 Rotationally invariant distributions

When running RPM or RKM, it is mathematically appealing to choose the starting vector $\boldsymbol{\omega}$ from the standard normal distribution. Nevertheless, we may ask whether this is actually the best choice. In this section, we establish a general result that justifies this decision. These ideas are implicit in the paper [KW92].

2.6.1 Averaging an orthogonally invariant function

We call a bivariate function $f : \mathbb{H}_n(\mathbb{R}) \times \mathbb{R}^n \to \mathbb{R}$ *orthogonally invariant* if it obeys

$$f(\boldsymbol{A}; \boldsymbol{v}) = f\left(\boldsymbol{O}\boldsymbol{A}\boldsymbol{O}^*; \boldsymbol{O}\boldsymbol{v}\right) \quad \text{for each orthogonal matrix } \boldsymbol{O} \in \mathbb{M}_n(\mathbb{R}).$$

Fix a matrix $\boldsymbol{\Lambda} \in \mathbb{H}_n$ and consider the orthogonal orbit

$$\mathscr{A} := \mathscr{A}(\boldsymbol{\Lambda}) = \{\boldsymbol{O}\boldsymbol{\Lambda}\boldsymbol{O}^* : \boldsymbol{O} \in \mathbb{M}_n(\mathbb{R}) \text{ is orthogonal}\}.$$

For the worst matrix in the orthogonal orbit, the average value of an orthogonally invariant function is minimized when the vector $\boldsymbol{v}$ is chosen from a rotationally invariant distribution.

Proposition 2.29 (Spherical symmetry). Consider an orthogonally invariant bivariate function and a random vector $\boldsymbol{\omega} \in \mathbb{R}^n$. Let $\boldsymbol{\Omega} \in \mathbb{R}^{n\times n}$ be a uniformly random orthogonal matrix, drawn independently from $\boldsymbol{\omega}$. Then,

$$\max_{\boldsymbol{A}\in\mathscr{A}} \mathbb{E}_{\boldsymbol{\Omega},\boldsymbol{\omega}}\, f(\boldsymbol{A}; \boldsymbol{\Omega}\boldsymbol{\omega}) \le \max_{\boldsymbol{A}\in\mathscr{A}} \mathbb{E}_{\boldsymbol{\omega}}\, f(\boldsymbol{A}; \boldsymbol{\omega}).$$

Proof. By the orthogonal invariance of f,

$$\begin{aligned}\max_{\boldsymbol{A}\in\mathscr{A}} \mathbb{E}_{\boldsymbol{\Omega},\boldsymbol{\omega}}\, f(\boldsymbol{A}; \boldsymbol{\Omega}\boldsymbol{\omega}) &= \max_{\boldsymbol{A}\in\mathscr{A}} \mathbb{E}_{\boldsymbol{\Omega},\boldsymbol{\omega}}\, f(\boldsymbol{\Omega}^*\boldsymbol{A}\boldsymbol{\Omega}; \boldsymbol{\omega}) \\ &\le \mathbb{E}_{\boldsymbol{\Omega}} \max_{\boldsymbol{A}\in\mathscr{A}} \mathbb{E}_{\boldsymbol{\omega}}\, f\left(\boldsymbol{\Omega}^*\boldsymbol{A}\boldsymbol{\Omega}; \boldsymbol{\omega}\right) = \max_{\boldsymbol{A}\in\mathscr{A}} \mathbb{E}_{\boldsymbol{\omega}}\, f(\boldsymbol{A}; \boldsymbol{\omega}).\end{aligned}$$

The inequality is Jensen's. The last identity follows from the definition of $\mathscr{A}$. ∎

2.6.2 The maximum eigenvalue

As a specific application, we study the problem of estimating the maximum eigenvalue of the worst matrix with eigenvalue spectrum $\boldsymbol{\Lambda}$ using RPM or RKM. Fix a natural number k, and define the orthogonally invariant function

$$f(\boldsymbol{A}; \boldsymbol{v}) = \operatorname{err}\bigl(\xi_k(\boldsymbol{A}, \boldsymbol{v})\bigr).$$

where $\xi_k(\boldsymbol{A}, \boldsymbol{v})$ is the eigenvalue estimate produced by k iterations of the power method or the Krylov method for the matrix $\boldsymbol{A} \in \mathbb{H}_n$ with the starting vector $\boldsymbol{v} \in \mathbb{R}^n$. The orthogonal invariance property follows from (2.1).

Let $\boldsymbol{v} \in \mathbb{R}^n$ be a random vector with any distribution whatsoever. Proposition 2.29 states that

$$\max_{\boldsymbol{A} \in \mathcal{A}} \mathbb{E}\, \mathrm{err}\big(\xi_k(\boldsymbol{\Omega v})\big) \leq \max_{\boldsymbol{A} \in \mathcal{A}} \mathbb{E}\, \mathrm{err}\big(\xi_k(\boldsymbol{A}, \boldsymbol{v})\big).$$

That is, for the worst-case input matrix, drawing a starting vector $\boldsymbol{\omega} = \boldsymbol{\Omega v}$ with a rotationally invariant distribution is no worse on average than drawing the starting vector $\boldsymbol{v}$. Since $\xi_k(\boldsymbol{A}, \boldsymbol{v})$ does not depend on the scale of the vector $\boldsymbol{v}$, the distribution of the norm $\|\boldsymbol{\omega}\|$ of the random vector does not play a role in the behavior of the algorithms. Thus, we may as well take $\boldsymbol{\omega}$ to be the standard normal vector.

The argument here is general enough to handle a wide range of other functions. For instance, we could take

$$f(\boldsymbol{A}; \boldsymbol{v}) = \mathbb{1}\{\mathrm{err}(\xi_k(\boldsymbol{A}, \boldsymbol{v})) \geq \varepsilon\}.$$

Then Proposition 2.29 shows that a spherically symmetric distribution also minimizes the probability of a large error. Likewise, we can apply the same methodology to study other kinds of NLA problems that involve rotationally invariant quantities.

Problems

Numerics 2.30 (Power to the People). In this problem, we will explore the behavior of the randomized power method as applied to several examples.

1. Implement the randomized power method to obtain a sequence of estimates $\{\xi_k\}$ for the maximum eigenvalue of a psd matrix. Allow either a real or complex normal starting vector. **Hint:** You may need to take the real part of the Rayleigh quotient in the complex case.
2. Write code to plot the sample paths $k \mapsto \xi_k$ for 100 realizations of the power method for different starting vectors. Compare the real and complex case. (Use translucent lines!)
3. For $n = 1000$, make plots of the sample paths for the Laplacian, the inverse Laplacian, and `LowRankHiNoise` with $R = 1$ and $\mathbb{F} = \mathbb{R}$. Do you witness burn-in phenomena? Which type of convergence behavior is visible ($1/k$ or e^{-k})? How does the convergence rate depend on the choice of a real or complex starting vector?
4. Repeat these experiments using a randomized Krylov subspace method. You may use `eig`, or equivalent, to compute the maximum eigenvalue of the compression of the input matrix to the Krylov subspace. Discuss.

Lecture bibliography

[Bub15] S. Bubeck. "Convex Optimization: Algorithms and Complexity". In: *Found. Trends Mach. Learn.* 8.3–4 (Nov. 2015), pages 231–357. DOI: 10.1561/2200000050.

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

[Har18] M. Hardt. *Convex optimization and approximation*. Course website, Berkeley EE 227C (Spring 2018). 2018. URL: https://ee227c.github.io.

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

[Olv+10] F. W. J. Olver et al., editors. *NIST handbook of mathematical functions*. U.S. Department of Commerce, National Institute of Standards and Technology, Washington, DC; Cambridge University Press, Cambridge, 2010, pages xvi+951.

[Par98] B. N. Parlett. *The symmetric eigenvalue problem*. Volume 20. Classics in Applied Mathematics. Corrected reprint of the 1980 original. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1998, pages xxiv+398. DOI: 10.1137/1.9781611971163.

[SAR17] M. Simchowitz, A. E. Alaoui, and B. Recht. "On the Gap Between Strict-Saddles and True Convexity: An Omega(log d) Lower Bound for Eigenvector Approximation". In: *CoRR* abs/1704.04548 (2017). arXiv: 1704.04548.

[TB97] L. N. Trefethen and D. Bau III. *Numerical linear algebra*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1997, pages xii+361. DOI: 10.1137/1.9780898719574.

[Ver18] R. Vershynin. *High-dimensional probability*. Volume 47. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, Cambridge, 2018, pages xiv+284. DOI: 10.1017/9781108231596.

[Woo14] D. P. Woodruff. "Sketching as a Tool for Numerical Linear Algebra". In: *Foundations and Trends in Theoretical Computer Science* 10.1-2 (2014), pages 1 –157. DOI: 10.1561/0400000060.

3. The Lanczos Method

Date: 14 January 2020 **Scribe:** Dmitry Burov

Agenda:

1. Krylov subspaces
2. Arnoldi iteration
3. Lanczos iteration
4. Spectral functions
5. Gaussian quadrature
6. Lanczos quadrature
7. Stochastic Lanczos quadrature

In the previous lecture, we introduced the power method (PM) for computing the largest eigenvalue of a matrix. We briefly mentioned Krylov subspace methods, which improve over the PM by using all of the information collected during the iteration. This lecture continues our treatment of Krylov subspace methods by presenting two iterative algorithms, the Arnoldi and Lanczos iterations, both of which construct a distinguished basis for a Krylov subspace. Furthermore, we will develop a beautiful result that connects the Lanczos iteration with the problem of computing the quadratic form induced by a spectral function; that is, a quantity of the form $\boldsymbol{x}^* f(\boldsymbol{A})\boldsymbol{x}$. Finally, at the end of the lecture, we will combine these ideas to develop a stochastic estimator for the trace of a spectral function $\operatorname{tr} f(\boldsymbol{A})$.

3.1 Krylov subspaces

Our initial focus is on computing the maximum eigenvalue of a self-adjoint matrix, under the constraint that we can only access the matrix via matrix–vector multiplication. We are going to work in the real setting today to avoid keeping track of complex conjugates.

Computational Problem (Maximum eigenvalue). For a self-adjoint matrix $\boldsymbol{A} \in \mathbb{H}_n(\mathbb{R})$, compute $\lambda_{\max}(\boldsymbol{A})$.

Computational Primitive (Matrix–vector multiplication). We can compute $\boldsymbol{u} \mapsto \boldsymbol{A}\boldsymbol{u}$ for any $\boldsymbol{u} \in \mathbb{R}^n$.

Krylov subspaces give a very powerful mechanism for exploiting the matrix–vector product to extract information about a matrix. Recall that the Krylov subspace of depth k is defined as

$$\mathsf{K}_k := \mathsf{K}_k(\boldsymbol{A}; \boldsymbol{x}) := \operatorname{span}\left\{\boldsymbol{x}, \boldsymbol{A}\boldsymbol{x}, \dots, \boldsymbol{A}^{k-1}\boldsymbol{x}\right\}.$$

The starting vector $\boldsymbol{x} \in \mathbb{R}^n$ is typically chosen by the user of the Krylov subspace. Note that the dimension of K_k does not exceed k.

As we have mentioned, the Krylov subspace K_k arises from $k-1$ steps of power iteration, provided that we *accumulate* the information from the PM instead of retaining only the highest-order monomial, i.e., $\boldsymbol{A}^{k-1}$.

As outlined in the previous lecture, to approximate the maximum eigenvalue $\lambda_{\max}(\boldsymbol{A})$, we can maximize the Rayleigh quotient over vectors $\boldsymbol{u}$ in the Krylov subspace:

$$\xi_{k-1} = \max_{\boldsymbol{u} \in \mathsf{K}_k} \frac{\boldsymbol{u}^* \boldsymbol{A} \boldsymbol{u}}{\boldsymbol{u}^* \boldsymbol{u}} = \max_{\deg \varphi \le k-1} \frac{\boldsymbol{x}^* \varphi^2(\boldsymbol{A}) \boldsymbol{A} \boldsymbol{x}}{\boldsymbol{x}^* \varphi^2(\boldsymbol{A}) \boldsymbol{x}}. \tag{3.1}$$

Indeed, every vector $\boldsymbol{u} \in \mathsf{K}_k$ has the form $\boldsymbol{u} = \sum_{i=0}^{k-1} c_i \boldsymbol{A}^i \boldsymbol{x} = \varphi(\boldsymbol{A})\boldsymbol{x}$, where $\varphi(t) = \sum_{i=0}^{k-1} c_i t^i$ is a polynomial with degree $k-1$ or less. We also used the fact that $\boldsymbol{A}$ is self-adjoint ($\boldsymbol{A} = \boldsymbol{A}^*$) and commutes with any matrix polynomial ($\boldsymbol{A}\varphi(\boldsymbol{A}) = \varphi(\boldsymbol{A})\boldsymbol{A}$). Theorems 2.25 and 2.27 state that the Krylov method significantly outperforms the power method when the starting vector is chosen at random from a standard normal distribution.

Today, we will be addressing the computational question: How do we actually solve the maximization problem (3.1)? What is the computational cost, as compared with the PM? We will develop the *Krylov scheme*; a procedure that conceptually involves three steps:

Krylov scheme

1. Compute an orthonormal basis $\boldsymbol{Q}_k$ for the Krylov subspace K_k.
2. Compress the input matrix to the Krylov subspace: $\boldsymbol{H}_k = \boldsymbol{Q}_k^* \boldsymbol{A} \boldsymbol{Q}_k$.
3. Compute the maximum eigenvalue of the compression: $\lambda_{\max}(\boldsymbol{H}_k)$.

We are going to focus on methods for efficiently performing the first two steps in concert.

Exercise 3.1 (Invariance). Assume that $\boldsymbol{A} \in \mathbb{H}_n$ is self-adjoint, and let $\boldsymbol{x} \in \mathbb{F}^n$ be a starting vector. Fix a natural number k. Verify the invariance properties of the Krylov subspace $\mathsf{K}_k(\boldsymbol{A}; \boldsymbol{x})$.

1. **Scale invariance:** $\mathsf{K}_k(\boldsymbol{A}; z\boldsymbol{x}) = \mathsf{K}_k(\boldsymbol{A}; \boldsymbol{x})$ for each nonzero $z \in \mathbb{F}$.
2. **Affine invariance:** $\mathsf{K}_k(\alpha\boldsymbol{A} + \beta\mathbf{I}; \boldsymbol{x}) = \mathsf{K}_k(\boldsymbol{A}; \boldsymbol{x})$ for all $\beta \in \mathbb{R}$ and all nonzero $\alpha \in \mathbb{R}$.
3. **Rotation covariance:** $\mathsf{K}_k(\boldsymbol{U}\boldsymbol{A}\boldsymbol{U}^*; \boldsymbol{U}\boldsymbol{x}) = \boldsymbol{U}\mathsf{K}_k(\boldsymbol{A}; \boldsymbol{x})$ for each orthogonal/unitary $\boldsymbol{U}$.

3.2 Arnoldi iteration

We approach the first step in the Krylov scheme for computing the maximum eigenvalue by means of the *Arnoldi iteration*, which computes a special basis for a Krylov subspace $\mathsf{K}_k(\boldsymbol{A}; \boldsymbol{x})$. Later, we will see that something remarkable happens when the matrix $\boldsymbol{A}$ is self-adjoint.

Arnoldi iteration

The idea behind Arnoldi iteration is to apply the modified Gram–Schmidt (MGS) process to a cleverly chosen sequence of vectors:

Idea: Sequentially construct an orthonormal basis for the Krylov space:

$$\operatorname{span}\{\boldsymbol{q}_1, \dots, \boldsymbol{q}_k\} = \operatorname{span}\{\boldsymbol{x}, \boldsymbol{A}\boldsymbol{x}, \dots, \boldsymbol{A}^{k-1}\boldsymbol{x}\} \qquad \text{for each } k = 1, 2, 3, \dots.$$

At each step, we obtain the next vector $\boldsymbol{q}_{k+1}$ by applying MGS to $\boldsymbol{A}\boldsymbol{q}_k$.

For concreteness, let us see how this process plays out.

- $k = 1$. The first vector $\boldsymbol{q}_1$ must be parallel to the starting vector $\boldsymbol{x}$. Thus, we take
$$\boldsymbol{q}_1 = \boldsymbol{x}/\|\boldsymbol{x}\|.$$
Clearly, $\boldsymbol{q}_1 \in \text{span}\,\{\boldsymbol{x}\}$.
- $k = 2$. Now, we orthogonalize $\boldsymbol{A}\boldsymbol{q}_1$ against the first vector in our orthonormal basis:
$$\begin{aligned}\hat{\boldsymbol{q}}_2 &= \boldsymbol{A}\boldsymbol{q}_1 - \langle \boldsymbol{A}\boldsymbol{q}_1,\ \boldsymbol{q}_1\rangle \boldsymbol{q}_1;\\ \boldsymbol{q}_2 &= \hat{\boldsymbol{q}}_2/\|\hat{\boldsymbol{q}}_2\|.\end{aligned}$$
Since $\boldsymbol{A}\boldsymbol{q}_1 \in \text{span}\{\boldsymbol{A}\boldsymbol{x}\}$, we have $\hat{\boldsymbol{q}}_2 \in \text{span}\{\boldsymbol{x}, \boldsymbol{A}\boldsymbol{x}\}$. Moreover, $\hat{\boldsymbol{q}}_2$ is orthogonal to $\boldsymbol{q}_1$. We simply need normalization to obtain $\boldsymbol{q}_2$.
- $k = 3$. Next, we orthogonalize $\boldsymbol{A}\boldsymbol{q}_2$ against the first two vectors:
$$\begin{aligned}\hat{\boldsymbol{q}}_3 &= \boldsymbol{A}\boldsymbol{q}_2 - \langle \boldsymbol{A}\boldsymbol{q}_2,\ \boldsymbol{q}_2\rangle \boldsymbol{q}_2 - \langle \boldsymbol{A}\boldsymbol{q}_2,\ \boldsymbol{q}_1\rangle \boldsymbol{q}_1;\\ \boldsymbol{q}_3 &= \hat{\boldsymbol{q}}_3/\|\hat{\boldsymbol{q}}_3\|.\end{aligned}$$
- And so on. For each k, use MGS to orthogonalize $\boldsymbol{A}\boldsymbol{q}_k$ against $\boldsymbol{q}_1, \dots, \boldsymbol{q}_k$.

To appreciate what is happening here, rewrite this procedure in matrix form. Let $\boldsymbol{Q}_k$ be the $n \times k$ orthonormal matrix that collects the first k vectors in the basis. Then

$$\begin{bmatrix}\boldsymbol{A}\end{bmatrix}\begin{bmatrix}\boldsymbol{q}_1 & \cdots & \boldsymbol{q}_k\end{bmatrix} = \begin{bmatrix}\boldsymbol{q}_1 & \cdots & \boldsymbol{q}_k & \boldsymbol{q}_{k+1}\end{bmatrix}\begin{bmatrix} * & * & \cdots & * \\ * & * & \cdots & * \\ & * & \cdots & * \\ & & \ddots & \vdots \\ & & & * \end{bmatrix}. \tag{3.2}$$

$$\boldsymbol{A} \qquad \boldsymbol{Q}_k \qquad = \qquad \boldsymbol{Q}_{k+1} \qquad \widehat{\boldsymbol{H}}_k$$

Indeed, the coefficient matrix $\widehat{\boldsymbol{H}}_k$ collects the (scaled) inner-products that arise during the MGS procedure. At each iteration, the $(k+1) \times k$ coefficient matrix $\widehat{\boldsymbol{H}}_k$ extends by one column and one row. Since we orthogonalize $\boldsymbol{A}\boldsymbol{q}_k$ against $\boldsymbol{q}_1, \dots, \boldsymbol{q}_k$, the nonzero coefficients appear on or above the first subdiagonal. A matrix with this form is called *upper Hessenberg*.

upper Hessenberg

Exercise 3.2 Verify that $\boldsymbol{A}\boldsymbol{q}_k \in \mathsf{K}_{k+1}$. Show that $\boldsymbol{A}\boldsymbol{q}_k \notin \mathsf{K}_k$, unless $\mathsf{K}_k = \mathsf{K}_{k+1}$. Conclude that $\text{span}\{\boldsymbol{q}_1, \dots, \boldsymbol{q}_k\} = \mathsf{K}_k(\boldsymbol{A}; \boldsymbol{x})$ for each index k.

3.3 Lanczos iteration

The Arnoldi iteration applies to any square matrix, but something incredible happens when the matrix $\boldsymbol{A}$ is self-adjoint. This specialization leads to a simpler algorithm, called the *Lanczos iteration*, that has much lower computational cost than the Arnoldi iteration.

Lanczos iteration

Since $\boldsymbol{Q}_k$ is orthonormal for each k, we can left-multiply the equation (3.2) by $\boldsymbol{Q}_k^*$ to arrive at the formula

$$\boldsymbol{Q}_k^* \boldsymbol{A} \boldsymbol{Q}_k = \boldsymbol{H}_k,$$

where $\boldsymbol{H}_k$ is the top $k \times k$ square of $\widehat{\boldsymbol{H}}_k$; i.e. the coefficient matrix without its bottom row. In particular, $\boldsymbol{H}_k$ is also upper Hessenberg. Given that $\boldsymbol{A} = \boldsymbol{A}^*$, we discover that $\boldsymbol{H}_k$ is also self-adjoint:

$$\boldsymbol{H}_k = \boldsymbol{Q}_k^* \boldsymbol{A} \boldsymbol{Q}_k = \left(\boldsymbol{Q}_k^* \boldsymbol{A} \boldsymbol{Q}_k\right)^* = \boldsymbol{H}_k^*.$$

But then $\boldsymbol{H}_k$ and $\boldsymbol{H}_k^*$ are *both* upper Hessenberg. Therefore, the matrix $\boldsymbol{H}_k$ is actually *tridiagonal*. We change notation to emphasize this fact: *tridiagonal*

$$\boldsymbol{Q}_k^* \boldsymbol{A} \boldsymbol{Q}_k = \boldsymbol{T}_k = \begin{bmatrix} \alpha_1 & \beta_1 & & \\ \beta_1 & \ddots & & \\ & & \ddots & \beta_{k-1} \\ & & \beta_{k-1} & \alpha_k \end{bmatrix}.$$

A matrix with this special structure (symmetric and tridiagonal) is called a *Jacobi matrix*. *Jacobi matrix*

This structure allows for a significant reduction in computation complexity. The Arnoldi iteration orthogonalizes $\boldsymbol{A}\boldsymbol{q}_k$ against each of $\boldsymbol{q}_1, \dots, \boldsymbol{q}_k$. Since the coefficient matrix is tridiagonal, we see that (in exact arithmetic) the only nonzero inner products are the ones between $\boldsymbol{A}\boldsymbol{q}_k$ and $\boldsymbol{q}_k$ and $\boldsymbol{q}_{k-1}$. As a consequence, we can omit the remaining orthogonalization steps from the MGS procedure.

This simplification results in the Lanczos iteration; see Algorithm 3.1 for the details. The total cost of each iteration is $O(n)$ arithmetic operations. The algorithm computes the tridiagonal matrix $\boldsymbol{T}$ automatically, by explicitly determining the coefficients α_i and β_i. The total storage cost is just $O(n)$ if we only need the tridiagonal matrix. The storage cost rises to $O(kn)$ if we store the orthonormal basis computed after k iterations; we need this basis if we want to estimate eigenvectors as well as eigenvalues.

The matrix $\boldsymbol{T} = \boldsymbol{Q}^* \boldsymbol{A} \boldsymbol{Q}$ is the compression of the input matrix to the Krylov subspace—the second step of our eigenvalue computation procedure. To fulfill our task of estimating the largest eigenvalue $\lambda_{\max}(\boldsymbol{A})$, we simply need to find the maximum eigenvalue $\lambda_{\max}(\boldsymbol{T})$ of the tridiagonal matrix. This can be achieved with only $O(n)$ operations via the bisection method. For example, see [TB97, Lec. 30] or [GVL13, Sec. 8.4]. We omit the details, as this topic exceeds the scope of the lecture.

Exercise 3.3 (Minimum eigenvalue). Explain how to use the Lanczos iteration to estimate the minimum eigenvalue of a self-adjoint matrix.

Exercise 3.4 (Complex field). What, if anything, changes if we want to apply the Lanczos iteration to a complex self-adjoint matrix $\boldsymbol{A} \in \mathbb{H}_n(\mathbb{C})$?

Warning 3.5 The Lanczos iteration has complicated behavior in floating-point arithmetic. Nevertheless, it is reliable if we simply wish to estimate the maximum (or minimum) eigenvalue of a self-adjoint matrix. ▪

3.4 Evaluating a spectral function

We now make a slight digression to introduce the concept of a spectral function. The rest of the lecture will explain how the Lanczos method can be used to estimate quadratic forms of a spectral function.

Algorithm 3.1 LanczosTridiagonalization.
Warning: This algorithm exhibits complicated behaviors in floating-point arithmetic.

Input: Input matrix $\boldsymbol{A} \in \mathbb{H}_n$, unit vector $\boldsymbol{x} \in \mathbb{F}$, number k of steps
Output: Orthogonal matrix $\boldsymbol{Q} \in \mathbb{M}_k$ and tridiagonal matrix $\boldsymbol{T} \in \mathbb{H}_k$ such that $\boldsymbol{Q}^*\boldsymbol{A}\boldsymbol{Q} = \boldsymbol{T}$

```
function LanczosTridiagonalization(A; x; k)
    i = 0, β_0 = 1, q_1 = 0, r_0 = x
    while i < k and β_i ≠ 0 do
        q_{i+1} = r_i / β_i
        i = i + 1
        α_i = q_i^* (A q_i)
        r_i = (A − α_i I) q_i − β_{i−1} q_{i−1}
        β_i = ‖r_i‖
```

3.4.1 Spectral functions

We begin with an alternative presentation of the spectral theorem for self-adjoint matrices.

Definition 3.6 (Spectral resolution). Let $\boldsymbol{A} \in \mathbb{H}_n$ be a self-adjoint matrix. The *spectral resolution* of $\boldsymbol{A}$ is the decomposition *spectral resolution*

$$\boldsymbol{A} = \sum\nolimits_{i=1}^{m} \lambda_i \boldsymbol{P}_i \quad \text{where} \quad \sum\nolimits_{i=1}^{m} \boldsymbol{P}_i = \mathbf{I} \quad \text{and} \quad \boldsymbol{P}_i \boldsymbol{P}_j = \boldsymbol{P}_i \delta_{ij}.$$

Here, λ_i are the *distinct* real eigenvalues of $\boldsymbol{A}$, and the matrix $\boldsymbol{P}_i$ is the orthogonal projector onto the invariant subspace associated with the eigenvalue λ_i. In this context, δ_{ij} is the Kronecker delta.

Using the spectral resolution, we can define what it means to apply a (scalar) function to a self-adjoint matrix.

Definition 3.7 (Spectral function). Let $f : I \to \mathbb{R}$ be a real-valued function on an interval I of the real line. Let $\boldsymbol{A} \in \mathbb{H}_n$ be a self-adjoint matrix whose eigenvalues lie in the interval I. Then we define the *spectral function* *spectral function*

$$f(\boldsymbol{A}) = \sum\nolimits_{i=1}^{m} f(\lambda_i) \boldsymbol{P}_i \quad \text{where} \quad \boldsymbol{A} = \sum\nolimits_{i=1}^{m} \lambda_i \boldsymbol{P}_i$$

is a spectral resolution of $\boldsymbol{A}$.

Spectral functions arise in many different contexts. In fact, we have already seen one example: a matrix polynomial. There are many others.

Example 3.8 (Spectral functions). Here are some important instances of spectral functions.

- For $f(\lambda) = \lambda^{-1}$, the spectral function is $f(\boldsymbol{A}) = \boldsymbol{A}^{-1}$.
- For $f(\lambda) = \log \lambda$, the spectral function is $f(\boldsymbol{A}) = \log \boldsymbol{A}$.

Whenever we apply a scalar-valued function to a matrix, we are always referring to the associated spectral function. ∎

Computing a spectral function typically requires $O(n^3)$ operations because it involves a full eigenvalue decomposition. Nevertheless, we may wonder if it is possible to obtain partial information with less work.

3.4.2 Evaluating the quadratic form

One basic question is how we can evaluate individual entries of a spectral function $f(\boldsymbol{A})$ of a self-adjoint matrix $\boldsymbol{A}$. In particular, we may wish to determine the quadratic form $\boldsymbol{x}^* f(\boldsymbol{A})\boldsymbol{x}$. Among other things, this computation allows us to obtain individual diagonal entries of the spectral function. We will attempt to perform this computation using matrix–vector multiplications.

Computational Problem (Quadratic form in a spectral function). Let $f : I \to \mathbb{R}$ be a function. For a self-adjoint matrix $\boldsymbol{A} \in \mathbb{H}_n(\mathbb{R})$ and a vector $\boldsymbol{x} \in \mathbb{R}^n$, compute the quadratic form $\boldsymbol{x}^* f(\boldsymbol{A})\boldsymbol{x}$.

Computational Primitive (Matrix–vector multiplication). We can compute $\boldsymbol{u} \mapsto \boldsymbol{A}\boldsymbol{u}$ for each $\boldsymbol{u} \in \mathbb{R}^n$.

The following incredible approach emerged from early work by Gene Golub and John Welsch [GW69]. Connections and ramifications are explored in the book of Golub and Meurant [GM10]. See [GVL13, Sec. 10.2] for a simple introduction.

First, observe that the quadratic form $\boldsymbol{x}^* f(\boldsymbol{A})\boldsymbol{x}$ can be rewritten as an integral against a discrete measure:

$$\boldsymbol{x}^* f(\boldsymbol{A})\boldsymbol{x} = \sum\nolimits_{j=1}^{m} f(\lambda_j)\, \boldsymbol{x}^* \boldsymbol{P}_j \boldsymbol{x} = \int_{\mathbb{R}} f(\lambda)\, \mathrm{d}\nu(\lambda), \tag{3.3}$$

where ν is a weighted spectral measure

$$\nu = \sum\nolimits_{j=1}^{m} \tau_j \delta_{\lambda_j} \quad \text{and} \quad \tau_j = \boldsymbol{x}^* \boldsymbol{P}_j \boldsymbol{x} \geq 0.$$

In this context, δ_a is the Dirac measure at a.

This may not seem like progress—in fact, it seems that we are obfuscating something simple. The potential of this approach becomes clearer when we remember that it is possible to approximate integrals using quadratures; that is, by a weighted sum of function values.

Idea: Approximate the integral in (3.3) by Gaussian quadrature.

It is an astonishing fact that we can obtain the required quadrature rule efficiently using an application of the Lanczos procedure. This is what we will discuss next.

Exercise 3.9 (Off-diagonal entries). Explain how to compute the bilinear form $\boldsymbol{x}^* f(\boldsymbol{A})\boldsymbol{y}$ using two evaluations of the quadratic form in $f(\boldsymbol{A})$. **Hint:** Polarization!

3.5 Gaussian quadrature

Gaussian quadrature (GQ) rule

A *Gaussian quadrature (GQ) rule* is a collection of *weights* w_i and *nodes* θ_i that allow one to approximate the integral of a function with respect to a measure:

$$\int_{\mathbb{R}} f(\lambda)\, \mathrm{d}\mu(\lambda) \approx \sum\nolimits_{i=1}^{k} w_i f(\theta_i).$$

Remark 3.10 The weights and nodes depend on the measure μ, of course.

The GQ rule has the property that the approximation has zero error when the function f is a polynomial of degree at most $2k - 1$. The error in the GQ rule is controlled by the maximum value of the $(2k)$th derivative of the function f on an interval containing the support of the measure.

The simplest form of GQ was invented by Gauss, who developed this procedure to estimate integrals against the Lebesgue measure on $[-1, +1]$. This method is now called Gauss–Legendre quadrature to distinguish it from the continuum of GQ rules.

The construction of a GQ rule hinges on the existence and properties of orthogonal polynomials, and this is usually how Gauss–Legendre quadrature is derived in elementary numerical analysis texts. A complete treatment is beyond the scope of our lecture, so we will present three facts that justify the method, leaving the proofs as not-too-difficult exercises.

Fact 3.11 (Existence of orthogonal polynomials). For each positive measure μ, there exists a graded system $\{\varphi_0, \varphi_1, \dots\}$ of orthogonal polynomials, which may be finite or infinite. That is, $\deg \varphi_i = i$ for each index i, and

$$\int_{\mathbb{R}} \varphi_i(\lambda)\varphi_j(\lambda) \,\mathrm{d}\mu(\lambda) = \begin{cases} 1, & i = j, \\ 0, & i \neq j. \end{cases}$$

Moreover, the orthogonal polynomials satisfy a 3-term recurrence:

$$\gamma_k \varphi_k(\lambda) = (\lambda - \zeta_k)\varphi_{k-1}(\lambda) - \gamma_{k-1}\varphi_{k-2}(\lambda),$$

with initial conditions $\varphi_{-1}(\lambda) = 0$ and $\varphi_0(\lambda) = 1$. The coefficients γ_k are nonzero. ∎

Fact 3.12 (Roots of orthogonal polynomials). The zeros of the polynomial φ_k are real and distinct. Moreover, they coincide with the eigenvalues of the symmetric tridiagonal matrix

$$\boldsymbol{T}_k = \begin{bmatrix} \zeta_1 & \gamma_1 & & \\ \gamma_1 & \ddots & & \\ & & \ddots & \gamma_{k-1} \\ & & \gamma_{k-1} & \zeta_k \end{bmatrix}$$

∎

Fact 3.13 (Construction of GQ rule). Given an eigenvalue decomposition $\boldsymbol{T}_k = \boldsymbol{U}\boldsymbol{\Theta}\boldsymbol{U}^*$, the k-point GQ rule for the measure μ has nodes θ_i and weights $w_i = |u_{1i}|^2$. ∎

In the next section, we will explain how these facts are connected with the Lanczos iteration.

Problem 3.14 (Existence of orthogonal polynomials). Prove Fact 3.11 by applying the Gram–Schmidt procedure to the monomials. To obtain the three-term recurrence, note that $\langle p,\ \lambda q\rangle_\mu = \langle \lambda p,\ q\rangle_\mu$ where p, q are polynomials, λ is multiplication by the independent variable, and $\langle \cdot,\ \cdot \rangle_\mu$ is the $L_2(\mu)$ inner product.

Problem 3.15 (Roots of orthogonal polynomials). Prove Fact 3.12 by computing the characteristic polynomial of the tridiagonal matrix $\boldsymbol{T}_k$ recursively. Connect it with the 3-term recurrence for the orthogonal polynomials.

Problem 3.16 (Construction of the GQ rule). Let θ_i be the roots of the orthogonal polynomial φ_k. By solving a linear system, determine weights w_i for which the k-point Gauss rule exactly integrates all polynomials of degree $k - 1$ against the measure μ. Prove that the resulting quadrature rule integrates all polynomials of degree $2k - 1$ against the measure μ.

Problem 3.17 (Efficient construction of weights). Prove the rest of Fact 3.13, namely that we can obtain the weights w_i from the eigenvector decomposition of the tridiagonal matrix $\boldsymbol{T}_k$. **Hint:** This step requires the Christoffel–Darboux formulas for orthogonal polynomials; it is the most difficult part of this development.

3.6 Lanczos quadrature

To connect these facts with the Lanczos iteration, we need to show that the Lanczos iteration is implicitly computing orthogonal polynomials with respect to a measure. We also need to demonstrate that the tridiagonal matrix produced by Lanczos describes the 3-term recurrence of these polynomials. From there, the facts we have reported above show that the Lanczos iteration can be used to construct quadrature rules for the integral (3.3).

First, observe that each orthonormal vector $\boldsymbol{q}_i$ computed by the Lanczos iteration takes the form $\boldsymbol{q}_i = \varphi_i(\boldsymbol{A})\boldsymbol{x}$ for some polynomial φ_i with degree i. These polynomials are called *Lanczos polynomials*.

Lanczos polynomials

Next, we show that the 3-term recurrence that underlies the Lanczos iteration is the same as the 3-term recurrence for the Lanczos polynomials. The key step in the Lanczos iteration takes the form

$$\beta_k \boldsymbol{q}_{k+1} = (\boldsymbol{A} - \alpha_k \mathbf{I})\boldsymbol{q}_k - \beta_{k-1}\boldsymbol{q}_{k-1}.$$

This relation implies that

$$\beta_k \varphi_{k+1}(\lambda) = (\lambda - \alpha_k)\varphi_k(\lambda) - \beta_{k-1}\varphi_{k-1}(\lambda) \quad \text{for all } \lambda \in \mathbb{R}.$$

To check the implication, form the eigendecomposition $\boldsymbol{A} = \boldsymbol{V}\boldsymbol{D}\boldsymbol{V}^*$. Multiply the 3-term recurrence for the matrix polynomials by $\boldsymbol{V}^*$ on the left and by $\boldsymbol{V}$ on the right. Then use the fact that a polynomial of degree $k < m$ that equals zero at m points must equal zero everywhere. Here, m is the number of distinct eigenvalues of $\boldsymbol{A}$, which is also the maximum number of distinct Lanczos polynomials.

Finally, it is easy to check that these polynomials are orthogonal with respect to the spectral measure:

$$\begin{aligned}
\int \varphi_i(\lambda)\varphi_j(\lambda)\,\mathrm{d}\nu(\lambda) &= \sum\nolimits_{k=1}^{m} (\boldsymbol{x}^*\boldsymbol{P}_k\boldsymbol{x})\,\varphi_i(\lambda_k)\varphi_j(\lambda_k) \\
&= \boldsymbol{x}^* \left[\sum\nolimits_{k=1}^{m} \varphi_i(\lambda_k)\varphi_j(\lambda_k)\boldsymbol{P}_k\right]\boldsymbol{x} \\
&= \boldsymbol{x}^* \left[\left(\sum\nolimits_{k=1}^{m} \varphi_i(\lambda_k)\boldsymbol{P}_k\right)\left(\sum\nolimits_{\ell=1}^{m} \varphi_j(\lambda_\ell)\boldsymbol{P}_\ell\right)\right]\boldsymbol{x} \\
&= \boldsymbol{x}^* \left[\varphi_i(\boldsymbol{A})\varphi_j(\boldsymbol{A})\right]\boldsymbol{x} = \boldsymbol{q}_i^*\boldsymbol{q}_j = 0.
\end{aligned}$$

The first identity follows from the definition of the spectral measure. The third relation depends on the properties of the spectral resolution. The last equality is true whenever $i \neq j$, and we have used the assumption that $\boldsymbol{A}$ is self-adjoint.

As a consequence of these calculations and the facts stated in the last section, we can approximate the quadratic form $\boldsymbol{x}^* f(\boldsymbol{A})\boldsymbol{x}$ using the following procedure:

1. Apply k steps of Lanczos iteration to $\boldsymbol{A}$, starting at the vector $\boldsymbol{x}$, to obtain a tridiagonal matrix $\boldsymbol{T}_k$.
2. Compute the eigenvalue decomposition $\boldsymbol{T}_k = \boldsymbol{U}\boldsymbol{\Theta}\boldsymbol{U}^*$.
3. Approximate the quadratic form using the k-point Gauss–Lanczos quadrature rule:

$$\boldsymbol{x}^* f(\boldsymbol{A})\boldsymbol{x} \approx \sum\nolimits_{i=1}^{k} |u_{1i}|^2 f(\theta_i)$$

This approach is known as *Lanczos quadrature*. It is possible to compute explicit error bounds that allow us to halt the procedure when the order k of the quadrature rule is high enough. For specific functions, we can also use methods from approximation

Lanczos quadrature

theory to obtain *a priori* bounds on how large k should be. The arithmetic cost for computing the eigenvalue decomposition of a Jacobi matrix is $O(k^2)$ and it is easy to compute the arithmetic cost of the other steps.

3.7 Stochastic Lanczos quadrature

What is the connection to randomized linear algebra? By combining Lanczos quadrature with the randomized trace estimator, we can obtain estimates for the trace of a spectral function using only matrix–vector multiplies.

Computational Problem (Trace of a spectral function). For a real function f and a self-adjoint matrix $\boldsymbol{A}$, estimate $\operatorname{tr} f(\boldsymbol{A})$.

Computational Primitive (Matrix–vector multiplication). We can obtain $\boldsymbol{u} \mapsto \boldsymbol{A}\boldsymbol{u}$ for any vector $\boldsymbol{u}$.

Example 3.18 (Trace of a spectral function). The problem of computing the trace of a spectral function comes up in a wide range of applications.

1. In electronic structure calculations, we may need to estimate $\operatorname{tr} \boldsymbol{A}^{-1}$, the trace of the spectral function $f(\cdot) = (\cdot)^{-1}$.
2. In Gaussian process regression, we need to compute the log-determinant $\log\det \boldsymbol{A}$, which is the trace of the spectral function $f(\cdot) = \log(\cdot)$.

See references [BFG96; GM10; UCS17; Don+17; Fit+18] for more examples. ▪

With what we know, we arrive at a very natural approach to estimating the trace of a spectral function.

Idea: Combine a stochastic trace estimator with the Lanczos quadrature.

Here is a brief outline of how this procedure might work.

- Draw an isotropic random vector $\boldsymbol{\omega} \in \mathbb{R}^n$.
- Form $X = \boldsymbol{\omega}^* f(\boldsymbol{A})\boldsymbol{\omega} \approx \mathsf{LanczosQuadrature}(f, \boldsymbol{A}, \boldsymbol{\omega})$.
- Average k independent copies of X to obtain the trace estimate $\bar{X}_k \approx \operatorname{tr} f(\boldsymbol{A})$.

This approach is called *stochastic Lanczos quadrature*. Although the ideas can be traced to Golub's work [BFG96] in the 1990s, the method has reemerged in the last few years [UCS17; Don+17; Fit+18] as a powerful approach for solving large problems that arise in machine learning, e.g., Gaussian process regression.

stochastic Lanczos quadrature

Numerics 3.19 (SLQ). Implement stochastic Lanczos quadrature to estimate the log-determinant of the RBF kernel matrix. Perform suitable experiments to explore how the bias and variance of the estimates as a function of the number s of samples.

Lecture bibliography

[BFG96] Z. Bai, M. Fahey, and G. Golub. "Some large-scale matrix computation problems". In: volume 74. 1-2. TICAM Symposium (Austin, TX, 1995). 1996, pages 71–89. DOI: 10.1016/0377-0427(96)00018-0.

[Don+17] K. Dong et al. “Scalable Log Determinants for Gaussian Process Kernel Learning”. In: *Advances in Neural Information Processing Systems 30*. Edited by I. Guyon et al. Curran Associates, Inc., 2017, pages 6327–6337. URL: http://papers.nips.cc/paper/7212-scalable-log-determinants-for-gaussian-process-kernel-learning.pdf.

[Fit+18] J. K. Fitzsimons et al. “Improved stochastic trace estimators using mutually unbiased bases”. In: *Uncertainty in Artificial Intelligence: Proceedings of the Thirty-Fourth Conference*. Edited by A. Globerson and R. Silva. Monterey, CA: AUAI Press, 2018.

[GM10] G. H. Golub and G. Meurant. *Matrices, moments and quadrature with applications*. Princeton Series in Applied Mathematics. Princeton University Press, Princeton, NJ, 2010, pages xii+363.

[GVL13] G. H. Golub and C. F. Van Loan. *Matrix computations*. Fourth. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, Baltimore, MD, 2013, pages xiv+756.

[GW69] G. H. Golub and J. H. Welsch. “Calculation of Gauss quadrature rules”. In: *Math. Comp. 23 (1969), 221-230; addendum, ibid.* 23.106, loose microfiche suppl (1969), A1–A10. DOI: 10.2307/2004418.

[TB97] L. N. Trefethen and D. Bau III. *Numerical linear algebra*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1997, pages xii+361. DOI: 10.1137/1.9780898719574.

[UCS17] S. Ubaru, J. Chen, and Y. Saad. “Fast estimation of $\mathtt{tr}(f(A))$ via stochastic Lanczos quadrature”. In: *SIAM J. Matrix Anal. Appl.* 38.4 (2017), pages 1075–1099. DOI: 10.1137/16M1104974.

4. Matrix Approximation by Sampling

Date: 16 January 2020 **Scribe:** Jing Yu

A core problem in numerical linear algebra is to approximate a matrix by another matrix that enjoys more structure or is easier to construct. In this lecture, we introduce a simple and versatile approach to matrix approximation: matrix approximation by randomized sampling, or matrix Monte Carlo. We illustrate this method with a toy algorithm for approximating a redundant matrix product. It has many other applications in computational mathematics.

Agenda:

1. Empirical approximation
2. The Matrix Monte Carlo Theorem
3. Approximate matrix multiplication
4. Uniform sampling
5. Importance sampling

4.1 Empirical approximation of matrices

The empirical approximation method was developed by Maurey in the late 1970s to bound the covering numbers of a convex hull. The idea was first published in a paper of Carl [Car85] on approximation theory; see also [Pis80]. We begin with the main idea behind empirical approximation in the context of matrices, and then we explain how to analyze it.

4.1.1 The empirical approximation method

Let $\boldsymbol{B} \in \mathbb{F}^{m \times n}$ be a fixed matrix that we wish to approximate; it is sometimes called the *input matrix / target matrix*. Imagine that we have an additive decomposition of the form

input matrix / target matrix

$$\boldsymbol{B} = \sum\nolimits_{k=1}^{d} \boldsymbol{B}_k,$$

where each $\boldsymbol{B}_k \in \mathbb{F}^{m \times n}$ is "simple." For example, the summands might be sparse or low rank. We also suppose that we have a set of sampling probabilities $(p_1, \dots, p_d) \in \Delta_d$, where Δ_d is the probability simplex. It often requires some creativity to determine the right set of sampling probabilities for a specific approximation problem.

Define a random matrix $\boldsymbol{X} \in \mathbb{F}^{m \times n}$ such that

$$\mathbb{P}\left\{\boldsymbol{X} = p_k^{-1} \boldsymbol{B}_k\right\} = p_k \quad \text{for each} \quad k = 1, \dots, d.$$

Observe that $\boldsymbol{X}$ inherits structure of $\boldsymbol{B}_k$. For example, if each $\boldsymbol{B}_k$ is sparse, so is the random matrix $\boldsymbol{X}$. Clearly, $\boldsymbol{X}$ is an unbiased estimator of $\boldsymbol{B}$; that is, $\mathbb{E}\,\boldsymbol{X} = \boldsymbol{B}$. A single realization of $\boldsymbol{X}$ is typically a poor approximation of $\boldsymbol{B}$. But we can average many copies of $\boldsymbol{X}$ to reduce the variance of our estimate.

Doing so, we arrive at the *matrix Monte Carlo estimator*: *matrix Monte Carlo estimator*

$$\bar{\boldsymbol{X}}_s := \frac{1}{s}\sum\nolimits_{i=1}^{s} \boldsymbol{X}_i \quad \text{where } \boldsymbol{X}_i \sim \boldsymbol{X} \text{ are iid.}$$

Applying linearity of expectation, we can check that $\mathbb{E}\,\bar{\boldsymbol{X}}_s = \boldsymbol{B}$. Moreover, $\bar{\boldsymbol{X}}_s$ also inherits structure from the $\boldsymbol{B}_k$, provided that the number s of samples is small.

4.1.2 Approximation in the spectral norm

We are interested in how big s has to be to ensure that $\bar{\boldsymbol{X}}_s$ approximates $\boldsymbol{B}$ with respect to the spectral norm. More precisely, we want

$$\|\bar{\boldsymbol{X}}_s - \boldsymbol{B}\| \le \varepsilon. \tag{4.1}$$

The spectral norm error bound (4.1) has several desirable consequences:

- Control of linear functionals:

$$|\langle \boldsymbol{F},\ \bar{\boldsymbol{X}}_s\rangle - \langle \boldsymbol{F},\ \hat{\boldsymbol{B}}\rangle| \le \varepsilon\|\boldsymbol{F}\|_* \quad \text{for } \boldsymbol{F} \in \mathbb{F}^{m\times n}.$$

  Here, $\|\cdot\|_*$ denotes the nuclear norm (i.e., the sum of singular values).
- Control of singular values:

$$|\sigma_j(\bar{\boldsymbol{X}}_s) - \sigma_j(\boldsymbol{B})| \le \varepsilon \quad \text{for each index } j.$$

- Control of singular vectors: When $\sigma_j(\boldsymbol{B})$ is isolated from the spectrum of $\boldsymbol{B}$, the jth left/right singular vector of $\bar{\boldsymbol{X}}_s$ is close to the jth left/right singular vector of $\boldsymbol{B}$.

A detailed statement about the singular vectors is complicated; see [Bha97, Chap. VIII] for discussion and results.

Warning 4.1 (Frobenius-norm approximation). It is easier to prove theorems about the Frobenius-norm error than the spectral-norm error. Unfortunately, Frobenius-norm error bounds are can be vacuous because the size of the error is often similar to the norm of the matrix we are trying to approximate. Except in rare cases, we will use the spectral norm to measure errors. ▪

4.2 Matrix Monte Carlo Theorem

The matrix Bernstein inequality allows us to control spectral-norm errors of matrix Monte Carlo approximations. This powerful tool is the subject of Lecture 5. For now, we content ourselves with stating and exploring a corollary that allows us to analyze Matrix Monte Carlo estimators.

Theorem 4.2 (Matrix Monte Carlo). Let $\boldsymbol{B} \in \mathbb{F}^{m\times n}$ be a matrix we wish to approximate. Assume that $\boldsymbol{X} \in \mathbb{F}^{m\times n}$ is a random matrix such that

$$\mathbb{E}\,\boldsymbol{X} = \boldsymbol{B} \quad \text{and} \quad \|\boldsymbol{X}\| \le L \quad \text{almost surely.}$$

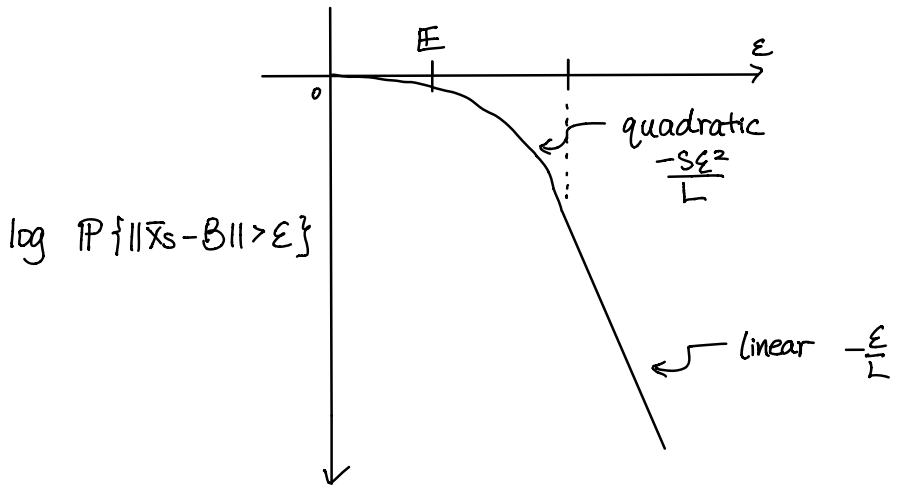


Figure 4.1 Log-probability that the error in the empirical approximation of a matrix exceeds a threshold ε.

Define the per-sample second moment

$$v(\boldsymbol{X}) = \max\left\{\|\mathbb{E}\left[\boldsymbol{X}\boldsymbol{X}^*\right]\|, \|\mathbb{E}\left[\boldsymbol{X}^*\boldsymbol{X}\right]\|\right\}.$$

Form $\bar{\boldsymbol{X}}_s = \frac{1}{s}\sum_{i=1}^{s} \boldsymbol{X}_i$, where $\boldsymbol{X}_i \sim \boldsymbol{X}$ iid. Then

$$\mathbb{E}\,\|\bar{\boldsymbol{X}}_s - \boldsymbol{B}\| \le \sqrt{\frac{2v(\boldsymbol{X})\log(m+n)}{s}} + \frac{2L\log(m+n)}{3s}.$$

For each $t > 0$,

$$\mathbb{P}\left\{\|\bar{\boldsymbol{X}}_s - \boldsymbol{B}\| > t\right\} \le (m+n)\cdot\exp\left(-\frac{st^2}{v(\boldsymbol{X}) + 2Lt/3}\right).$$

In particular, we have the following sample complexity bound.

$$s \ge \frac{2v(\boldsymbol{X})\log(m+n)}{\varepsilon^2} \vee \frac{2L\log(m+n)}{3\varepsilon} \quad \text{implies} \quad \mathbb{E}\,\|\bar{\boldsymbol{X}}_s - \boldsymbol{B}\| \le 2\varepsilon. \tag{4.2}$$

Recall that $\vee$ is infix notation for the maximum of two numbers.

The probability bound heralds a mixed tail behavior in the deviation parameter t. The statement is vacuous until t approaches the expectation value. This is followed by a regime of quadratic decay in the log probability (subgaussian decay) that transforms into a linear decay (subexponential tails) near the threshold $t = v(\boldsymbol{X})/L$. We refer to Figure 4.1 for an illustration.

Warning 4.3 (Sample complexity). The sample complexity bound (4.2) scales with ε^{-2}! Therefore, it is expensive to achieve high accuracy with Matrix Monte Carlo sampling. This is a fundamental drawback of any Monte Carlo sampling method. It is an unavoidable consequence of the central limit theorem. But see Problem 4.11 for ways to improve the situation. ∎

4.3 Application: Approximate matrix multiplication

As an illustration of the matrix Monte Carlo method, we will develop a toy algorithm for approximating a highly redundant matrix product.

Computational Problem (Matrix Product). Compute $\boldsymbol{B} = \boldsymbol{C}\boldsymbol{R}$ where $\boldsymbol{C} \in \mathbb{F}^{m\times d}$ and $\boldsymbol{R} \in \mathbb{F}^{n\times d}$ where the inner dimension $d \gg m \wedge n$. For simplicity, we assume for the rest of this chapter that $\|\boldsymbol{C}\| = \|\boldsymbol{R}\| = 1$.

We will pursue the following approach:

Idea: Approximate the product $\boldsymbol{B} = \boldsymbol{C}\boldsymbol{R}^*$ by matrix Monte Carlo sampling.

4.3.1 Monte Carlo for matrix products

To develop a sampling estimate for the matrix product $\boldsymbol{B} = \boldsymbol{C}\boldsymbol{R}^*$, we need to break it down into a sum of simple terms that we can compute easily. To that end, write out the columns of the factors:

$$\boldsymbol{C} = \begin{bmatrix} \boldsymbol{c}_1 & \cdots & \boldsymbol{c}_d \end{bmatrix} \quad \text{and} \quad \boldsymbol{R} = \begin{bmatrix} \boldsymbol{r}_1 & \cdots & \boldsymbol{r}_d \end{bmatrix}.$$

Then we can express the product as

$$\boldsymbol{B} = \sum\nolimits_{k=1}^{d} \boldsymbol{c}_k \boldsymbol{r}_k^* =: \sum\nolimits_{k=1}^{d} \boldsymbol{B}_k. \tag{4.3}$$

That is, each summand $\boldsymbol{B}_k$ is the rank-one matrix formed by the outer product of the kth column of $\boldsymbol{C}$ and the kth column of $\boldsymbol{R}$.

To form the Monte Carlo estimator, we need a set of sampling probabilities $(p_1, \dots, p_d)$; we will discuss some specific choices in the next two sections. Form the random matrix $\boldsymbol{X} \in \mathbb{F}^{m\times n}$ that satisfies

$$\mathbb{P}\left\{\boldsymbol{X} = p_k^{-1} \boldsymbol{c}_k \boldsymbol{r}_k^*\right\} = p_k \quad \text{for each } k = 1, \dots, d.$$

The associated matrix Monte Carlo estimator is the empirical average of s independent samples:

$$\bar{\boldsymbol{X}}_s = \frac{1}{s} \sum\nolimits_{i=1}^{s} \boldsymbol{X}_i \quad \text{where } \boldsymbol{X}_i \sim \boldsymbol{X} \text{ iid.}$$

This procedure produces an unbiased estimator of the matrix product: $\mathbb{E}\,\bar{\boldsymbol{X}}_s = \boldsymbol{B}$.

The arithmetic cost of forming the approximation $\bar{\boldsymbol{X}}_s$ is only $O(smn)$. When the number s of samples satisfies $s \ll d$, the Monte Carlo procedure may be faster than the naïve matrix product, which costs $O(dmn)$ operations. On the other hand, to achieve the speed-up, we must agree to suffer a substantial approximation error.

The actual number of samples s required to achieve a desired toleranace ε depends on the choice of sampling probabilities. We shall analyze two popular strategies: uniform sampling and importance sampling.

4.3.2 Uniform sampling

First, we consider the simplest possible approach, where we sample each term in the representation with equal probability. That is, $p_k = 1/d$ for each $k = 1, \dots, d$. Computing the uniform sampling distribution is free, and it does not require us to look at the matrix factors $\boldsymbol{C}, \boldsymbol{R}$.

The coherence statistic

To analyze uniform sampling, we need to introduce a measure of the uniformity of the columns of a matrix with respect to the standard basis.

Definition 4.4 (Coherence statistic). Let $\boldsymbol{C} \in \mathbb{F}^{m\times d}$ be a matrix with the normalization $\|\boldsymbol{C}\| = 1$. The *coherence statistic* is defined as *coherence statistic*

$$\mu(\boldsymbol{C}) := d \cdot \max_k \|\boldsymbol{c}_k\|^2.$$

Exercise 4.5 (Coherence statistic). Verify that $\mu(\boldsymbol{C}) \in [m, d]$. The lower bound is attained when all the columns of $\boldsymbol{C}$ have equal norm; the upper bound is attained when one column is a standard basis vector.

Analysis

In order to activate the Matrix Monte Carlo theorem, we need to compute the per-sample second moment $\nu(\boldsymbol{X})$ and provide a uniform upper bound $\|\boldsymbol{X}\| \le L$ for the sample matrix $\boldsymbol{X}$ constructed with the uniform sampling probabilities $p_k = 1/d$.

Observe that the spectral norm of $\boldsymbol{X}$ satisfies

$$\|\boldsymbol{X}\| \le \max_k \|p_k^{-1}\boldsymbol{c}_k\boldsymbol{r}_k^*\| \le d \cdot (\max_k \|\boldsymbol{c}_k\|)\,(\max_k \|\boldsymbol{r}_k\|) \le \mu(\boldsymbol{C}) \vee \mu(\boldsymbol{R}).$$

The bound $L = \mu(\boldsymbol{C}) \vee \mu(\boldsymbol{R})$ may seem rather crude, but it matches the scaling of the per-sample second moment.

Next, let us estimate the per-sample second moment. First,

$$\begin{aligned}
\mathbf{0} \preccurlyeq \mathbb{E}[\boldsymbol{X}\boldsymbol{X}^*] &= \sum\nolimits_{k=1}^{d} p_k \left(p_k^{-1}\boldsymbol{c}_k\boldsymbol{r}_k^*\right)\left(p_k^{-1}\boldsymbol{c}_k\boldsymbol{r}_k^*\right)^* \\
&= \sum\nolimits_{k=1}^{d} p_k^{-1}\|\boldsymbol{r}_k\|^2\boldsymbol{c}_k\boldsymbol{c}_k^* \\
&\preccurlyeq d \cdot \max_k \|\boldsymbol{r}_k\|^2 \sum\nolimits_{k=1}^{d} \boldsymbol{c}_k\boldsymbol{c}_k^* \\
&= \mu(\boldsymbol{R})(\boldsymbol{C}\boldsymbol{C}^*).
\end{aligned}$$

As usual, $\preccurlyeq$ denotes the psd order. A parallel computation reveals that

$$\mathbf{0} \preccurlyeq \mathbb{E}[\boldsymbol{X}^*\boldsymbol{X}] \preccurlyeq \mu(\boldsymbol{C})(\boldsymbol{R}\boldsymbol{R}^*).$$

Together, these psd order relations ensure

$$\begin{aligned}
\nu(\boldsymbol{X}) &= \|\mathbb{E}[\boldsymbol{X}\boldsymbol{X}^*]\| \vee \|\mathbb{E}[\boldsymbol{X}^*\boldsymbol{X}]\| \\
&\le \|\mu(\boldsymbol{R})(\boldsymbol{C}\boldsymbol{C}^*)\| \vee \|\mu(\boldsymbol{C})(\boldsymbol{R}\boldsymbol{R}^*)\| \\
&= \mu(\boldsymbol{R}) \vee \mu(\boldsymbol{C}).
\end{aligned}$$

The last step requires the assumption that $\|\boldsymbol{C}\| = \|\boldsymbol{R}\| = 1$.

We can now invoke the matrix Monte Carlo theorem. If

$$s \ge \left(\frac{1}{\varepsilon^2} \vee \frac{1}{3\varepsilon}\right)(\mu(\boldsymbol{C}) \vee \mu(\boldsymbol{R}))\log(m+n), \tag{4.4}$$

then

$$\frac{\mathbb{E}\,\|\bar{\boldsymbol{X}}_s - \boldsymbol{C}\boldsymbol{R}^*\|}{\|\boldsymbol{C}\|\|\boldsymbol{R}\|} \le 2\varepsilon.$$

This result gives us an estimate for the number s of samples we need to approximate the matrix product $\boldsymbol{B} = \boldsymbol{C}\boldsymbol{R}^*$ in terms of the coherence of the columns of the factors.

Discussion

If $\boldsymbol{C}$ and $\boldsymbol{R}$ have uniformly small columns, then $\mu(\boldsymbol{C}) \approx n$ and $\mu(\boldsymbol{R}) \approx m$. In this case, we only need $s \approx \varepsilon^2(m \vee n)\log(m+n)$ samples. The total arithmetic cost for the Monte Carlo approximation is $O\left((m^2 n \vee mn^2)\log(m+n)\right)$. When $d \gg m \vee n$, the cost of performing the approximation compares favorably with the $O(dmn)$ cost of naïve matrix multiplication.

4.3.3 Importance sampling

In the case where columns of $\boldsymbol{C}$ and $\boldsymbol{R}$ have disparate norms, it is natural to adjust the sampling scheme to compensate. Specifically, we consider the following importance sampling distribution

$$p_k = \frac{\|\boldsymbol{c}_k\|^2 + \|\boldsymbol{r}_k\|^2}{\|\boldsymbol{C}\|_F^2 + \|\boldsymbol{R}_k\|_F^2} \quad \text{for each } k = 1, \dots, d. \tag{4.5}$$

This construction has a natural interpretation: we sample the kth term in the decomposition (4.3) with probability proportional to a measure of its energy. The justification for the precise form of (4.5) comes from the fact that it controls the parameters that arise from the matrix Monte Carlo theorem.

Warning 4.6 The importance sampling distribution (4.5) does not come for free. Assuming that $\boldsymbol{C}$ and $\boldsymbol{R}$ are dense, the arithmetic cost is $O(d(m+n))$. We also need access to the columns of the factors. ▪

Stable rank

To analyze the performance of matrix Monte Carlo estimation with importance sampling, we recall the definition of the *stable rank*:

stable rank

$$\operatorname{srank}(\boldsymbol{C}) = \operatorname{intdim}(\boldsymbol{C}^*\boldsymbol{C}) = \frac{\|\boldsymbol{C}\|_F^2}{\|\boldsymbol{C}\|^2} \in [1, \operatorname{rank}(\boldsymbol{C})].$$

The stable rank already arose in our discussion of trace estimators.

Exercise 4.7 (Stable rank versus coherence). Under the assumption that $\|\boldsymbol{C}\| = 1$, check that $\operatorname{srank}(\boldsymbol{C}) \le \mu(\boldsymbol{C})$.

Analysis

One can show that importance sampling always outperforms uniform sampling. We leave the details of the computation to the reader.

Exercise 4.8 (Approximate matrix multiplication by importance sampling). Construct the sample matrix $\boldsymbol{X}$ using the importance sampling probabilities (4.5). Show that the per-sample second moment $v(\boldsymbol{X})$ and the uniform bound on the spectral norm $\|\boldsymbol{X}\| \le L$ satisfy the relation

$$L \vee v(\boldsymbol{X}) \le \frac{1}{2}(\operatorname{srank}(\boldsymbol{C}) + \operatorname{srank}(\boldsymbol{R})).$$

Hint: Do not forget that we imposed the normalization $\|\boldsymbol{C}\| = \|\boldsymbol{R}\| = 1$.

With the result from Exercise 4.8 at hand, the matrix Monte Carlo theorem tells us that

$$s \ge \left(\frac{1}{\varepsilon^2} \vee \frac{1}{3\varepsilon}\right)(\operatorname{srank}(\boldsymbol{C}) + \operatorname{srank}\boldsymbol{R})\log(m+n) \tag{4.6}$$

implies that

$$\frac{\mathbb{E}\,\|\bar{\boldsymbol{X}}_s - \boldsymbol{C}\boldsymbol{R}^*\|}{\|\boldsymbol{C}\|\|\boldsymbol{R}\|} \leq 2\varepsilon.$$

This result gives a bound for the number s of samples we need to approximate the product $\boldsymbol{B} = \boldsymbol{C}\boldsymbol{R}^*$ in terms of the stable ranks of the factors.

Discussion

Note that $\operatorname{srank}(\boldsymbol{C}) \leq m$ and $\operatorname{srank}(\boldsymbol{R}) \leq n$, so that the arithmetic cost of forming the approximation is always $O((mn^2 \vee m^2 n)\log(m+n))$. In fact, there are many matrices for which the stable rank is significantly smaller than the algebraic rank, and importance sampling works especially well for these instances. Recall, however, that we also pay $O(d(m+n))$ arithmetic to compute the importance sampling distribution, and it requires a separate pass over the data.

Exercise 4.7 implies that $\operatorname{srank}(\boldsymbol{C}) + \operatorname{srank}(\boldsymbol{R}) \leq \mu(\boldsymbol{C}) + \mu(\boldsymbol{R})$. Therefore, we may conclude that the bound (4.6) for the sample complexity of matrix multiplication by importance sampling always improves on the bound (4.4) for uniform sampling. The discrepancy is particularly large for matrices that are low-rank (the stable rank is small) and non-uniform columns (the coherence is large). Of course, it costs us some effort to compute the importance sampling probabilities, whereas the uniform sampling probabilities are free.

Problems

Exercise 4.9 (Subspace embeddings). Let $\boldsymbol{U} \in \mathbb{F}^{n\times k}$ be an orthonormal matrix; that is, $\boldsymbol{U}^*\boldsymbol{U} = \mathbf{I}_k$. We can interpret $\boldsymbol{U}$ as the basis for some k-dimensional subspace in $\mathbb{F}^n$. Suppose that we would like to approximate the product using the Monte Carlo matrix multiplication estimate $\bar{\boldsymbol{X}}_s$.

1. If we use uniform sampling, how many samples do we need to guarantee that $\mathbb{E}\,\|\bar{\boldsymbol{X}}_s - \mathbf{I}\| \leq \varepsilon$? What summary statistic of $\boldsymbol{U}$ determines the behavior of the approximation?
2. Repeat the last part for importance sampling. How much does it cost us to get the sampling probabilities? Do we need to know $\boldsymbol{U}$ to construct the sampling distribution?
3. Suppose that we sample a set S of coordinates and achieve error ε in the approximate matrix product. What can we say about the singular values of the submatrix $\boldsymbol{U}(\mathsf{S},:)$? How does $\|\boldsymbol{U}(\mathsf{S},:)\boldsymbol{x}\|$ compare with $\|\boldsymbol{x}\|$? How do you interpret the last observation?
4. Implement both procedures. Apply them to (i) the matrix obtained by stacking n/k copies of the identity matrix $\mathbf{I}_k$ and rescaling; and (ii) the matrix obtained from the first k columns of the unitary DFT matrix. Plot the sampling distribution of the spectral-norm error as a function of the number s of samples. How variable is the error?

Problem 4.10 (Sparsification). Let $\boldsymbol{B} \in \mathbb{F}^{m\times n}$ be a matrix. In this problem, we will design and analyze a Monte Carlo method for approximating $\boldsymbol{B}$ by a sparse matrix. Use the Matrix Monte Carlo theorem to make short work of the computations.

1. Express $\boldsymbol{B}$ as a sum of mn matrices, each with one nonzero entry.
2. Show how to construct an unbiased estimator $\boldsymbol{X}$ of $\boldsymbol{B}$ via uniform sampling.
3. For $\varepsilon \in [0,1]$, give an upper bound on the number s of uniform samples needed to obtain $\mathbb{E}\,\|\bar{\boldsymbol{X}}_s - \boldsymbol{B}\| \leq 2\varepsilon\|\boldsymbol{B}\|$. Interpret the result in words.

4 Importance sampling works better. Define the probability mass

$$p_{ij} = \frac{1}{2}\left[\frac{|b_{ij}|^2}{\|\boldsymbol{B}\|_{\mathrm{F}}^2} + \frac{|b_{ij}|}{\|\boldsymbol{B}\|_{\ell_1}}\right] \quad \text{for } i = 1, \dots, m \text{ and } j = 1, \dots, n.$$

Here, $\|\cdot\|_{\ell_1}$ is the entrywise ℓ_1 norm. Analyze the resulting matrix Monte Carlo estimator to obtain a bound on the number s of samples needed to achieve $\mathbb{E}\,\|\bar{\boldsymbol{X}}_s - \boldsymbol{B}\| \leq 2\varepsilon\|\boldsymbol{B}\|$. Express the result in terms of the stable rank, and give an interpretation.

5 Implement both procedures, and apply them to the RBF kernel matrix associated with a dataset. Plot the sampling distribution of the spectral-norm error as a function of the number s of samples.

Problem 4.11 (Variance reduction). We designed the empirical matrix approximation procedure to produce an unbiased estimate of the target matrix $\boldsymbol{B}$ by a pure iid sampling method. This approach can result in an estimate with very high variance, so the matrix Monte Carlo theorem will demand an unduly large sample complexity s.

There are two mechanisms for correcting this behavior. First, we can force important terms $\boldsymbol{B}_k$ to appear in our approximation (those that would have large p_k), only using sampling for less important summands. Second, we can omit unimportant terms $\boldsymbol{B}_k$ entirely (those with very small p_k), not sampling them at all.

1 Develop a variant of the approximate matrix multiplication procedure that incorporates both of these improvements. How do we define an "important" term? How do we define an "unimportant" term? Give an explicit analysis of your procedure. Implement it and compare it with basic importance sampling for appropriate instances.

2 Do the same thing for randomized sparsification.

Lecture bibliography

[Bha97] R. Bhatia. *Matrix analysis*. Volume 169. Graduate Texts in Mathematics. Springer-Verlag, New York, 1997, pages xii+347. DOI: 10.1007/978-1-4612-0653-8.

[Car85] B. Carl. "Inequalities of Bernstein-Jackson-type and the degree of compactness of operators in Banach spaces". In: *Ann. Inst. Fourier (Grenoble)* 35.3 (1985), pages 79–118. URL: http://www.numdam.org/item?id=AIF_1985__35_3_79_0.

[Pis80] G. Pisier. "Remarques sur un résultat non publié de B. Maurey". fr. In: *Séminaire d'Analyse fonctionnelle (dit "Maurey-Schwartz")* (1980-1981). talk:5. URL: http://www.numdam.org/item/SAF_1980-1981____A5_0/.

5. Matrix Concentration

Date: 21 January 2020 **Scribe:** Nikola Kovachki

In the previous lecture, we analyzed a method for approximate matrix multiplication based on random sampling. The key theoretical tool was the Matrix Monte Carlo theorem. In this lecture, we will explain how to derive results about the concentration of random matrices. We will start with basic scalar concentration results and draw an analogy to the matrix setting using deep methods from matrix analysis. This development culminates in the Matrix Bernstein inequality. The matrix Monte Carlo theorem is an easy corollary of this result.

Agenda:

1. Scalar concentration
2. Matrix concentration
3. Matrix Bernstein
4. Rectangular case

5.1 Scalar concentration

Consider an independent family of random numbers $\{X_1, \dots, X_s\} \subset \mathbb{R}$, and form their sum:

$$Y = \sum_{i=1}^{s} X_i.$$

We wish to bound the probability that the random sum Y exceeds a level t:

$$\mathbb{P}\{Y \geq t\}.$$

Using the strict monotonicity of the exponential function,

$$\mathbb{P}\{Y \geq t\} = \mathbb{P}\left\{e^{\theta Y} \geq e^{\theta t}\right\} \leq e^{-\theta t}\, \mathbb{E}[e^{\theta Y}] \quad \text{for any } \theta > 0. \tag{5.1}$$

The last inequality is Markov's. We have replaced the problem of bounding the probability with the probability of bounding the *moment generating function (mgf)*:

moment generating function (mgf)

$$m_Y(\theta) = \mathbb{E}[e^{\theta Y}].$$

Since the factors X_i are statistically independent, the mgf factorizes:

$$m_Y(\theta) = \mathbb{E}\, e^{\theta \sum_{i=1}^{s} X_i} = \mathbb{E} \prod_{i=1}^{s} e^{\theta X_i} = \prod_{i=1}^{s} \mathbb{E}\, e^{\theta X_i} = \prod_{i=1}^{s} m_{X_i}(\theta)$$

The logarithm of the mgf is called the *cumulant generating function (cgf)*, and we have seen that the cgf obeys

cumulant generating function (cgf)

$$\log m_Y(\theta) = \sum\nolimits_{i=1}^{s} \log m_{X_i}(\theta).$$

We can now use the bound (5.1) to determine that

$$\mathbb{P}\{Y \geq t\} \leq \exp\left(-\theta t + \sum\nolimits_{i=1}^{s} \log m_{X_i}(\theta)\right).$$

This claim is valid for all $\theta > 0$, so we can take the infimum of the right-hand side over θ to obtain the tightest bound.

Similarly, we can obtain a bound for the lower tail of the sum by noting that

$$\mathbb{P}\{Y \leq t\} = \mathbb{P}\{-Y \geq -t\} \leq \inf_{\theta<0} \exp\left(-\theta t + \sum\nolimits_{i=1}^{s} \log m_{X_i}(\theta)\right).$$

Note that the infimum takes place over *negative* values of the parameter θ.

5.2 Matrix concentration

The methods outlined in Section 5.1 are among the most familiar and powerful tools from probability theory. In this section, we will execute an audacious and powerful extension of this approach. We will figure out how to develop a variant of the Laplace transform method that applies to an independent sum of random matrices.

5.2.1 The matrix Laplace transform method

Consider an independent family $\{\boldsymbol{X}_1, \dots, \boldsymbol{X}_s\} \subset \mathbb{H}_n$ of random self-adjoint matrices, and form the sum

$$\boldsymbol{Y} = \sum\nolimits_{i=1}^{s} \boldsymbol{X}_i.$$

The restriction to self-adjoint matrices is a natural generalization of real-valued random variables. We will discuss extensions to rectangular matrices below (Section 5.4).

We want to obtain the probability that the matrix $\boldsymbol{Y}$ is "large and positive." A natural way to quantify the positive skew of a self-adjoint matrix is by passing to its largest eigenvalue. As in the scalar setting, Markov's inequality asserts

$$\mathbb{P}\{\lambda_{\max}(\boldsymbol{Y}) \geq t\} = \mathbb{P}\left\{\mathrm{e}^{\theta\lambda_{\max}(\boldsymbol{Y})} \geq \mathrm{e}^{\theta t}\right\} \leq \mathrm{e}^{-\theta t}\,\mathbb{E}\,\mathrm{e}^{\theta\lambda_{\max}(\boldsymbol{Y})} \quad \text{for all } \theta > 0.$$

Now, things start to deviate from the scalar case, but the underlying intuition persists. In particular, we can use the spectral mapping theorem to conclude that

$$\mathbb{P}\{\lambda_{\max}(\boldsymbol{Y}) \geq t\} \leq \mathbb{E}\,\lambda_{\max}(\mathrm{e}^{\theta\boldsymbol{Y}}) \leq \mathrm{e}^{-\theta t}\,\mathbb{E}\,\mathrm{tr}(\mathrm{e}^{\theta\boldsymbol{Y}}). \tag{5.2}$$

Indeed, the exponential of the maximum eigenvalue of $\theta\boldsymbol{Y}$ coincides with the maximum eigenvalue of the spectral function $\mathrm{e}^{\theta\boldsymbol{Y}}$ because the exponential is strictly increasing. The last step is valid because the matrix $\mathrm{e}^{\theta\boldsymbol{Y}}$ is positive definite.

The last inequality may seem unnecessary, but it will allow us to exploit some remarkable properties of the matrix exponential function. Moreover, the bounds that result from this method are sharp—including the constants—for certain examples.

5.2.2 Subbaditivity of matrix cumulants

Let us introduce the *matrix mgf* of the sum: *matrix mgf*

$$m_{\boldsymbol{Y}} = \mathbb{E}\,\mathrm{tr}\exp(\theta \boldsymbol{Y}) = \mathbb{E}\,\mathrm{tr}\exp\left(\theta \sum\nolimits_{i=1}^{s} \boldsymbol{X}_i\right)$$

We now need some way of decomposing the matrix mgf into its individual constituents. A straightforward generalization of the scalar argument is impossible, because matrix exponentials do not factorize in general:

$$\mathrm{e}^{\boldsymbol{A}+\boldsymbol{B}} \neq \mathrm{e}^{\boldsymbol{A}}\mathrm{e}^{\boldsymbol{B}} \quad \text{for } \boldsymbol{A}, \boldsymbol{B} \in \mathbb{H}_n \text{ unless they commute.}$$

One possible substitute is to apply the Golden–Thompson inequality:

$$\mathrm{tr}(\mathrm{e}^{\boldsymbol{A}+\boldsymbol{B}}) \leq \mathrm{tr}(\mathrm{e}^{\boldsymbol{A}}\mathrm{e}^{\boldsymbol{B}}).$$

Although this seems like a promising way to bypass the issue, the relation does not necessarily extend to three or more matrices. That is,

$$\mathrm{tr}(\mathrm{e}^{\boldsymbol{A}+\boldsymbol{B}+\boldsymbol{C}}) \not\leq \mathrm{tr}(\mathrm{e}^{\boldsymbol{A}}\mathrm{e}^{\boldsymbol{B}}\mathrm{e}^{\boldsymbol{C}}).$$

An example of where this fails is obtained from the set of three Pauli matrices, a family of anti-commuting self-adjoint matrices that features prominently in quantum information [Bha97, Prob. IX.8.4]. There is a more conceptual reason that this kind of bound is impossible: the matrix on the right-hand side can have complex eigenvalues.

Despite this difficulty, Ahlswede and Winter [AW02] managed to iteratively apply the Golden–Thompson inequality to factorize the matrix mgf step by step. Using this method, they derived a matrix concentration inequality of Chernoff type. Their analysis, while beautiful, leads to suboptimal results.

Here, we will pursue another approach, introduced by your lecturer [Tro12]. This approach is based on the *matrix cgf*, the logarithm of the matrix mgf. Unfortunately, taking logarithms does not immediately solve the problem: *matrix cgf*

$$\log \mathbb{E}\,\mathrm{e}^{\theta \boldsymbol{Y}} \neq \sum\nolimits_{i=1}^{s} \log \mathbb{E}\,\mathrm{e}^{\theta \boldsymbol{X}_i}.$$

Nevertheless, the sum of the matrix cgfs remains self-adjoint, which suggests that we might be able to do something.

To proceed, we require a deep theorem from matrix analysis, established by Elliott Lieb [Lie73].

Fact 5.1 (Lieb 1973). For each fixed $\boldsymbol{H} \in \mathbb{H}_n$, the function

$$\boldsymbol{A} \mapsto \mathrm{tr}\exp(\boldsymbol{H} + \log \boldsymbol{A})$$

is concave on the cone of positive semidefinite matrices. ∎

In the scalar case, the corresponding function $a \mapsto a\mathrm{e}^h$ is linear. Lieb's theorem identifies a new phenomenon that takes place in the matrix setting. See [Tro15, Chap. 8] for a complete proof of Fact 5.1 from first principles.

Lieb's theorem, Fact 5.1, immediately supplies the following corollaries.

Corollary 5.2 (Tropp 2012). Fix $\boldsymbol{H} \in \mathbb{H}_n$, and let $\boldsymbol{X} \in \mathbb{H}_n$ be a random matrix. Then

$$\mathbb{E}_{\boldsymbol{X}}\,\mathrm{tr}\exp(\boldsymbol{H} + \boldsymbol{X}) \leq \mathrm{tr}\exp(\boldsymbol{H} + \log \mathbb{E}\,\mathrm{e}^{\boldsymbol{X}}).$$

Proof. Since the logarithm is the inverse of the exponential,

$$\mathbb{E}_{\boldsymbol{X}} \operatorname{tr} \exp(\boldsymbol{H} + \boldsymbol{X}) = \mathbb{E}_{\boldsymbol{X}} \operatorname{tr} \exp(\boldsymbol{H} + \log \mathrm{e}^{\boldsymbol{X}}).$$

Since the function is concave in $\boldsymbol{X}$, we can use Jensen's inequality to draw the expectation inside the logarithm. ∎

By iterating the argument in Corollary 5.2, we arrive at the following important result.

Corollary 5.3 (Subadditivity of matrix cgfs). Consider an independent family $\{\boldsymbol{X}_1, \dots, \boldsymbol{X}_s\} \subset \mathbb{H}_n$ of random matrices. Then

$$\mathbb{E} \operatorname{tr} \exp\left(\sum\nolimits_{i=1}^{s} \boldsymbol{X}_i\right) \le \operatorname{tr} \exp\left(\sum\nolimits_{i=1}^{s} \log \mathbb{E}\, \mathrm{e}^{\boldsymbol{X}_i}\right).$$

Equivalently,

$$\operatorname{tr} \exp\left(\log \mathbb{E} \exp\left(\sum\nolimits_{i=1}^{s} \boldsymbol{X}_i\right)\right) \le \operatorname{tr} \exp\left(\sum\nolimits_{i=1}^{s} \log \mathbb{E}\, \mathrm{e}^{\boldsymbol{X}_i}\right).$$

Corollary 5.3 is our substitute for the additivity law for scalar cgfs. To obtain the equivalence between the first and the second display, simply note that the exponential is the inverse function of the logarithm on the class of positive-definite matrices.

5.2.3 Master inequalities

We are now ready to state the key result of this lecture [Tro12].

Theorem 5.4 (Tropp 2012). Let $\boldsymbol{Y} = \sum_{i=1}^{s} \boldsymbol{X}_i$ be a sum of independent random matrices. For all $t \in \mathbb{R}$,

$$\mathbb{P}\{\lambda_{\max}(\boldsymbol{Y}) \ge t\} \le \inf_{\theta > 0} \mathrm{e}^{-\theta t} \operatorname{tr} \exp\left(\sum\nolimits_{i=1}^{s} \log \mathbb{E}\, \mathrm{e}^{\theta \boldsymbol{X}_i}\right). \tag{5.3}$$

Furthermore,

$$\mathbb{E}\, \lambda_{\max}(\boldsymbol{Y}) \le \inf_{\theta > 0} \frac{1}{\theta} \log \operatorname{tr} \exp\left(\sum\nolimits_{i=1}^{s} \log \mathbb{E}\, \mathrm{e}^{\theta \boldsymbol{X}_i}\right).$$

Proof. This result is an immediate consequence of the matrix Laplace transform inequality (5.2) and Corollary 5.3 on the subadditivity of cumulants. ∎

Exercise 5.5 (Expectation bound). Explain how to derive the inequality for the expectation in Theorem 5.4.

Exercise 5.6 (Minimum eigenvalue). Derive an analog of Theorem 5.4 for $\lambda_{\min}(\boldsymbol{Y})$. **Hint:** $\lambda_{\min}(\boldsymbol{Y}) = -\lambda_{\max}(-\boldsymbol{Y})$.

5.3 The Matrix Bernstein inequality

The master inequality (5.3) is general but abstract. To derive more applicable results, we need to exploit properties of the random matrices $\boldsymbol{X}_i$ to bound the matrix cgfs $\log \mathbb{E}\, \mathrm{e}^{\theta \boldsymbol{X}_i}$. To execute this approach, we simply adapt existing techniques for bounding the cgf of a real random variable.

As a concrete example, we shall prove the matrix Bernstein inequality in the self-adjoint case. This is perhaps the single most useful matrix concentration theorem.

5.3.1 The Bernstein matrix cgf bound

We begin with a lemma about the matrix cgf.

Lemma 5.7 (Bernstein cgf bound). Assume that $\boldsymbol{X} \in \mathbb{H}_n$ is a random matrix that satisfies

$$\mathbb{E}\,\boldsymbol{X} = \mathbf{0} \quad \text{and} \quad \|\boldsymbol{X}\| \le 1 \quad \text{almost surely.}$$

For all $\theta \in \mathbb{R}$,

$$\log \mathbb{E}\, \mathrm{e}^{\theta \boldsymbol{X}} \preccurlyeq g(\theta)(\mathbb{E}\,\boldsymbol{X}^2) \quad \text{where} \quad g(\theta) = \frac{\theta^2/2}{1 - |\theta|/3}.$$

Proof. The result follows from a natural extension of the scalar argument. For each $x \in [-1, 1]$, a Taylor expansion of the exponential gives

$$\begin{aligned} \mathrm{e}^{\theta x} &= 1 + \theta x + \sum\nolimits_{p=2}^{\infty} \frac{\theta^p}{p!} x^p \le 1 + \theta x + \frac{\theta^2 x^2}{2} \sum\nolimits_{p=2}^{\infty} \frac{|\theta|^{p-2}}{3^{p-2}} \\ &= 1 + \theta x + \frac{\theta^2 x^2}{2} \cdot \frac{1}{1 - |\theta|/3} = 1 + \theta x + g(\theta) x^2. \end{aligned}$$

This elementary argument extends directly to self-adjoint matrices. Let $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^*$ be an eigenvalue decomposition. The requirement that $\|\boldsymbol{X}\| \le 1$ ensures that each (random) eigenvalue λ_i of the matrix $\boldsymbol{X}$ obeys the bound $|\lambda_i| \le 1$. Hence,

$$\mathrm{e}^{\theta \boldsymbol{\Lambda}} \preccurlyeq \mathbf{I} + \theta \boldsymbol{\Lambda} + g(\theta) \boldsymbol{\Lambda}^2 \quad \text{where } \boldsymbol{\Lambda} = \operatorname{diag}(\lambda_1, \dots, \lambda_n).$$

Since the psd order is preserved by unitary conjugation,

$$\mathrm{e}^{\theta \boldsymbol{X}} = \boldsymbol{U} \mathrm{e}^{\theta \boldsymbol{\Lambda}} \boldsymbol{U}^* \preccurlyeq \mathbf{I} + \theta \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^* + g(\theta) \boldsymbol{U}\boldsymbol{\Lambda}^2\boldsymbol{U}^* = \mathbf{I} + \theta \boldsymbol{X} + g(\theta) \boldsymbol{X}^2.$$

Since the psd cone is convex, the psd order is preserved by expectations. Thus,

$$\mathbb{E}\, \mathrm{e}^{\theta \boldsymbol{X}} \preccurlyeq \mathbf{I} + \theta\, \mathbb{E}\, \boldsymbol{X} + g(\theta)\, \mathbb{E}\, \boldsymbol{X}^2 \preccurlyeq \exp(g(\theta)\, \mathbb{E}\, \boldsymbol{X}^2).$$

The last inequality is (the matrix extension of) the numerical relation $1 + a \le \mathrm{e}^a$. To conclude, we invoke the nontrivial fact that the logarithm preserves the psd order. ∎

Problem 5.8 (Logarithm is operator monotone). Suppose that $\mathbf{0} \prec \boldsymbol{A} \preccurlyeq \boldsymbol{B}$. Prove that $\log \boldsymbol{A} \preccurlyeq \log \boldsymbol{B}$. **Hint:** Use an integral representation of the logarithm:

$$\log a = \int_0^{\infty} \left[(1 + u)^{-1} - (a + u)^{-1} \right] \mathrm{d}u \quad \text{for } a > 0.$$

You also need to argue that the negative inverse preserves the psd order, but this is more straightforward.

5.3.2 Matrix Bernstein: Self-adjoint case

With the matrix cgf bound at hand, the matrix Bernstein inequality follows as an easy consequence of the master tail bound, Theorem 5.4.

Theorem 5.9 (Matrix Bernstein: Self-adjoint case). Consider an independent family $\{\boldsymbol{X}_1, \dots, \boldsymbol{X}_s\} \subset \mathbb{H}_n$ of random matrices. Assume that

$$\mathbb{E}\,\boldsymbol{X}_i = \mathbf{0} \quad \text{and} \quad \|\boldsymbol{X}_i\| \le L \quad \text{almost surely.}$$

Let $\boldsymbol{Y} = \sum_{i=1}^{s} \boldsymbol{X}_i$, and define the variance proxy

$$\sigma^2 = \|\mathbb{E}\,\boldsymbol{Y}^2\| = \left\|\sum\nolimits_{i=1}^{s} \mathbb{E}\,\boldsymbol{X}_i^2\right\|.$$

For all $t > 0$,

$$\mathbb{P}\{\lambda_{\max}(\boldsymbol{Y}) \geq t\} \leq n \cdot \exp\left(\frac{-t^2/2}{\sigma^2 + Lt/3}\right).$$

Furthermore,

$$\mathbb{E}\,\lambda_{\max}(\boldsymbol{X}) \leq \sqrt{2\sigma^2 \log n} + \frac{1}{3} L \log n.$$

When $n = 1$, this result collapses to the scalar Bernstein inequality. We have not lost anything by extending to the matrix setting. When the dimension $n > 1$, we only pay very weakly for the dimension—which is why this result is so powerful.

Proof. By homogeneity, we may rescale the matrices so that $L = 1$. Theorem 5.4 implies that

$$\begin{aligned}
\mathbb{P}\{\lambda_{\max}(\boldsymbol{Y}) \geq t\} &\leq \inf_{\theta>0} \mathrm{e}^{-\theta t} \operatorname{tr} \exp\left(\sum\nolimits_{i=1}^{s} \log \mathbb{E}\,\mathrm{e}^{\theta \boldsymbol{X}_i}\right) \\
&\leq \inf_{\theta>0} \mathrm{e}^{-\theta t} \operatorname{tr} \exp\left(g(\theta) \sum\nolimits_{i=1}^{s} \mathbb{E}\,\boldsymbol{X}_i^2\right) \\
&\leq n \cdot \inf_{\theta>0} \mathrm{e}^{-\theta t} \lambda_{\max}(\exp(g(\theta)\,\mathbb{E}\,\boldsymbol{Y}^2) \\
&= n \cdot \inf_{\theta>0} \mathrm{e}^{-\theta t} \exp(g(\theta)\sigma^2).
\end{aligned}$$

The second inequality holds because the trace exponential is monotone with respect to the semidefinite order. The third inequality holds because the trace of a pd matrix is at most the dimension times the maximum eigenvalue. The last relation follows from spectral mapping and the definition of σ^2.

To complete the proof, we cleverly select $\theta = t/(\sigma^2 + t/3)$. This is not the optimal value, but it results in a clean bound. ∎

Exercise 5.10 (Trace exponential is operator monotone). Suppose that $\boldsymbol{A} \preccurlyeq \boldsymbol{B}$. Show that $\operatorname{tr} \mathrm{e}^{\boldsymbol{A}} \leq \operatorname{tr} \mathrm{e}^{\boldsymbol{B}}$.

Exercise 5.11 (Expectation bound). Prove the expectation bound in Theorem 5.9.

Problem 5.12 (Intrinsic dimension). The dimensional factor n in the matrix Bernstein inequality can be reduced to $4\,\operatorname{intdim}(\mathbb{E}\,\boldsymbol{Y}^2)$. Prove it.

5.4 Matrix Bernstein: Rectangular case

Surprisingly, we can extend the matrix Bernstein inequality to rectangular matrices as an easy corollary. The key idea is to extend the rectangular matrices to self-adjoint matrices so that we can apply Theorem 5.9.

Corollary 5.13 (Matrix Bernstein inequality: Rectangular case). Consider an independent family $\{\boldsymbol{Z}_1, \dots, \boldsymbol{Z}_s\} \subset \mathbb{F}^{m \times n}$ of random rectangular matrices. Assume that

$$\mathbb{E}\,\boldsymbol{Z}_i = \boldsymbol{0} \quad \text{and} \quad \|Z_i\| \leq L \quad \text{almost surely.}$$

Let $\boldsymbol{S} = \sum_{i=1}^{s} \boldsymbol{Z}_i$, and define the variance proxy

$$\sigma^2 = \|\mathbb{E}[\boldsymbol{S}^*\boldsymbol{S}]\| \vee \|\mathbb{E}[\boldsymbol{S}\boldsymbol{S}^*]\|.$$

For all $t > 0$,

$$\mathbb{P}\{\|\boldsymbol{S}\| \geq t\} \leq (m+n)\exp\left(\frac{-t^2/2}{\sigma^2 + Lt/3}\right).$$

Furthermore,

$$\mathbb{E}\|\boldsymbol{S}\| \leq \sqrt{2\sigma^2 \log(m+n)} + \frac{1}{3}L\log(m+n).$$

The only price we have paid for passing to the rectangular case is that we pay for the *sum* of the two dimensions.

Proof. Apply Theorem 5.9 to the self-adjoint dilation

$$\boldsymbol{Y} = \begin{bmatrix} \mathbf{0} & \boldsymbol{S} \\ \boldsymbol{S}^* & \mathbf{0} \end{bmatrix} = \sum\nolimits_{i=1}^{s} \begin{bmatrix} \mathbf{0} & \boldsymbol{Z}_i \\ \boldsymbol{Z}_i^* & \mathbf{0} \end{bmatrix} \in \mathbb{H}_{n+m}.$$

To complete the argument, observe that the self-adjoint dilation preserves spectral properties. In particular,

$$\lambda_{\max}\left(\begin{bmatrix} \mathbf{0} & \boldsymbol{S} \\ \boldsymbol{S}^* & \mathbf{0} \end{bmatrix}\right) = \|\boldsymbol{S}\|.$$

The remaining calculations are straightforward. ∎

Exercise 5.14 (Matrix Monte Carlo). Derive the matrix Monte Carlo theorem, Theorem 4.2, from Corollary 5.13.

Exercise 5.15 (Intrinsic dimension). We can establish a version of Corollary 5.13 where the sum $(m+n)$ of dimensions is replaced by a kind of intrinsic dimension quantity. How does this work out?

Problems

Problem 5.16 (Matrix Chernoff). This problem contains a derivation of another fundamental matrix concentration inequality and an application in statistics. This version of the matrix Chernoff bound is different in spirit than the results that appear in the literature, such as [Tro15, Thm. 5.1.1].

1. Consider an independent family $\{\boldsymbol{X}_1, \dots, \boldsymbol{X}_s\} \subset \mathbb{H}_n$ of random psd matrices that satisfy $\lambda_{\max}(\boldsymbol{X}_i) \leq L$ almost surely. Let $\boldsymbol{Y} = \sum_{i=1}^{s} \boldsymbol{X}_s$, and define the mean parameters
$$\mu_{\max} := \lambda_{\max}(\mathbb{E}\,\boldsymbol{Y}) \quad \text{and} \quad \mu_{\min} := \lambda_{\min}(\mathbb{E}\,\boldsymbol{Y}).$$
Prove that
$$\mathbb{P}\{\lambda_{\max}(\boldsymbol{Y} - (1+t)(\mathbb{E}\,\boldsymbol{Y})) \geq 0\} \leq n \cdot \left[\frac{\mathrm{e}^{+t}}{(1+t)^{1+t}}\right]^{\mu_{\min}/L} \quad \text{for } t \geq 0;$$
$$\mathbb{P}\{\lambda_{\max}((1-t)(\mathbb{E}\,\boldsymbol{Y}) - \boldsymbol{Y}) \geq 0\} \leq n \cdot \left[\frac{\mathrm{e}^{-t}}{(1-t)^{1-t}}\right]^{\mu_{\min}/L} \quad \text{for } t \in (0,1).$$
Hint: The proof is analogous with the scalar Chernoff inequalities. You just need to bound the matrix mgf by a linear function.
2. What implications do these maximum eigenvalue inequalities have for the relationship between $\boldsymbol{Y}$ and $\mathbb{E}\,\boldsymbol{Y}$?

3 Let $\boldsymbol{x}$ be a centered random vector with covariance matrix $\boldsymbol{C} = \mathbb{E}[\boldsymbol{x}\boldsymbol{x}^*]$. The sample covariance estimator takes the form $\widehat{\boldsymbol{C}}_s = s^{-1} \sum_{i=1}^{s} \boldsymbol{x}_i \boldsymbol{x}_i^*$, where $\boldsymbol{x}_i \sim \boldsymbol{x}$ iid. Use the matrix Chernoff inequalities to estimate how many samples s are required to obtain the pair of bounds $\widehat{\boldsymbol{C}}_s \preccurlyeq (1+\varepsilon)\boldsymbol{C}$ and $\widehat{\boldsymbol{C}}_s \succcurlyeq (1-\varepsilon)\boldsymbol{C}$. Explain the significance of your bound. **Hint:** Before applying matrix Chernoff, conjugate the random matrix $\hat{\boldsymbol{C}}_s$ by $\boldsymbol{C}^{-1/2}$ so that its expectation is $\mathbf{I}$.

Lecture bibliography

[AW02] R. Ahlswede and A. Winter. "Strong converse for identification via quantum channels". In: *IEEE Trans. Inform. Theory* 48.3 (2002), pages 569–579. DOI: 10.1109/18.985947.

[Bha97] R. Bhatia. *Matrix analysis*. Volume 169. Graduate Texts in Mathematics. Springer-Verlag, New York, 1997, pages xii+347. DOI: 10.1007/978-1-4612-0653-8.

[Lie73] E. H. Lieb. "Convex trace functions and the Wigner-Yanase-Dyson conjecture". In: *Advances in Math.* 11 (1973), pages 267–288. DOI: 10.1016/0001-8708(73)90011-X.

[Tro12] J. A. Tropp. "User-friendly tail bounds for sums of random matrices". In: *Found. Comput. Math.* 12.4 (2012), pages 389–434. DOI: 10.1007/s10208-011-9099-z.

[Tro15] J. A. Tropp. "An Introduction to Matrix Concentration Inequalities". In: *Foundations and Trends® in Machine Learning* 8.1-2 (2015), pages 1–230. ISSN: 1935-8237. DOI: 10.1561/2200000048.

6. Gaussian Embeddings

Date: 23 January 2020 **Scribe:** Ziyun Zhang

In this lecture, we begin our discussion of randomized linear embeddings, which are a core tool in modern theory of algorithms. We introduce the simplest construction, the Gaussian embedding. We show how to summarize its performance using the Gaussian width. As applications, we establish the Johnson–Lindenstrauss lemma, and we develop the concept of a subspace embedding.

Agenda:

1. (Random) embeddings
2. Gaussian width
3. Analysis of Gaussian embeddings
4. Johnson-Lindenstrauss
5. Subspace embeddings

6.1 Random embeddings

The purpose of a random embedding is to reduce the dimension of the data while preserving its geometry. By operating on the low-dimensional representation, we can develop faster algorithms that give approximate solutions to data analysis problems.

Consider a set $\mathsf{E} \subset \mathbb{R}^n$. Let $\varepsilon \in (0, 1)$ be a tolerance, which we call the *distortion*. A (linear) map $\boldsymbol{S} : \mathsf{E} \to \mathbb{R}^d$ is called an ℓ_2 *embedding with distortion* ε if

ℓ_2 embedding with distortion ε

$$(1 - \varepsilon)\|\boldsymbol{x}\| \le \|\boldsymbol{S}\boldsymbol{x}\| \le (1 + \varepsilon)\|\boldsymbol{x}\| \quad \text{for all } \boldsymbol{x} \in \mathsf{E}. \tag{6.1}$$

The unadorned norm $\|\cdot\|$ denotes the ℓ_2-norm of a vector. The linearity of $\boldsymbol{S}$ ensures that $\boldsymbol{S}$ is a metric space embedding of (E, ℓ_2) into ℓ_2. See Figure 6.1 for a picture of a dimension reduction map.

In many cases, the embedding dimension d is much smaller than the ambient dimension n, so the map $\boldsymbol{S}$ performs dimension reduction on E. But how can we construct such a map? Randomness offers a powerful mechanism.

Suppose that $\boldsymbol{S} \in \mathbb{R}^{d \times n}$ is a random matrix such that

$$\mathbb{E}\,\|\boldsymbol{S}\boldsymbol{x}\|^2 = \|\boldsymbol{x}\|^2 \quad \text{for all } \boldsymbol{x} \in \mathsf{E}.$$

In other words, $\boldsymbol{S}$ preserves the energy of a vector on average. A sufficient condition is that the random matrix is isotropic: $\mathbb{E}[\boldsymbol{S}^*\boldsymbol{S}] = \mathbf{I}_n$. To verify that $\boldsymbol{S}$ is an embedding, we need uniform control on the deviations from the average behavior. The question is how large the embedding dimension $d = d(\varepsilon)$ should be to guarantee distortion ε.

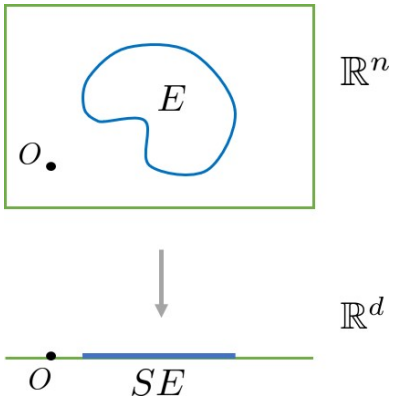


Figure 6.1 Dimension reduction.

6.2 Gaussian embeddings

In this lecture, we build and analyze the simplest example of a random embedding. This theory allows us to understand the potential opportunities for dimension reduction. Today, we will work in the real field ($\mathbb{F} = \mathbb{R}$) so that we can use methods for studying Gaussian processes; this restriction is not necessary in applications.

Definition 6.1 (Gaussian embedding). A *Gaussian embedding* is a real random matrix $\boldsymbol{\Gamma} \in \mathbb{R}^{d\times n}$ whose entries are iid $\sim \textsc{normal}(0, d^{-1})$. The dimensions n and d are respectively called the *ambient / embedding dimension*.

Gaussian embedding

ambient / embedding dimension

The Gaussian embedding has the following simple properties. First,

$$\mathbb{E}_{\boldsymbol{\Gamma}} \|\boldsymbol{\Gamma}\boldsymbol{x}\|^2 = \|\boldsymbol{x}\|^2 \quad \text{for all } \boldsymbol{x} \in \mathbb{R}^n. \tag{6.2}$$

Second, the construction is *oblivious* to the set E; that is, we use no information about the set to construct the embedding. Of course, the embedding dimension d that is sufficient to achieve distortion ε will depend on the geometry of the set E.

oblivious

Exercise 6.2 Check property (6.2).

6.2.1 Restricted singular values

Let us introduce some convenient notation for describing the embedding properties of a linear map.

Definition 6.3 (Set-restricted singular values). Let $\mathsf{E} \subset \mathbb{R}^n$. The *restricted singular values* of a (linear) map $\boldsymbol{S} : \mathbb{R}^n \to \mathbb{R}^d$ are

restricted singular values

$$\sigma_{\min}(\boldsymbol{S}; \mathsf{E}) := \inf_{\boldsymbol{x}\in\mathsf{E}} \frac{\|\boldsymbol{S}\boldsymbol{x}\|}{\|\boldsymbol{x}\|} \quad \text{and} \quad \sigma_{\max}(\boldsymbol{S}; \mathsf{E}) := \sup_{\boldsymbol{x}\in\mathsf{E}} \frac{\|\boldsymbol{S}\boldsymbol{x}\|}{\|\boldsymbol{x}\|}. \tag{6.3}$$

To appreciate the terminology, notice that the restricted singular values for the set $\mathsf{E} = \mathbb{R}^n$ coincide with the ordinary minimal or maximal singular values. In practice, $\sigma_{\min}$ is more important than $\sigma_{\max}$. Indeed, $\sigma_{\min}(\boldsymbol{S}; \mathsf{E}) = 0$ implies that $\boldsymbol{S}$ annihilates points in E. The precise value of $\sigma_{\max}$ does not usually matter so much.

The restricted singular values allow us to state an alternative definition of an embedding. Note that

$$\sigma_{\min}(\boldsymbol{S}; \mathsf{E}) \le \frac{\|\boldsymbol{S}\boldsymbol{x}\|}{\|\boldsymbol{x}\|} \le \sigma_{\max}(\boldsymbol{S}; \mathsf{E}) \quad \text{for all } \boldsymbol{x} \in \mathsf{E}$$

Therefore, if both $1 - \varepsilon \le \sigma_{\min}$ and $\sigma_{\max} \le 1 + \varepsilon$, then the matrix $\boldsymbol{S}$ is an embedding of E of distortion ε.

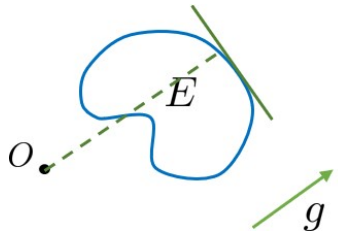


Figure 6.2 The support function $\sup_{\boldsymbol{x} \in \mathsf{E}} \langle \boldsymbol{g}, \boldsymbol{x} \rangle$ in the direction $\boldsymbol{g}$ gives the level of a supporting hyperplane to E with outward normal $\boldsymbol{g}$.

6.3 Gaussian width

The analysis of a Gaussian embedding involves a parameter that summarizes the geometry of the set E. For this purpose, we introduce the following concept.

Definition 6.4 (Gaussian width). Let $\mathsf{E} \subset \mathbb{R}^n$. The *Gaussian width* of E is defined as

$$w(\mathsf{E}) := \mathbb{E} \sup_{\boldsymbol{x} \in \mathsf{E}} \langle \boldsymbol{g}, \boldsymbol{x} \rangle \quad \text{where } \boldsymbol{g} \sim \text{NORMAL}(0, \mathbf{I}_n). \tag{6.4}$$

See Figure 6.2 for an illustration.

In other words, the Gaussian width measures the level of a supporting hyperplane of a set E, averaged over all directions. (Up to scaling, it is equivalent to average over all unit vectors.) The Gaussian width $w(\mathsf{E})$ can be computed analytically for many sets E or approximated numerically.

The Gaussian width is a measure of content of a set E. It has the following properties:

Monotonicity. The width is increasing with respect to set inclusion: $\mathsf{E} \subset \mathsf{F}$ implies $w(\mathsf{E}) \le w(\mathsf{F})$.

Invariance. The width is invariant under Euclidean rigid motions: $w(\boldsymbol{t} + \boldsymbol{Q}\mathsf{E}) = w(\mathsf{E})$ for all $\boldsymbol{t} \in \mathbb{R}^n$ and all orthogonal $\boldsymbol{Q} \in \mathbb{M}_n$.

Range. The width lies in the range $0 \le w(\mathsf{E}) \le \sqrt{n}\,\text{radius}(\mathsf{E})$.

Here is a simple, but important, example of a Gaussian width computation.

Example 6.5 (Subspace). Let $\mathsf{L} \subset \mathbb{R}^n$ be a k-dimensional subspace of $\mathbb{R}^n$. Let us evaluate the width $w(\mathsf{E})$ of the spherical set $\mathsf{E} = \mathsf{L} \cap \mathbb{S}^{n-1}$.

$$w(\mathsf{E}) = \mathbb{E} \sup_{\substack{\boldsymbol{x} \in \mathsf{L} \\ \|\boldsymbol{x}\| = 1}} \langle \boldsymbol{g}, \boldsymbol{x} \rangle = \mathbb{E} \, \|\boldsymbol{P}_{\mathsf{L}} \boldsymbol{g}\| = \mathbb{E}\, \chi_k \in \left[\sqrt{k-1}, \sqrt{k} \right].$$

We have written χ_k for a chi random variable with k degrees of freedom. ∎

This example suggests the following heuristic interpretation.

Remark 6.6 (Heuristic). The *squared* Gaussian width $w^2(\mathsf{E})$ is a measure of the "dimension" of a subset E of the unit sphere.

Remark 6.7 (Valuation). The Gaussian width is a valuation on the class of convex bodies in $\mathbb{R}^n$. This connection has many geometric consequences.

6.4 Analysis of Gaussian embedding

For a subset of the unit sphere, the behavior of a Gaussian embedding is controlled by the Gaussian width.

Theorem 6.8 (Restricted singular values of Gaussian embedding). Let $\mathsf{E} \subset \mathbb{S}^{n-1}$. Draw a Gaussian embedding $\mathbf{\Gamma} \in \mathbb{R}^{d\times n}$. The restricted singular values of $\mathbf{\Gamma}$ satisfy

$$\begin{aligned} \mathbb{P}\left\{\sigma_{\max}(\mathbf{\Gamma};\mathsf{E}) \geq 1 + \frac{w(\mathsf{E})}{\sqrt{d}} + t\right\} &\leq \exp\left(\frac{-dt^2}{2}\right); \\ \mathbb{P}\left\{\sigma_{\min}(\mathbf{\Gamma};\mathsf{E}) \leq 1 - \frac{1 + w(\mathsf{E})}{\sqrt{d}} - t\right\} &\leq \exp\left(\frac{-dt^2}{2}\right). \end{aligned} \tag{6.5}$$

Proof sketch. The two inequalities of Theorem 6.8 are respectively due to Chevet and to Gordon. The book [Ver18] gives an account of these results. The paper [TOH14] contains a partial converse for the lower bound on the restricted minimum singular value. ∎

Theorem 6.8 has an immediate consequence. To embed E with distortion ε via a Gaussian embedding, it suffices that the embedding dimension obeys $d \sim w^2(\mathsf{E})/\varepsilon^2$. Moreover, the constant of proportionality is essentially equal to one.

An important fact is that the tail bound for $\sigma_{\min}$ is *universal* over a large class of random embeddings. In other words, the same result holds for random embeddings with other distributions. In particular, [OT18] establishes that any random embedding with iid subgaussian entries has essentially the same behavior as a Gaussian embedding, provided that E is not too small. (More precisely, the universality result requires that $w(\mathsf{E}) \sim \text{const} \cdot n$.)

6.5 Application: The Johnson–Lindenstrauss lemma

In this section, we use Theorem 6.8 to develop a well-known result on random embedding of a finite point set into a low-dimensional space. This fact follows from more elementary arguments, but it serves as a nice illustration of the general theory.

Let $\mathcal{X} = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_N\} \subset \mathbb{R}^n$ be a finite set of points. We want to embed $\mathcal{X}$ into $\mathbb{R}^d$ while preserving point-wise distances approximately. In other words, we seek a linear map $\boldsymbol{S} \in \mathbb{R}^{d\times n}$ such that

$$(1-\varepsilon)\|\boldsymbol{x}_i - \boldsymbol{x}_j\| \leq \|\boldsymbol{S}\boldsymbol{x}_i - \boldsymbol{S}\boldsymbol{x}_j\| \leq (1+\varepsilon)\|\boldsymbol{x}_i - \boldsymbol{x}_j\| \quad \text{for all } i, j.$$

Note that $\boldsymbol{S}\boldsymbol{x}_i - \boldsymbol{S}\boldsymbol{x}_j = \boldsymbol{S}(\boldsymbol{x}_i - \boldsymbol{x}_j)$ since $\boldsymbol{S}$ is linear. See Figure 6.3 for an illustration. A classic algorithmic application of this type of embedding is to accelerate approximate nearest neighbor computation.

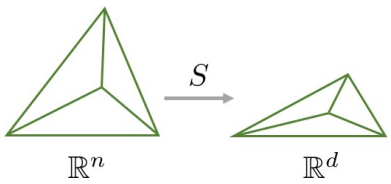


Figure 6.3 Preservation of point-wise distances.

One way to achieve this goal is to make $\boldsymbol{S}$ a Gaussian embedding. Theorem 6.8 yields a short analysis. Define the set of secants

$$\mathsf{E} := \left\{\frac{\boldsymbol{x}_i - \boldsymbol{x}_j}{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|} \;:\; i < j\right\} \subset \mathbb{S}^{n-1}.$$

Then we obtain an embedding guarantee if

$$1 - \varepsilon \leq \sigma_{\min}(\mathbf{\Gamma};\mathsf{E}) \leq \sigma_{\max}(\mathbf{\Gamma};\mathsf{E}) \leq 1 + \varepsilon.$$

Now, from Theorem 6.8 we know that

$$1 - \sigma_{\min}(\boldsymbol{\Gamma}, \mathsf{E}) \approx \frac{1 + w(\mathsf{E})}{\sqrt{d}}, \quad \sigma_{\max}(\boldsymbol{\Gamma}; \mathsf{E}) - 1 \approx \frac{w(\mathsf{E})}{\sqrt{d}}.$$

Therefore we need $d \gtrsim w^2(\mathsf{E})/\varepsilon^2$ to obtain ε distortion. It remains to estimate $w(\mathsf{E})$. We have

$$w(\mathsf{E}) = \mathbb{E} \sup_{\boldsymbol{u}_i \in \mathsf{E}} \langle \boldsymbol{g}, \boldsymbol{u}_i \rangle \le \sqrt{2 \log \#\mathsf{E}} \le 2\sqrt{\log N}.$$

The last inequality is true because the cardinality of E is less than N^2. Thus, if $d \approx 4 \log N / \varepsilon^2$, we get ε distortion. This result is called the *Johnson–Lindenstrauss lemma*.

Exercise 6.9 Prove the inequality $\mathbb{E} \sup_{\boldsymbol{u}_i \in \mathsf{E}} \langle \boldsymbol{g}, \boldsymbol{u}_i \rangle \le \sqrt{2 \log \#\mathsf{E}}$.

Observe that the embedding dimension does not depend on the ambient dimension n at all, and it only depends *logarithmically* on the number N of points. Still, ε^{-2} is very big when ε is small, so it is expensive to achieve small distortion using a randomized linear embedding. This phenomenon is real—it is not an artifact of the analysis. As a consequence, random embeddings are not very useful for achieving accurate low-dimensional representations of a set. For most applications, more sophisticated techniques are needed.

Exercise 6.10 (Johnson–Lindenstrauss; Indyk–Motwani). Develop a direct proof of the embedding result in this section using only Gaussian concentration inequalities.

Remark 6.11 (History). The above problem was initially considered by Johnson and Lindenstrauss [JL84] in the context of embedding a finite metric space into an ℓ_2 space. Later, their lemma was used to design approximation algorithms for graph problems [LLR95]. The paper [HPIM12] connected randomized embeddings with the approximate nearest neighbor problem. Subsequent work by Ravi Kannan and others led to connections with numerical linear algebra.

6.6 Application: Subspace embeddings

For our purposes, the most important application of random embeddings is to preserve the geometry of an entire subspace.

Let $\mathsf{L} \subset \mathbb{R}^n$ be a k-dimensional subspace. A (linear) map $\boldsymbol{S} : \mathbb{R}^n \to \mathbb{R}^d$ is called a *subspace embedding with distortion* ε if

subspace embedding with distortion ε

$$(1 - \varepsilon)\|\boldsymbol{x}\| \le \|\boldsymbol{S}\boldsymbol{x}\| \le (1 + \varepsilon)\|\boldsymbol{x}\| \quad \text{for all } \boldsymbol{x} \in \mathsf{L}.$$

The embedding is *oblivious* if the construction of $\boldsymbol{S}$ does not require any knowledge of L other than its dimension.

Random embeddings lead to easy constructions of oblivious subspace embeddings. In particular, let us analyze the performance of a Gaussian embedding in this context. To apply Theorem 6.8, define $\mathsf{E} = \mathsf{L} \cap \mathbb{S}^{n-1}$. By our earlier computation, we have $w(\mathsf{E}) \le \sqrt{k}$. Thus, $d \approx k/\varepsilon^2$ is the embedding dimension that suffices to achieve distortion ε. In practice, common choices of the embedding dimension are $d = k + p$, where $p \in \{5, 10, k\}$.

Lecture bibliography

[HPIM12] S. Har-Peled, P. Indyk, and R. Motwani. "Approximate nearest neighbor: towards removing the curse of dimensionality". In: *Theory Comput.* 8 (2012), pages 321–350. DOI: 10.4086/toc.2012.v008a014.

[JL84] W. B. Johnson and J. Lindenstrauss. “Extensions of Lipschitz mappings into a Hilbert space”. In: *Conference in modern analysis and probability (New Haven, Conn., 1982)*. Volume 26. Contemp. Math. Amer. Math. Soc., Providence, RI, 1984, pages 189–206. DOI: 10.1090/conm/026/737400.

[LLR95] N. Linial, E. London, and Y. Rabinovich. “The geometry of graphs and some of its algorithmic applications”. In: *Combinatorica* 15.2 (1995), pages 215–245. DOI: 10.1007/BF01200757.

[OT18] S. Oymak and J. A. Tropp. “Universality laws for randomized dimension reduction, with applications”. In: *Inf. Inference* 7.3 (2018), pages 337–446. DOI: 10.1093/imaiai/iax011.

[TOH14] C. Thrampoulidis, S. Oymak, and B. Hassibi. “The Gaussian min-max theorem in the presence of convexity”. In: *arXiv preprint arXiv:1408.4837* (2014).

[Ver18] R. Vershynin. *High-dimensional probability*. Volume 47. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, Cambridge, 2018, pages xiv+284. DOI: 10.1017/9781108231596.

7. Structured Random Embeddings

Date: 28 January 2020 **Scribe:** Fariborz Salehi

In the last lecture, we studied (random) linear embeddings. In our setting, an embedding is a linear operator that maps the points of a subset of a Euclidean space into a lower-dimensional Euclidean space while approximately preserving the geometry of the set. Our discussion focused on Gaussian embeddings, for which the analysis is exceptionally clean.

In this lecture, we introduce two families of structured random embeddings. Unlike Gaussian embeddings, these linear maps can be applied quickly to a vector. This improvement, however, comes at a price. The mathematical analysis of these maps is more complicated and less precise than the analysis in the Gaussian case. In the next lecture, we will discuss some applications of structured subspace embeddings.

Agenda:

1. Review: Gaussian embeddings
2. Structured embeddings
3. Tools of analysis
4. Sparse sign embeddings
5. SRFTs

7.1 Review: Gaussian embeddings

First, let us recall the concept of an embedding.

Definition 7.1 (Embedding). Let $\mathsf{E} \subseteq \mathbb{F}^n$ be a set. A linear map $\boldsymbol{S} : \mathbb{F}^n \to \mathbb{F}^d$ is called an ℓ_2-embedding of E with distortion $\varepsilon \in (0, 1)$ if

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

We call d the *embedding dimension*. *embedding dimension*

We prefer to use embeddings that can be constructed with minimal knowledge of the set E. In this case, we say that the embedding is *oblivious*. *oblivious*

Last time we studied Gaussian embeddings where $\boldsymbol{\Gamma} : \mathbb{F}^n \to \mathbb{F}^d$ has iid $\textsc{normal}(0, d^{-1})$ entries. These embeddings are *isotropic*: *isotropic*

$$\mathbb{E} \|\boldsymbol{\Gamma}\boldsymbol{x}\|^2 = \|\boldsymbol{x}\|^2 \quad \text{for all } \boldsymbol{x} \in \mathbb{F}^n. \tag{7.1}$$

For a subset $\mathsf{E} \subseteq \mathbb{S}^{n-1}(\mathbb{R})$ of the real unit sphere, we saw that the performance of $\boldsymbol{\Gamma}$ is controlled by the Gaussian width $w(\mathsf{E})$ of the set. Roughly speaking, to achieve

distortion ε, it suffices to take the embedding dimension

$$d \gtrapprox \frac{w^2(\mathsf{E})}{\varepsilon^2}.$$

The restriction to unit vectors is inoffensive because the map $\boldsymbol{S}$ is homogeneous.

We also have seen two applications of Gaussian embeddings:

Johnson–Lindenstrauss lemma. A Gaussian embedding with embedding dimension $O(\log N)/\varepsilon^2$ approximately preserves the pairwise distances among N points in $\mathbb{R}^n$.

Subspace embeddings. A Gaussian embedding with embedding dimension $O\left(k/\varepsilon^2\right)$ approximately preserves any fixed k-dimensional subspace $\mathsf{L} \subset \mathbb{R}^n$.

We also discussed the *universality* property. Matrices with iid entries have the same embedding performance as a Gaussian matrix, modulo some technical assumptions. *universality*

7.2 Structured embeddings

Even though Gaussian embeddings work very well, there are several issues that limit their applicability in practice. In particular:

- To construct a Gaussian embedding, we need to generate nd iid samples from the normal distribution. This is costly.
- Gaussian embeddings are expensive to store because they have nd entries.
- Gaussian matrices lack any structure. Applying a Gaussian matrix to a vector requires $O(nd)$ arithmetic operations.

Inspired by the universality property of random embeddings, we study structured random embeddings. We will see that structured random matrices can address the shortcomings of Gaussian embeddings and (in some cases) give even better embedding behavior. This improvement, however, comes at a theoretical price. The analysis of structured embeddings is more complicated and less precise than the analysis of a Gaussian embedding.

We shall focus on two concrete constructions of random embeddings:

1. **Sparse sign matrices.** These embeddings are constructed from a sparse random matrix. Sparsity reduces the cost of construction, storage, and matrix–vector multiplications.
2. **Subsampled randomized Fourier transforms (SRFTs).** These embeddings are based on a Fourier transform that can be executed quickly using the FFT. Since these matrices are partial unitaries (isometries), they can even outperform Gaussian embeddings. Moreover, they use limited storage and are efficient to construct.

Each of these constructions leads to embeddings that are effective for general sets E. We will focus on their behavior as subspace embeddings, which is the core concern for most NLA applications.

7.3 Tools for analysis

Let $\mathsf{L} \in \mathbb{F}^n$ be a k-dimensional subspace. Recall that a linear map $\boldsymbol{S} : \mathbb{F}^n \to \mathbb{F}^d$ is called a *subspace embedding* with distortion ε if *subspace embedding*

$$(1 - \varepsilon)\,\|\boldsymbol{x}\| \le \|\boldsymbol{S}\boldsymbol{x}\| \le (1 + \varepsilon)\,\|\boldsymbol{x}\| \quad \text{for all } \boldsymbol{x} \in \mathsf{L}. \tag{7.2}$$

The Gaussian case provides us with useful guiding intuition. We expect that the embedding dimension should approximately scale like k/ε^2.

To study other kinds of random embeddings, it is valuable to reformulate the embedding condition. Let $\boldsymbol{U} \in \mathbb{F}^{n\times k}$ be a matrix whose columns form an orthonormal basis for L. We can check whether $\boldsymbol{S}$ is an embedding of L by studying the matrix

$$\boldsymbol{Y} = (\boldsymbol{S}\boldsymbol{U})^*(\boldsymbol{S}\boldsymbol{U}).$$

The condition (7.2) is equivalent to the following statement:

$$(1-\varepsilon)^2 \boldsymbol{x}^*(\boldsymbol{U}^*\boldsymbol{U})\boldsymbol{x} \le \boldsymbol{x}^*\boldsymbol{U}^*\boldsymbol{S}^*\boldsymbol{S}\boldsymbol{U}\boldsymbol{x} \le (1+\varepsilon)^2 \boldsymbol{x}^*(\boldsymbol{U}^*\boldsymbol{U})\boldsymbol{x} \quad \text{for all } \boldsymbol{x} \in \mathbb{F}^k.$$

Next, use the fact that $\boldsymbol{U}^*\boldsymbol{U} = \mathbf{I}_k$ and homogeneity to rewrite this condition as

$$(1-\varepsilon)^2 \le \boldsymbol{x}^*\boldsymbol{Y}\boldsymbol{x} \le (1+\varepsilon)^2 \quad \text{for all } \boldsymbol{x} \in \mathbb{F}^k \text{ with } \|\boldsymbol{x}\| = 1.$$

Equivalently,

$$(1-\varepsilon)^2 \le \lambda_{\min}(\boldsymbol{Y}) \le \lambda_{\max}(\boldsymbol{Y}) \le (1+\varepsilon)^2.$$

If we choose $\boldsymbol{S}$ at random, then $\boldsymbol{Y}$ is a random psd matrix. This condition lends itself to analysis using tools from matrix concentration.

Since $\boldsymbol{S}$ is isotropic (7.1), the *expectation* of the matrix $\boldsymbol{Y}$ satisfies the required bounds. Indeed,

$$\mathbb{E}[\boldsymbol{S}^*\boldsymbol{S}] = \mathbf{I}_n \quad \text{implies} \quad \mathbb{E}\,\boldsymbol{Y} = \mathbf{I}_k.$$

We can use the matrix Chernoff inequality to determine the likelihood that the random matrix $\boldsymbol{Y}$ deviates from its expectation.

Theorem 7.2 (Matrix Chernoff Inequality). Let $\boldsymbol{X}_1, \boldsymbol{X}_2, \dots, \boldsymbol{X}_n \in \mathbb{H}_k$ be statistically independent psd matrices that obey $\|\boldsymbol{X}_j\| \le B$. Define $\boldsymbol{Y} = \sum_{j=1}^n \boldsymbol{X}_j$, and set $\mu_{\min} = \lambda_{\min}(\mathbb{E}\,\boldsymbol{Y})$ and $\mu_{\max} = \lambda_{\max}(\mathbb{E}\,\boldsymbol{Y})$. Then,

$$\mathbb{P}\{\lambda_{\max}(\boldsymbol{Y}) \ge (1+t)\mu_{\max}\} \le k\left(\frac{\mathrm{e}^t}{(1+t)^{1+t}}\right)^{\mu_{\max}/B} \quad \text{for } t > 0;$$

$$\mathbb{P}\{\lambda_{\min}(\boldsymbol{Y}) \le (1-t)\mu_{\min}\} \le k\left(\frac{\mathrm{e}^{-t}}{(1-t)^{1-t}}\right)^{\mu_{\min}/B} \quad \text{for } t \in (0,1).$$

This result is an extremely useful tool for controlling the extremal eigenvalues of a random psd matrix. In case $k = 1$, it reduces to the scalar Chernoff inequality. In contrast to the matrix Bernstein inequality, no matrix variance bounds are required, which makes the Chernoff inequality easier to use.

Problem 7.3 (Matrix Chernoff). Prove the matrix Chernoff inequality by generalizing the proof of the scalar Chernoff inequality and invoking the master matrix concentration inequality.

7.4 Sparse sign matrices

Our first construction of an efficient random embedding is called a *sparse sign matrix*. *sparse sign matrix*
The idea is to construct the matrix $\boldsymbol{S} \in \mathbb{R}^{d\times n}$ randomly so that most of its entries are zero. The simplest approach is based on (appropriately rescaled) matrices with iid random ternary entries:

$$\boldsymbol{S} = \alpha \begin{bmatrix} s_{11} & \cdots & s_{1n} \\ \vdots & & \vdots \\ s_{d1} & \cdots & s_{dn} \end{bmatrix} \in \mathbb{R}^{d\times n} \quad \text{where} \quad s_{ij} \sim \begin{cases} +1 & \text{w.p. } p/2; \\ -1 & \text{w.p. } p/2; \\ 0 & \text{w.p. } 1-p. \end{cases}$$

The probability $p \in (0, 1)$ determines the proportion of nonzero entries. The parameter α is an additional scaling factor that we adjust to ensure isotropy. Setting

$$\alpha^2 = \frac{1}{dp} \quad \text{implies} \quad \mathbb{E}\, \boldsymbol{S}^* \boldsymbol{S} = \mathbf{I}_n.$$

Exercise 7.4 Verify that this choice of α leads to an isotropic random embedding.

Each column contains dp nonzero entries on average. Therefore, the total storage cost is roughly $O(ndp)$. When p is small, this is much better than the $O(nd)$ cost of storing a dense matrix. Applying $\boldsymbol{S}$ to a vector requires roughly $O(ndp)$ arithmetic operations. Note that efficient implementation of a sparse random embedding requires good sparse arithmetic libraries.

Having established isotropy, the next step is to apply matrix concentration. We intend to do this via the matrix Chernoff inequality. To that end, we decompose the matrix $\boldsymbol{Y}$ into a sum of psd rank-one matrices:

$$\boldsymbol{Y} = \sum\nolimits_{j=1}^{d} \frac{1}{dp} \boldsymbol{U}^* \boldsymbol{s}_j \boldsymbol{s}_j^* \boldsymbol{U} = \sum\nolimits_{j=1}^{d} \boldsymbol{X}_j \quad \text{where } \boldsymbol{s}_j \text{ is the } j\text{th row of } \boldsymbol{S}.$$

By construction, the $\boldsymbol{X}_j$ are independent and psd. Moreover, isotropy implies

$$\mathbb{E}\, \boldsymbol{Y} = \mathbf{I}_k \quad \text{and therefore} \quad \mu_{\min} = \mu_{\max} = 1.$$

To invoke matrix concentration, we also require a suitable *a priori* bound on the operator norm of $\boldsymbol{X}_j$. This step is actually a bit tricky. Indeed,

$$\mathbb{E}\, \|\boldsymbol{X}_j\| = \frac{k}{d} \quad \text{and} \qquad \|\boldsymbol{X}_j\| \le \frac{n}{dp}.$$

The worst-case bound is much bigger than the typical value, and it is too large to deduce any nontrivial result at all unless the embedding dimension $d \gg n$. At the same time, it is extremely unlikely that summands with extremely large operator norms ever occur.

We can use truncation to control the contribution for large summands. Define the event

$$\mathsf{A}_j := \left\{ \|\boldsymbol{X}_j\| \le \frac{k}{d} + \frac{\text{const} \cdot \sqrt{k} \log(n/p)}{dp} \right\}.$$

We pass to the truncated matrix

$$\boldsymbol{Y}_{\text{trunc}} = \sum\nolimits_{j=1}^{d} \mathbb{1}_{\mathsf{A}_j} \boldsymbol{X}_j,$$

where $\mathbb{1}_{\mathsf{A}_j}$ denotes the indicator function associated with the event A_j. This truncation enforces an operator norm bound, but it also introduces a bias: $\mathbb{E}\, \boldsymbol{Y}_{\text{trunc}} \neq \mathbb{E}\, \boldsymbol{Y}$. Happily, one can show that the bias is small, because each event A_j occurs with overwhelming probability. By applying the matrix Chernoff inequality to $\boldsymbol{Y}_{\text{trunc}}$, together with some additional arguments, we arrive at the following result.

Theorem 7.5 (Sparse sign matrix—informal). For constants $C_1, C_2 > 0$, suppose that

$$d \gtrapprox C_1(k + \log n) \cdot \log k \quad \text{and} \quad p \gtrapprox C_2 \cdot \frac{\log k}{d}.$$

With high probability, for any fixed k-dimensional subspace, the sparse sign matrix

> $\boldsymbol{S}$ is a subspace embedding with constant distortion ($\varepsilon = \frac{1}{2}$).

In practice, sparse sign embeddings have similar performance to Gaussian embeddings, but they enjoy advantages in storage and arithmetic costs.

Problem 7.6 (Sparse sign matrices). Prove Theorem 7.5. **Hint:** Use the Hanson–Wright inequality to control the probability of the events A_j.

Problem 7.7 (Fixed sparsity sign matrices). A more effective construction places exactly ζ nonzero random signs in random locations in each column of $\boldsymbol{S}$. Show that the resulting random matrix is a more effective embedding than a sparse sign matrix with independent entries. **Hint:** Use decoupling and matrix Bernstein.

Remark 7.8 (Analysis). We can obtain a cleaner analysis of sparse embeddings using more sophisticated matrix concentration inequalities, such as the matrix Rosenthal inequalities [Tro16].

7.5 Subsampled randomized Fourier transforms (SRFTs)

Our second construction of an efficient random embedding is called a *subsampled randomized Fourier transform (SRFT)*. The basic idea is to apply a fast Fourier transform to mix vector coordinates quickly and efficiently. The actual embedding is composed of three simpler linear transforms:

subsampled randomized Fourier transform (SRFT).

$$\boldsymbol{S} = \sqrt{\frac{n}{d}}\boldsymbol{R}\boldsymbol{F}\boldsymbol{E} \quad \text{where} \quad \begin{cases} \boldsymbol{R} \in \mathbb{C}^{d\times n} & \text{is a random restriction,} \\ \boldsymbol{F} \in \mathbb{C}^{n\times n} & \text{is a discrete Fourier transform,} \\ \boldsymbol{E} \in \mathbb{C}^{n\times n} & \text{is random diagonal sign matrix.} \end{cases}$$

The restriction $\boldsymbol{R}$ extracts d entries at random, and the sign matrix $\boldsymbol{E} = \operatorname{diag}(\varepsilon_1, \dots, \varepsilon_n)$ with $\varepsilon_i \sim \{\pm 1\}$ iid. The SRFT is a partial unitary matrix, which allows the SRFT to outperform Gaussian embeddings (which are not partial isometries) in some cases.

SRFTs are very cheap to store and apply. One simply needs to store $n + d$ numbers, namely the diagonal entries of the matrix $\boldsymbol{E}$ and the d coordinates chosen by the restriction. Applying the SRFT to a vector in $\mathbb{C}^n$ can be achieved via a subsampled Fast Fourier Transform (FFT) and requires only $O(n \log d)$ arithmetic operations. Note, however, that a good FFT library is required to achieve this performance.

The most important feature of SRFTs is that they rapidly mix and flatten coordinates. In expectation,

$$\mathbb{E}\,|(\boldsymbol{F}\boldsymbol{E}\boldsymbol{x})_i|^2 = \mathbb{E}\left|\sum\nolimits_j f_{ij}\varepsilon_i x_j\right|^2 = \frac{1}{n}\,\|\boldsymbol{x}\|^2 \ . \tag{7.3}$$

Once the coordinates are flat, we can simply sample d coordinates to collect a (d/n)-fraction of the total energy. Since the coordinates are small, the variance of the sampling step is low.

7.5.1 Analysis of SRFTs

We can analyze the performance of an SRFT using the matrix Chernoff bound. As the first step, we decompose the matrix $\boldsymbol{Y}$ as a sum of independent psd matrices:

$$\boldsymbol{Y} = \boldsymbol{U}^*\boldsymbol{S}^*\boldsymbol{S}\boldsymbol{U} = (\boldsymbol{F}\boldsymbol{E}\boldsymbol{U})^*(\boldsymbol{R}^*\boldsymbol{R})(\boldsymbol{F}\boldsymbol{E}\boldsymbol{U}) = \frac{n}{d}\sum\nolimits_{j=1}^{n} \delta_j \boldsymbol{w}_j\boldsymbol{w}_j^* = \sum\nolimits_{j=1}^{n} \boldsymbol{X}_j.$$

The random selector $\delta_j \in \{0, 1\}$ coincides with the jth diagonal entry of the diagonal matrix $\boldsymbol{R}^*\boldsymbol{R}$. The vector $\boldsymbol{w}_j^*$ is the jth row of the matrix $\boldsymbol{F}\boldsymbol{E}\boldsymbol{U}$. Next, assume that

$\boldsymbol{R}$ is prepared by selecting d rows of the identity matrix at random. Then each $\delta_j \sim \textsc{Bernoulli}(d/n)$ iid.

The remaining argument is based on two steps. First, we prove that the SRFT flattens the rows of $\boldsymbol{U}$ with high probability:

$$\mathbb{P}\left\{\max_j \|\mathbf{e}_j^*(\boldsymbol{FEU})\|^2 \geq \frac{d}{n} + \frac{\text{const} \cdot \log n}{n}\right\} \quad \text{is small.}$$

This is achieved by applying Hoeffding's inequality and a union bound.

Conditional on the first step succeeding, we continue to the second part of the analysis. We can exploit the randomness in $\boldsymbol{R}^*\boldsymbol{R}$ to apply a matrix Chernoff argument. Isotropy implies $\mathbb{E}\boldsymbol{Y} = \mathbf{I}_k$, which ensures that $\mu_{\min} = \mu_{\max} = 1$. Since the rows of $\boldsymbol{FEU}$ are flat, we have the operator norm bound

$$\|\boldsymbol{X}_j\| \leq \frac{n}{d}\|\boldsymbol{w}_j\|^2 = \frac{n}{d}\|\boldsymbol{e}_j^*(\boldsymbol{FEU})\|^2 \leq \frac{k}{d} + \frac{\text{const} \cdot \log n}{d}.$$

This analysis leads to the following result.

Theorem 7.9 (SRFTs—informal). For a constant $C > 0$, suppose that

$$d \gtrapprox C(k + \log n) \cdot \log k.$$

With high probability, for a fixed k-dimensional subspace, the SRFT matrix $\boldsymbol{S}$ is a subspace embedding with constant distortion ($\varepsilon = 1/2$).

In practice, the extra factor of $\log k$ is usually unnecessary. There are also variants of the SRFT that use random permutations or apply the randomized transform twice to mix the coordinates better. These approaches are more reliable, and only slightly more expensive.

Problem 7.10 (SRFTs). Prove Theorem 7.9.

Lecture bibliography

[Tro16] J. A. Tropp. "The expected norm of a sum of independent random matrices: an elementary approach". In: *High dimensional probability VII*. Volume 71. Progr. Probab. Springer, [Cham], 2016, pages 173–202. DOI: 10.1007/978-3-319-40519-3_8.

8. How to Use Random Embeddings

Date: 30 January 2020 **Scribe:** Riley Murray

This lecture explores how subspace embeddings can improve on classical methods for some problems in numerical linear algebra. Overdetermined least-squares serves as the main case study; Section 8.1 gives the problem statement and important background material. Section 8.2 covers three algorithms for least-squares based on subspace embeddings, which we compare in Section 8.3. Section 8.4 covers implementation details and further reading.

Agenda:

1. Least-squares: problem & background
2. Least-squares: three randomized algorithms
3. Runtime discussion
4. Beyond least-squares
5. Further reading
6. Problems

8.1 A case study: Overdetermined least-squares

Let $\boldsymbol{A}$ be a matrix in $\mathbb{F}^{m\times n}$ with linearly independent columns, and let $\boldsymbol{b}$ be a vector in $\mathbb{F}^m$. The ordinary least squares problem with respect to these parameters is to compute

$$\boldsymbol{x}_\star = \operatorname{argmin}\left\{\|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}\|^2 : \boldsymbol{x}\in\mathbb{F}^n\right\}. \tag{8.1}$$

The assumption that $\boldsymbol{A}$ has linearly independent columns ensures that (8.1) has a unique minimizer. In particular, $\boldsymbol{x}_\star$ solves the *normal equations* $\boldsymbol{A}^*\boldsymbol{A}\boldsymbol{x}_\star = \boldsymbol{A}^*\boldsymbol{b}$. In the regime $n \ll m$, the problem is said to be *overdetermined*, and we have $\|\boldsymbol{A}\boldsymbol{x}_\star - \boldsymbol{b}\| > 0$ for almost all $\boldsymbol{b}$ in $\mathbb{F}^m$.

normal equations
overdetermined

The methods presented in this lecture focus on highly overdetermined problems, with $n \ll m/\log n$. We also assume $\boldsymbol{A}$ is dense. This assumption is largely so we can cleanly compare the asymptotic time complexities of various algorithms. Once the reader understands the concepts presented here, it would be worthwhile to revisit this problem assuming $\boldsymbol{A}$ is sparse, or assuming $\boldsymbol{A}$ can only be accessed with a matrix–vector multiplication primitive.

8.1.1 A classical direct method for least-squares

In linear algebra, a *direct method* is an algorithm that produces an exact solution to a given problem using a finite number of arithmetic operations. Modulo issues

with finite-precision arithmetic, direct methods in numerical linear algebra generally produce extremely accurate solutions. Direct methods exhibit almost no variation in runtime, unless they attempt to account for sparsity.

Before we turn to the randomized algorithms for solving (8.1), we first review the basics of classical algorithms for the same.

A common direct method for solving problem (8.1) is to employ a QR-factorization of the matrix $\boldsymbol{A}$. By applying (double) Gram–Schmidt to the columns of $\boldsymbol{A}$, we can factorize $\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{R}$ with orthonormal $\boldsymbol{Q} \in \mathbb{F}^{m\times n}$ and nonsingular upper-triangular $\boldsymbol{R} \in \mathbb{F}^{n\times n}$. The columns of $\boldsymbol{Q}$ compose an orthonormal basis for the subspace range($\boldsymbol{A}$). The factorization process takes $O(mn^2)$ time. From there, we may compute the solution $\boldsymbol{x}^\star = \boldsymbol{R}^{-1}\boldsymbol{Q}^*\boldsymbol{b}$ in $O(mn)$ time. The total time complexity is dominated by the factorization step, and comes in at $O(mn^2)$ arithmetic operations.

Remark 8.2 (Pivoting). In practice, it is often necessary to use column-pivoted QR (CPQR). For conceptual and notational simplicity, we use a plain QR factorization instead.

Warning 8.1 Our description of these classical algorithms is a little simplified. Asymptotic runtimes are unchanged from practical implementations, but some important improvements can be made. See [She94] for details on the conjugate gradient method. ∎

8.1.2 A classical iterative method for least-squares

An *iterative method* is an algorithm that generates a sequence of points $(\boldsymbol{z}_k)_{k\in\mathbb{N}}$, where $\lim_{k\to\infty} \boldsymbol{z}_k = \boldsymbol{z}_\star$ for a value $\boldsymbol{z}_\star$ with some desired properties. Iterative methods are essential for eigenvalue problems, but also play an important role in solving linear equations.

iterative method

Some of the methods in this lecture involve the *conjugate gradient (CG)* algorithm. The conjugate gradient method is usually introduced as a technique for solving positive-definite linear systems $\boldsymbol{G}\boldsymbol{z} = \boldsymbol{h}$ with $\boldsymbol{G}$ in $\mathbb{H}^{++}_n$. The advantage of CG over matrix factorization methods, is that CG only requires access to $\boldsymbol{G}$ by way of the matrix–vector multiplication primitive: $\boldsymbol{z} \mapsto \boldsymbol{G}\boldsymbol{z}$.

conjugate gradient (CG)

Algorithm 8.1 provides a basic implementation of the conjugate gradient method. The time required by an iteration of Algorithm (8.1) is proportional to the time required to evaluate $\boldsymbol{G}\boldsymbol{p}$ for some $\boldsymbol{p} \in \mathbb{F}^n$. That is, $O(n^2)$ time per iteration in the dense case.

The intermediate calculations of Algorithm 8.1 may seem mysterious, but the overall effect is this: CG is a descent method for minimizing the convex quadratic function $\boldsymbol{z} \mapsto \frac{1}{2}\boldsymbol{z}^*\boldsymbol{G}\boldsymbol{z} - \boldsymbol{h}^*\boldsymbol{z}$, where the kth search direction $\boldsymbol{p}_k := \boldsymbol{z}_k - \boldsymbol{z}_{k-1}$ is required to be $\boldsymbol{G}$-orthogonal (or "conjugate") to all preceding search directions.

The following standard theorem [She94] addresses CG's convergence, which tells us how many iterations are required to achieve a certain error.

Theorem 8.3 (Convergence of CG). Let $\boldsymbol{e}_k$ denote the error of Algorithm 8.1 after k iterations; i.e., $\boldsymbol{e}_k = \boldsymbol{z}_k - \boldsymbol{z}_\star$ for $\boldsymbol{z}_\star = \boldsymbol{G}^{-1}\boldsymbol{h}$. We have the bound

$$\|\boldsymbol{G}^{1/2}\boldsymbol{e}_k\| \le 2\left(\frac{\sqrt{\kappa(\boldsymbol{G})}-1}{\sqrt{\kappa(\boldsymbol{G})}+1}\right)^k \|\boldsymbol{G}^{-1/2}\boldsymbol{h}\|$$

where $\kappa(\boldsymbol{G}) = \sigma_{\max}(\boldsymbol{G})/\sigma_{\min}(\boldsymbol{G})$ is the condition number of $\boldsymbol{G}$.

We can solve the least-squares problem (8.1) by applying CG to the normal equations $\boldsymbol{A}^*\boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^*\boldsymbol{b}$. In this case, we identify $\boldsymbol{G} = \boldsymbol{A}^*\boldsymbol{A}$ and $\boldsymbol{h} = \boldsymbol{A}^*\boldsymbol{b}$. Since CG only requires access to $\boldsymbol{G} = \boldsymbol{A}^*\boldsymbol{A}$ via matrix–vector products, we do not need to form $\boldsymbol{G}$ explicitly. This is good news, because while the complexity of forming $\boldsymbol{G}$ is $O(mn^2)$, the complexity of applying $\boldsymbol{G}$ to a vector is only $O(mn)$. We thus see that if the condition number of $\boldsymbol{G}$ is small, a CG method for solving problem (8.1) can have overall complexity far lower

Warning 8.4 Using CG directly on the normal equations squares the condition number of the problem. To avoid this issue, we use a variant of CG, called CGLS, to solve least-squares problems. ∎

Algorithm 8.1 BasicCG.

Input: Matrix $\boldsymbol{G} \in \mathbb{H}_n^{++}$, nonzero $\boldsymbol{h} \in \mathbb{F}^n$, tolerance $\varepsilon \geq 0$, iteration limit $k_{\max}$.
Output: Approximate solution $\boldsymbol{z} \in \mathbb{F}^n$ of the system $\boldsymbol{G}\boldsymbol{z} = \boldsymbol{h}$.

```
function BasicCG(G, h, ε, k_max)
    z = 0                                  ▷ The current approximate solution.
    r = h                                  ▷ Residual: h − Gz.
    p = h                                  ▷ The current "conjugate direction."
    k = 0
    while k < k_max do
        num = r*r
        den = p*Gp                         ▷ Often, explicitly store the product Gp.
        α = num/den
        z = z + αp
        r = r − αGp                        ▷ Reuse Gp, if it was stored explicitly.
        if ||r|| ≤ ε then
            Break
        β = r*r/num
        p = r + βp
        k = k + 1
    return z
```

than the $O(mn^2)$ method provided by a QR factorization.

But therein lies the problem. If the condition number of $\boldsymbol{G}$ is large, CG may be prohibitively slow.

One remedy it to employ the *preconditioned conjugate gradient (PCG)* method. The idea behind PCG is to form a "preconditioner" $\boldsymbol{C}$ for which we can efficiently evaluate both $\boldsymbol{C}$ and $\boldsymbol{C}^{-1}$, and the condition number of the matrix $\hat{\boldsymbol{G}} := \boldsymbol{C}^{-1}\boldsymbol{G}(\boldsymbol{C}^{-1})^*$ is $O(1)$. We then solve $\hat{\boldsymbol{G}}\boldsymbol{y} = \boldsymbol{C}^{-1}\boldsymbol{h}$ with just a few iterations of CG. Last, we report $\boldsymbol{z} = (\boldsymbol{C}^*)^{-1}\boldsymbol{y}$.

preconditioned conjugate gradient (PCG)

Remark 8.5 (PCG variants). There is a variant of PCG which only requires access to $\boldsymbol{G}$ and $(\boldsymbol{C}^{-1})^*(\boldsymbol{C}^{-1})$, and so loosens the requirement on the preconditioner. We do not need that refinement for this lecture.

8.2 Three randomized algorithms for least-squares

This section develops and analyzes three subspace embedding approaches to problem (8.1). Each of these methods requires the user to specify the family of embeddings, such as Gaussian, sparse sign matrices, or SRFTs.

The embedding should always be chosen to take advantage of problem structure. Since we assume $\boldsymbol{A}$ is dense, both sparse sign matrices and SRFTs are reasonable candidates here. This lecture proceeds with SRFT embeddings, and henceforth reserves $\boldsymbol{S} \in \mathbb{F}^{d \times m}$ as an SRFT embedding matrix, where d is determined from context. We use the fact that $\boldsymbol{S}$ can be stored in $O(m)$ space, and that $\boldsymbol{S}$ can be applied to a vector in $\mathbb{F}^m$ in $O(m \log d)$ time. We also use the following theorem.

Aside: A naive SRFT implementation would require $O(m \log m)$ to evaluate $\boldsymbol{S}$ via FFT. The stronger $O(m \log d)$ bound is possible when using a subsampled FFT.

Theorem 8.6 (SRFTs). For a large positive integer m, consider a subspace L of dimension $n = \Omega(\log m)$ within $\mathbb{R}^m$. Also consider distortion parameters $\varepsilon \in (0, 1)$.

If m tends to infinity while d is chosen according to

$$d \in \Omega(n \log n/\varepsilon^2),$$

then the probability that a d-by-m SRFT matrix ε-embeds L tends to 1.

Remark 8.7 (Alternative formulation). The requirement that $n \in \Omega(\log m)$ in Theorem 8.6 can equivalently be written as $m \in O(2^n)$. This requirement on n is only used to keep downstream expressions simple. The theorem holds more generally with $d \in \Omega([n + \log m] \log n/\varepsilon^2)$.

8.2.1 One-shot sketch & solve

Given a matrix $\boldsymbol{A} \in \mathbb{F}^{m\times n}$, a vector $\boldsymbol{b} \in \mathbb{F}^m$, and an embedding dimension $d \le m$ the *Sketch-and-Solve* paradigm first constructs a subspace embedding $\boldsymbol{S} \in \mathbb{F}^{d\times m}$. Then we find *Sketch-and-Solve*

$$\hat{\boldsymbol{x}} \in \operatorname{argmin} \left\{ \|\boldsymbol{SAx} - \boldsymbol{Sb}\|^2 \,:\, \boldsymbol{x} \in \mathbb{F}^n \right\}, \tag{8.2}$$

and return $\hat{\boldsymbol{x}}$ as an approximate solution to (8.1).

Sketch-and-Solve is very simple, but there is intuition to support its use. Consider how if $\boldsymbol{S}$ is an ε-embedding for the subspace spanned by vectors $\{\boldsymbol{Ax} - \boldsymbol{b} : \boldsymbol{x} \in \mathbb{F}^n\}$, then $\boldsymbol{r}_\star = \boldsymbol{Ax}_\star - \boldsymbol{b}$ will satisfy $\|\boldsymbol{Sr}_\star\| \in (1 \pm \varepsilon)\|\boldsymbol{r}_\star\|$. So, since the optimal solution $\boldsymbol{x}_\star$ for (8.1) has similar cost when considered in problem (8.2), perhaps $\boldsymbol{x}_\star$ is close to to $\hat{\boldsymbol{x}}$?

While this intuition is not quite correct, we *can* say something interesting. We indeed suppose $\boldsymbol{S}$ is an ε-embedding for range($[\boldsymbol{A}|\boldsymbol{b}]$). From there, we set $\hat{\boldsymbol{r}} = \boldsymbol{A}\hat{\boldsymbol{x}} - \boldsymbol{b}$, and deduce the chain of inequalities

$$(1-\varepsilon)\|\hat{\boldsymbol{r}}\|^2 \le \|\boldsymbol{S}\hat{\boldsymbol{r}}\|^2 \le \|\boldsymbol{Sr}_\star\|^2 \le (1+\varepsilon)\|\boldsymbol{r}_\star\|^2.$$

The first and third inequalities above used the ε-embedding property of $\boldsymbol{S}$, and the middle inequality used optimality of $\hat{\boldsymbol{x}}$ for problem (8.2). The preceding display can be summarized as

$$\|\boldsymbol{A}\hat{\boldsymbol{x}} - \boldsymbol{b}\|^2 \le \left(\frac{1+\varepsilon}{1-\varepsilon}\right) \|\boldsymbol{Ax}_\star - \boldsymbol{b}\|^2. \tag{8.3}$$

One might simplify (8.3) further by using $(1+\varepsilon)/(1-\varepsilon) < (1+3\varepsilon)$ for $\varepsilon \in (0, 4/10)$, or $(1+\varepsilon)/(1-\varepsilon) \approx 1 - 2\varepsilon$ for $\varepsilon \ll 1$.

What is the computational cost of the Sketch-and-Solve paradigm for overdetermined least-squares? The cost of forming the sketched problem data $\boldsymbol{SA}$ and $\boldsymbol{Sb}$ takes $O(mn \log d)$ time. Once this is done, we can solve (8.2) with a direct method in $O(dn^2)$ time. If we weren't worried about accuracy guarantees, the resulting runtime of $O(mn \log d + n^2 d)$ would seem appealing.

However, if we want satisfy inequality (8.3) with high probability, we ought to set d in accordance to Theorem 8.6. Supposing we choose d in this way the resulting runtime of Sketch-and-Solve becomes $O(mn \log d + n^3 \log(n)/\varepsilon^2)$. The dependence $1/\varepsilon^2$ is *terrible*, and it precludes the use of Sketch-and-Solve when high accuracy is needed.

8.2.2 Iterative sketching

Here we describe the "Iterative-Sketch" algorithm. A key ingredient to this algorithm's success is sketching in a way that lends itself to successive refinement. To begin, note

that (8.1) is equivalent to minimizing

$$f(\boldsymbol{x}) := \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x}\|^2 - (\boldsymbol{A}^*\boldsymbol{b})^*\boldsymbol{x}.$$

Writing least-squares as minimizing f can make it easier to see the forces at play: on one hand, we want $\boldsymbol{x}$ to point as far along the ray $\boldsymbol{A}^*\boldsymbol{b}$ as possible, on the other hand, the size of $\boldsymbol{x}$ is penalized by a quadratic term $\|\boldsymbol{A}\boldsymbol{x}\|^2/2$.

Idea: If we want an approximate minimizer of f over $\mathbb{F}^n$, we should only apply sketching to the quadratic penalty term.

We can put this idea into action by sampling a sketching matrix $\boldsymbol{S}$, and setting $\hat{\boldsymbol{x}}$ as the solution to

$$\text{minimize} \quad \frac{1}{2}\|\boldsymbol{S}\boldsymbol{A}\boldsymbol{z}\|^2 - (\boldsymbol{A}^*\boldsymbol{r})^*\boldsymbol{z} \quad \text{over} \quad \boldsymbol{z} \in \mathbb{F}^n \tag{8.4}$$

for $\boldsymbol{r} = \boldsymbol{b}$. In principle, we could return $\hat{\boldsymbol{x}}$ as an approximate minimizer of f, but we can also take this farther. Once $\hat{\boldsymbol{x}}$ is in hand, we

1. Replace $\boldsymbol{r}$ by $\boldsymbol{r} - \boldsymbol{A}\hat{\boldsymbol{x}}$.
2. Find the vector $\boldsymbol{u} \in \mathbb{F}^n$ that solves (8.4).
3. Accumulate $\hat{\boldsymbol{x}} = \hat{\boldsymbol{x}} + \boldsymbol{u}$.

The Iterative-Sketch paradigm repeats the above process for a prescribed number of iterations.

All of the work in the Iterative Sketch approach is contained in Part (2). In order to perform that task reliably, we use a direct method for solving the linear system $(\boldsymbol{S}\boldsymbol{A})^*(\boldsymbol{S}\boldsymbol{A})\boldsymbol{u} = \boldsymbol{A}^*\boldsymbol{r}$. The *first iteration* begins by forming the matrix $\boldsymbol{G} = (\boldsymbol{S}\boldsymbol{A})^*(\boldsymbol{S}\boldsymbol{A})$ in $O(mn\log d + dn^2)$ time, and then performing a Cholesky factorization of $\boldsymbol{G}$ at an additional cost of $O(n^3)$ arithmetic. Since $n < d$, the overall complexity of forming $\boldsymbol{G}$ and computing the Cholesky factorization is $O(mn\log d + dn^2)$. In *every iteration*, we must solve (8.3) with a new vector $\boldsymbol{r}$. This requires forming $\boldsymbol{h} = \boldsymbol{A}^*\boldsymbol{r}$ in $O(nm)$ time and then solving $\boldsymbol{G}\boldsymbol{u} = \boldsymbol{h}$ in time $O(n^2) \ll O(nm)$.

Note that the complexity of the first iteration of the Iterative Sketch approach is the same as the entirety of the Sketch-and-Solve paradigm, *provided that d is the same*. The difference between these two algorithms is that Sketch-and-Solve requires $d \in \Omega(n\log n/\varepsilon^2)$ to produce an ε-accurate solution, but the inner Iterative-Sketch steps can get away with $d \in \Omega(n\log n)$. In the iterative approach, the dependence on ε only comes into play in the number of *iterations*.

Let us see why this is the case. Begin by setting $\boldsymbol{P} \in \mathbb{F}^{m\times m}$ to the orthogonal projector onto the subspace range$(\boldsymbol{A})$. Verify the claim in the following exercise.

Exercise 8.8 (Iterative Sketching). Suppose $\boldsymbol{S}$ is subspace embedding for range$(\boldsymbol{A})$ with distortion $\delta \in (0, 0.25]$, and set $\boldsymbol{r} = \boldsymbol{b}$. Show that the solution $\hat{\boldsymbol{x}}$ to (8.4) satisfies

$$\|\boldsymbol{A}\hat{\boldsymbol{x}} - \boldsymbol{P}\boldsymbol{b}\|^2 \le 3\delta\|\boldsymbol{P}\boldsymbol{b}\|^2. \tag{8.5}$$

Furthermore,

$$\|\boldsymbol{A}\hat{\boldsymbol{x}} - \boldsymbol{b}\|^2 \le \|\boldsymbol{A}\boldsymbol{x}_\star - \boldsymbol{b}\|^2 + 3\delta\|\boldsymbol{P}\boldsymbol{b}\|^2. \tag{8.6}$$

The analysis of the Iterative Sketch approach becomes simple once (8.6) is in hand. Suppose $\boldsymbol{S}$ is a δ-embedding for range$(\boldsymbol{A})$, and that we run Iterative Sketch for k

steps. Letting $\boldsymbol{x}_k$ denote the accumulated approximate solution to (8.1), an inductive argument establishes linear convergence (!)

$$\|\boldsymbol{A}\boldsymbol{x}_k - \boldsymbol{b}\|^2 \le \|\boldsymbol{A}\boldsymbol{x}_\star - \boldsymbol{b}\|^2 + (3\delta)^k \|\boldsymbol{P}\boldsymbol{b}\|^2. \tag{8.7}$$

Equation (8.7) is the key to the success of the Iterative Sketch paradigm. We can simply take δ as a modest constant (e.g., $\delta = 1/4$), and choose the embedding dimension $d \in \Theta(n \log n)$. In terms of the *dynamic range* $D := \|\boldsymbol{P}\boldsymbol{b}\| / \|(\boldsymbol{I} - \boldsymbol{P})\boldsymbol{b}\| > 0$ of the least-squares problems, we need only run for $O(\log(D/\varepsilon))$ before finding an ε-accurate solution $\hat{\boldsymbol{x}}$.

dynamic range

Theorem 8.9 (Iterative Sketching). Let $\boldsymbol{A} \in \mathbb{F}^{m \times n}$ and $\boldsymbol{b} \in \mathbb{F}^m$ specify an instance of problem (8.1). Set $D = \|\boldsymbol{P}\boldsymbol{b}\| / \|(\boldsymbol{I} - \boldsymbol{P})\boldsymbol{b}\|$ and assume $D > 0$. With high probability, the SRFT-based Iterative Sketch procedure produces a solution $\hat{\boldsymbol{x}}$ satisfying

$$\|\boldsymbol{A}\hat{\boldsymbol{x}} - \boldsymbol{b}\|^2 \le (1 + \varepsilon)\|\boldsymbol{A}\boldsymbol{x}_\star - \boldsymbol{b}\|^2.$$

The arithmetic cost is

$$O\left(mn \log n + n^3 \log n + mn \log(D/\varepsilon)\right).$$

8.2.3 Sketch & precondition

The Sketch-and-Solve and Iterative-Sketch paradigms use subspace embeddings to directly transform the problem data for (8.1). This section introduces the *Sketch-and-Precondition* approach, which operates in a different way: the subspace embedding is only used to accelerate the classical conjugate gradient method.

We proceed by sampling a δ-distortion subspace embedding $\boldsymbol{S}$ for range$(\boldsymbol{A})$, where δ is a modest constant. By Theorem 8.6, taking $d = \Theta(n \log n)$ allows us to sample such $\boldsymbol{S}$ with high probability. Form the dense matrix $\boldsymbol{Y} = \boldsymbol{S}\boldsymbol{A}$ in $O(mn \log n)$ time, and perform the QR factorization $\boldsymbol{Y} = \boldsymbol{Q}\boldsymbol{R}$ in additional $O(n^3 \log n)$ time.

We solve the least-squares problem (8.1) using preconditioned conjugate gradient, where the matrix $\boldsymbol{R}^*$ serves as the preconditioner. Since $\boldsymbol{R}$ is upper-triangular, we can compute matrix–vector products with $\boldsymbol{G} = (\boldsymbol{A}\boldsymbol{R}^{-1})^*(\boldsymbol{A}\boldsymbol{R}^{-1})$ in $O(nm)$ time. Thus, the complexity of each PCG iteration is only $O(mn)$. The postprocessing step of transforming the preconditioned solution back to the original domain is done in negligible $O(n^2)$ time. The total arithmetic cost of the Sketch-and-Precondition approach with k iterations is $O(mn \log n + n^3 \log n + mnk)$.

Now we bound the number k of iterations in terms of a desired precision ε.

Exercise 8.10 (Preconditioning by subspace embedding). Prove the two-sided linear matrix inequality

$$(1 - \delta)\boldsymbol{A}^*\boldsymbol{A} \preccurlyeq \boldsymbol{R}^*\boldsymbol{R} \preccurlyeq (1 + \delta)\boldsymbol{A}^*\boldsymbol{A} \tag{8.8}$$

where $\boldsymbol{R}$ is as defined above.

Continuing from the exercise, conjugate the relations in (8.8) by $\boldsymbol{R}^{-1}$, and substitute the definition of $\boldsymbol{G}$ to obtain

$$(1 - \delta)\boldsymbol{G} \preceq \boldsymbol{I} \preceq (1 + \delta)\boldsymbol{G}. \tag{8.9}$$

The linear matrix inequalities (8.9) ensure the condition number of $\boldsymbol{G}$ is bounded by $(1 + \delta)/(1 - \delta)$. We can easily take $\delta = 1/3$ so that $\kappa(\boldsymbol{G}) \le 2$.

Theorem 8.11 (Sketch and precondition). Let $\boldsymbol{A} \in \mathbb{F}^{m\times n}$ and $\boldsymbol{b} \in \mathbb{F}^n$ specify an instance of (8.1), let D denote the corresponding dynamic range of $(\boldsymbol{A}, \boldsymbol{b})$. With high probability, the Sketch-and-Precondition approach obtains a solution $\hat{\boldsymbol{x}}$ satisfying

$$\|\boldsymbol{A}\hat{\boldsymbol{x}} - \boldsymbol{b}\|^2 \le (1+\varepsilon)\|\boldsymbol{A}\boldsymbol{x}_\star - \boldsymbol{b}\|^2.$$

The arithmetic cost is

$$O(mn\log n + n^3 \log n + mn\log(D/\varepsilon)).$$

Proof. Most of the work in PCG is spent computing $\hat{\boldsymbol{y}}$ which approximates $\boldsymbol{y} = \boldsymbol{G}^{-1}\boldsymbol{h}$ for $\boldsymbol{h} = (\boldsymbol{A}\boldsymbol{R}^{-1})^*\boldsymbol{b}$. We need to address the nature of this approximation, and translate that approximation to the final solution $\hat{\boldsymbol{x}}$. To reduce clutter in notation, let $\boldsymbol{B} = \boldsymbol{A}\boldsymbol{R}^{-1}$, so that $\boldsymbol{G} = \boldsymbol{B}^*\boldsymbol{B}$ and $\boldsymbol{h} = \boldsymbol{B}^*\boldsymbol{b}$. Also set $\boldsymbol{P}$ as the orthogonal projector onto range$(\boldsymbol{B})$ = range$(\boldsymbol{A})$.

Theorem 8.3 gives us a bound on the error $E := \|\boldsymbol{G}^{1/2}(\hat{\boldsymbol{y}} - \boldsymbol{y})\|$; we need to express E in terms that are closer to the claim of the theorem. Begin by observing $\|\boldsymbol{G}^{1/2}(\hat{\boldsymbol{y}} - \boldsymbol{y})\| = \|\boldsymbol{B}(\hat{\boldsymbol{y}} - \boldsymbol{y})\|$. Next, consider how $\boldsymbol{y}$ can equivalently be expressed as $\boldsymbol{y} = \boldsymbol{B}^+\boldsymbol{b}$ where $\boldsymbol{B}^+$ is the pseudo-inverse of $\boldsymbol{B}$. This allows us to write

$$\boldsymbol{B}(\hat{\boldsymbol{y}} - \boldsymbol{y}) = \boldsymbol{B}\hat{\boldsymbol{y}} - \boldsymbol{B}\boldsymbol{B}^+\boldsymbol{b} = \boldsymbol{B}\hat{\boldsymbol{y}} - \boldsymbol{P}\boldsymbol{b} =: \boldsymbol{u}.$$

In order to express $E := \|\boldsymbol{u}\|$ in more useful terms, set $\boldsymbol{r} = (\boldsymbol{I} - \boldsymbol{P})\boldsymbol{b}$. Then check that

$$\|\boldsymbol{u}\|^2 + \|\boldsymbol{r}\|^2 = \|\boldsymbol{B}\hat{\boldsymbol{y}} - \boldsymbol{b}\|^2 \tag{8.10}$$

by using $\boldsymbol{u}^*\boldsymbol{r} = 0$ and $\boldsymbol{B}\hat{\boldsymbol{y}} - \boldsymbol{b} = \boldsymbol{u} - \boldsymbol{r}$. We use (8.10) with identities

$$\|\boldsymbol{r}\|^2 = \|\boldsymbol{A}\boldsymbol{x}_\star - \boldsymbol{b}\|^2 \quad \text{and} \quad \|\boldsymbol{B}\hat{\boldsymbol{y}} - \boldsymbol{b}\|^2 = \|\boldsymbol{A}\hat{\boldsymbol{x}} - \boldsymbol{b}\|^2$$

to establish $\|\boldsymbol{A}\hat{\boldsymbol{x}} - \boldsymbol{b}\|^2 - \|\boldsymbol{A}\boldsymbol{x}_\star - \boldsymbol{b}\|^2 = E^2$.

Now we turn to appropriately bounding E^2. Letting $\kappa = \kappa(\boldsymbol{G})$, Theorem 8.3 tells us that

$$E^2 \le 4\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^{2k} \|\boldsymbol{G}^{-1/2}\boldsymbol{h}\|^2.$$

One may then verify that $\|\boldsymbol{G}^{-1/2}\boldsymbol{h}\| = \|\boldsymbol{P}\boldsymbol{b}\|$. Taking this as given, we arrive at the inequality

$$\|\boldsymbol{A}\hat{\boldsymbol{x}} - \boldsymbol{b}\|^2 - \|\boldsymbol{A}\boldsymbol{x}_\star - \boldsymbol{b}\|^2 \le 4\theta^{2k}\|\boldsymbol{P}\boldsymbol{b}\|^2 \tag{8.11}$$

for $\theta = (\sqrt{\kappa}-1)/(\sqrt{\kappa}+1)$. The quantity $\theta < 1$ is bounded away from 1 in a manner independent of $\boldsymbol{A}$ and $\boldsymbol{b}$. For example, $\kappa \le 2$ when $\boldsymbol{S}$ embeds range$(\boldsymbol{A})$ with distortion $\delta = 1/3$.

The essence of the inequality (8.11) is the same as that of inequality (8.7) for the Iterative Sketch approach. Since the runtime claim of this theorem matches that of Theorem 8.9, we may conclude the proof. ∎

8.3 Runtime comparisions of the least-squares algorithms

We begin by summarizing the runtimes of the approaches from the previous sections. Each of these runtimes involves parameters m, n, and relative backward-error tolerance $0 < \varepsilon \ll 1$. In the case of Iterative Sketching and Sketch-and-Precondition, we also involve the dynamic range $D := \|\boldsymbol{P}\boldsymbol{b}\|/\|(\boldsymbol{I} - \boldsymbol{P})\boldsymbol{b}\|$ where $\boldsymbol{P}$ is the orthogonal projector onto range$(\boldsymbol{A})$. Naturally, we assume $\|\boldsymbol{P}\boldsymbol{b}\| < \|\boldsymbol{b}\|$ so that the dynamic range is finite. The runtimes are

- $O(mn \log n + n^3 \log(n)/\varepsilon^2)$ for Sketch-and-Solve;
- $O(mn \log n + n^3 \log n + mn \log(D/\varepsilon))$ for Iterative Sketching; and
- $O(mn \log n + n^3 \log n + mn \log(D/\varepsilon))$ for Sketch-and-Precondition.

It's clear that Sketch-and-Solve requires far more time than the other algorithms. However it *does* have a redeeming property that there is no dependence on D. In particular, it can handle the case when $\boldsymbol{A}\boldsymbol{x}_\star = \boldsymbol{b}$. Nevertheless, this property alone is not enough to make Sketch-and-Solve viable for practical levels of precision.

Our runtime bounds for Iterative Sketching and the Sketch-and-Precondition approaches match. However, Sketch-and-Precondition is likely superior to Iterative-Sketch when $\boldsymbol{A}$ is sparse. In such a situation, Sketch-and-Precondition still pays $O(mn \log n + n^3 \log n)$ time to form the preconditioner, but the complexity of a conjugate gradient step is reduced from $O(mn)$ to $O(n^2 + \text{nnz}(\boldsymbol{A}))$. Moreover, Sketch-and-Precondition can then be implemented using sparse matrix products instead of using the FFT algorithm to apply the SRFT embedding.

The classical QR-based algorithm for (8.1) takes time $O(mn^2)$, and it solves the least-squares problem to near machine precision ε_{m}. Treating ε_{m} and D as constants, both Sketch-and-Precondition and Iterative Sketching compare favorably to the classical method when $\log n \ll n \ll m/\log n$.

Remark 8.12 (Constants). Constant factors matter in numerical linear algebra, and not all convergence rates of the form "$O(\log(D/\varepsilon))$" are created equal. The exercises address convergence rates in more detail.

Remark 8.13 (Removing the dynamic range). While it is somewhat dubious to treat D as a constant, the exercises will show how modifications to Iterative Sketching or Sketch-and-Precondition can remove dependence on dependence on D.

8.4 Practical implementations and further reading

The conjugate gradient algorithm works wonderfully in exact arithmetic, however in finite precision there can be catastrophic rounding errors. Specialized implementations exist to reduce the effects of these rounding errors. For generic positive-definite systems, you should follow Appendix B of [She94]. For least-squares problems you should ideally use the LSQR algorithm [Pai82], but the simpler PCGLS algorithm is a reasonable alternative; see Section 7.4 of [Bjö96] for information on the latter method. Most programming languages have easy access to high quality implementations of these algorithms.

One issue we did not cover here is how to solve least-squares problems when $\boldsymbol{A}$ has linearly dependent columns. In such a setting, any of the standard conjugate gradient algorithms should converge to the optimal solution $\boldsymbol{A}^+\boldsymbol{b}$ as long as the initial point $\boldsymbol{x}_0$ belongs to the range of $\boldsymbol{A}^*$ [Bjö96]. Iterative Sketching should also work in this setting, although more care is required with the initial Cholesky factorization.

A slightly more direct proof of Theorem 8.11 is possible if you work with convergence bounds stated directly for least-squares conjugate gradients [Bjö96, Eqn. 7.4.6].

The Sketch-and-Precondition approach is a reinterpretation of an algorithm proposed by Rokhlin and Tygert [RT08]. The reported "typical" runtime of that algorithm is $O(mn \log n + n^3 + mn \log(1/\varepsilon))$. We note this runtime is called "typical" because it uses $d \sim n$, rather than the theoretical requirement $d \sim n^2$ stated in the paper's main technical lemma. The term $\log(1/\varepsilon)$ is meaningfully distinct from the $\log(D/\varepsilon)$ proven in these lecture notes. The difference can be attributed to how Rokhlin and Tygert's PCG algorithm is initialized, and it is explored in the exercises.

The idea of iterative sketching emerged from work on the randomized Kaczmarz method [SV09], which is a variant of stochastic gradient descent for overdetermined least-squares problems. Gower & Richtárik reinterpreted this approach as an iterative sketching technique [GR15]. The specific approach here was proposed by Mert Pilanci and collaborators in [PW16] and elaborated in [LP19]. In [PW16, Lemma 1, displays 22b and 23], the sketching matrices are resampled at each iteration, and the embedding dimension d is required to be on the order of $O(n(\log n)^4)$. [LP19] addresses the case where the sketching matrix is only sampled once, but then it modifies the resulting algorithm to take step sizes $\boldsymbol{x} = \boldsymbol{x} + \eta \boldsymbol{u}$ for $\eta \neq 1$. The effect of this revision is a modest improvement in the convergence rate, relative to the one described here.

The article [PW17] specifically addresses sketching in the context of convex optimization. The approach of that article is along the lines of Iterative Sketching, where the decision variable $\boldsymbol{x}$ (and hence the nominal Hessian $\nabla^2 f(\boldsymbol{x})$) is updated with each solution to (8.4).

Warning 8.14 Our parameters (d, m, n) correspond to (m, n, d) in [PW16], [PW17], and [LP19]. ▪

Problems and additional exercises

Exercise 8.15 (Relative performance of methods). Suppose Iterative Sketching and Sketch-and-Precondition are both used to solve the same (dense) least squares problem, with the same sketching matrix $\boldsymbol{S}$. Is there reason to believe that either algorithm will terminate before the other? If so, what can you say about the ratio of the faster algorithm's runtime divided by the slower algorithm's runtime? Justify your answer rigorously.

Exercise 8.16 (Removing the dynamic range). Let $\boldsymbol{S}$ be a sketching matrix for range$(\boldsymbol{A})$ with distortion $\delta = 1/4$. The current formulations of Iterative Sketching and Sketch-and-Precondition essentially initialize $\boldsymbol{x}_0 = \boldsymbol{0}$. Suppose instead that $\boldsymbol{x}_0$ is the result of running Sketch-and-Solve with embedding matrix $\boldsymbol{S}$. Address how the runtimes of both Iterative Sketching and Sketch-and-Precondition would change with this new initialization. (Note: for convergence of conjugate gradient method in Theorem 8.3, the term $\|\boldsymbol{G}^{-1/2}\boldsymbol{h}\|$ is a simplification of $\|\boldsymbol{G}^{1/2}(\boldsymbol{z}_0 - \boldsymbol{z}_\star)\|$.")

Lecture bibliography

[Bjö96] Å. Björck. *Numerical Methods for Least Squares Problems*. Society for Industrial and Applied Mathematics, Jan. 1996. DOI: 10.1137/1.9781611971484.

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

[LP19] J. Lacotte and M. Pilanci. *Faster Least Squares Optimization*. 2019. eprint: arXiv:1911.02675.

[Pai82] M. A. Paige Christopher C. and. "LSQR: An Algorithm for Sparse Linear Equations and Sparse Least Squares". In: *ACM Trans. Math. Softw.* 8.1 (Mar. 1982), pages 43–71. DOI: 10.1145/355984.355989.

[PW16] M. Pilanci and M. J. Wainwright. "Iterative Hessian Sketch: Fast and Accurate Solution Approximation for Constrained Least-Squares". In: *J. Mach. Learn. Res.* 17.1 (Jan. 2016), pages 1842–1879.

[PW17] M. Pilanci and M. J. Wainwright. "Newton Sketch: A Near Linear-Time Optimization Algorithm with Linear-Quadratic Convergence". In: *SIAM Journal on Optimization* 27.1 (Jan. 2017), pages 205–245. DOI: 10.1137/15m1021106.

[RT08] V. Rokhlin and M. Tygert. “A fast randomized algorithm for overdetermined linear least-squares regression”. In: *Proceedings of the National Academy of Sciences* 105.36 (Sept. 2008), pages 13212–13217. DOI: 10.1073/pnas.0804869105.

[She94] J. R. Shewchuk. *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*. Technical report. USA, 1994.

[SV09] T. Strohmer and R. Vershynin. “A randomized Kaczmarz algorithm with exponential convergence”. In: *J. Fourier Anal. Appl.* 15.2 (2009), pages 262–278. DOI: 10.1007/s00041-008-9030-4.

9. Randomized SVD

Date: 04 February 2020 **Scribe:** Jiace Sun

Agenda:

1. Truncated SVD
2. Two-phase SVD algorithms
3. Randomized rangefinder
4. Randomized subspace iteration
5. Randomized block Krylov

Today we are going to talk about one of the most widely used randomized linear algebra methods. Randomized SVD refers to a family of randomized algorithms for computing truncated singular value decompositions.

9.1 The truncated SVD

Every matrix $\boldsymbol{B} \in \mathbb{F}^{m \times n}$ has a singular value decomposition:

$$\boldsymbol{B} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^*,$$

where $\boldsymbol{U} \in \mathbb{F}^{n \times n}$ and $\boldsymbol{V} \in \mathbb{F}^{m \times m}$ are unitary and $\boldsymbol{\Sigma} = \operatorname{diag}(\sigma_1, \sigma_2, \ldots)$ is nonnegative, diagonal, and weakly decreasing. The columns of $\boldsymbol{U}$ are called left singular vectors, the columns of $\boldsymbol{V}$ are called right singular vectors, and the numbers $\sigma_1, \sigma_2, \ldots$ are called singular values. For concreteness, we assume that $m \geq n$.

We are going to be interested in a related matrix decomposition. For a parameter $k \leq n$, write

$$\boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_1 & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Sigma}_2 \end{bmatrix} \quad \text{with} \quad \boldsymbol{\Sigma}_1 \in \mathbb{R}^{k \times k} \quad \text{and} \quad \boldsymbol{\Sigma}_2 \in \mathbb{R}^{(n-k) \times (n-k)}.$$

The matrix $\boldsymbol{\Sigma}_1 = \operatorname{diag}(\sigma_1, \ldots, \sigma_k)$ lists the largest k singular values. We conformally decompose the singular vector matrices as

$$\boldsymbol{U} = \begin{bmatrix} \boldsymbol{U}_1 & \boldsymbol{U}_2 \end{bmatrix} \quad \text{and} \quad \boldsymbol{V} = \begin{bmatrix} \boldsymbol{V}_1 & \boldsymbol{V}_2 \end{bmatrix},$$

where $\boldsymbol{U}_1$ and $\boldsymbol{V}_1$ each have k columns. The *k-truncated SVD* is the matrix approximation:

k-truncated SVD

$$[\![\boldsymbol{B}]\!]_k := \boldsymbol{U}_1 \boldsymbol{\Sigma}_1 \boldsymbol{V}_1^*$$

The matrices $\boldsymbol{U}_1 \in \mathbb{F}^{n \times k}$ and $\boldsymbol{V}_1 \in \mathbb{F}^{m \times k}$ have orthonormal columns and $\boldsymbol{\Sigma}_1 \in \mathbb{R}^{k \times k}$ is nonnegative, diagonal and weakly decreasing.

9.1.1 Uniqueness

The singular values are always determined uniquely. But the SVD is never unique. Indeed, we can always switch column signs to obtain an ostensibly different factorization. When there is a duplicated singular value, we can also choose singular vectors to form arbitrary bases for the singular subspaces associated with that singular value. The truncated SVD inherits this lack of uniqueness.

Warning 9.1 (Uniqueness). Since the truncated SVD is not unique, the symbol $[\![\boldsymbol{B}]\!]_k$ is not well-defined. We use it informally to represent any k-truncated SVD. Typically, we will use this notation in contexts where the statement is valid for any k-truncated SVD. ▪

9.1.2 Applications

The truncated SVD plays a fundamental role in matrix approximation because of the following classic theorem.

Fact 9.2 (Eckart–Young, Mirsky). The k-truncated SVD gives a *best rank-k approximation* of a matrix $\boldsymbol{B}$ with respect to the Frobenius norm and the spectral norm. More precisely, *best rank-k approximation*

$$\min_{\mathrm{rk}(\hat{\boldsymbol{B}})=k} \|\boldsymbol{B} - \hat{\boldsymbol{B}}\|_{\mathrm{F}} = \|\boldsymbol{B} - [\![\boldsymbol{B}]\!]_k\|_{\mathrm{F}} = \left(\sum\nolimits_{j>k} \sigma_j\right)^{1/2} \qquad \text{(Eckart–Young)};$$

$$\min_{\mathrm{rk}(\hat{\boldsymbol{B}})=k} \|\boldsymbol{B} - \hat{\boldsymbol{B}}\| = \|\boldsymbol{B} - [\![\boldsymbol{B}]\!]_k\| = \sigma_{k+1} \qquad \text{(Mirsky)}.$$

A similar statement holds for every unitarily invariant norm. ▪

The truncated SVD has a wide range of applications in linear algebra, statistics, and related fields. These include

- Principal component analysis (PCA), total least-squares (TLS), proper orthogonal decomposition (POD), Karhunen–Loève expansions;
- Computations for ordinary least-squares, regularized least-squares problems (ridge regression or Tykhonov regularization), and regularized solution of linear systems;
- Data reduction, feature extraction, compression, and visualization.

9.1.3 Classical SVD algorithms and runtime comparison

The truncated SVD is actually somewhat challenging to compute in the dense case. By combining column pivoted QR, bidiagonal reduction, and bidiagonal SVD algorithms, we can obtain a method for computing the k-truncated SVD of an $m \times n$ matrix that $O(mnk)$ arithmetic cost.

There are also algorithms designed for computing the k-truncated SVD of a sparse matrix. These methods are based on Lanczos bidiagonalization, and they require k matrix–vector multiplies in sequence plus $O(k(m+n))$ extra arithmetic. These methods require care to implement properly.

Randomized SVD algorithms *do not* achieve better complexity. Indeed, they require a small number of $m \times n \times k$ matrix–matrix multiplies, plus $O((m+n)k)$ extra arithmetic. This profile resembles an algorithm for sparse SVD computation. But the matrix–matrix multiplies are practically much more efficient than a sequence of matrix–vector multiplies. Indeed, most computational platforms have highly optimized matrix multiplication routines, and there is commodity hardware (for example, GPUs) that can achieve very high speed for this primitive. As a consequence, randomized SVD methods can be much faster than classical SVD algorithms.

9.2 Two-phase randomized SVD algorithms and the rangefinder problem

Randomized SVD algorithms are based on a two-phase approach:

(A) Find a subspace aligned with the span of the dominant left-singular vectors, range($\boldsymbol{U}_1$).
(B) Compress the matrix to this subspace, and compute its SVD.

This procedure instantiates the motto "figure out where to look, and go look there."

9.2.1 The rangefinder

rangefinder problem

The first step is called the *rangefinder problem*. Given a matrix $\boldsymbol{B} \in \mathbb{F}^{m \times n}$, a target rank k, and a number $\ell = k + p$ of samples, we want to find an orthonormal matrix $\boldsymbol{Q} \in \mathbb{F}^{m \times \ell}$ such that range($\boldsymbol{Q}$) $\approx$ range($\boldsymbol{U}_1$). Heuristically, we want

$$\|\boldsymbol{B} - \boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{B}\| \lessapprox \sigma_{k+1}.$$

The key idea is to choose $\ell = k + p$ slightly larger than the actual target rank k. Note that the new matrix $\boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{B}$ has rank at most ℓ. Mirsky's theorem (Fact 9.2) imposes a fundamental restrictions on this procedure: the approximation error must be at least $\sigma_{\ell+1}$. By allowing $\ell > k$ and aiming for an error σ_{k+1}, we can develop very reliable algorithms.

9.2.2 The reduced SVD computation

Given a solution $\boldsymbol{Q}$ to the rangefinder problem, we can approximate the k-truncated SVD of the matrix $\boldsymbol{B}$.

1 Form $\boldsymbol{C} = \boldsymbol{Q}^*\boldsymbol{B} \in \mathbb{F}^{\ell \times n}$ using one $\ell \times m \times n$ matrix–matrix multiply with $\boldsymbol{B}^*$. Note that we do need to apply the adjoint $\boldsymbol{B}^*$.
2 Compute the dense $\ell \times n$ full SVD

$$\boldsymbol{C} = \hat{\boldsymbol{U}}\hat{\boldsymbol{\Sigma}}\hat{\boldsymbol{V}}^*$$

This step requires $O(\ell^2 n)$ arithmetic, but $\ell \ll m$.
3 We obtain an approximate truncated SVD of the original matrix in factored form:

$$\hat{\boldsymbol{B}} = (\boldsymbol{Q}\hat{\boldsymbol{U}})\hat{\boldsymbol{\Sigma}}\hat{\boldsymbol{V}}^*$$

using a single $m \times \ell \times \ell$ matrix multiplication.
4 Note that $\hat{\boldsymbol{B}}$ is actually a rank-ℓ approximate SVD. We can truncate $\hat{\boldsymbol{\Sigma}}$ to $[\![\hat{\boldsymbol{\Sigma}}]\!]_k$ to obtain a rank-k approximation of the k-truncated SVD. The cost of this optional truncation is negligible, but the corresponding analysis turns out to be more involved.

The total cost is one $\ell \times m \times n$ matrix–matrix multiplication, plus $O((m + n)\ell)$ additional arithmetic. Modulo rounding errors, the error in the approximate SVD $\hat{\boldsymbol{B}}$ is the same as the approximation in the rangefinder step:

$$\|\boldsymbol{B} - \hat{\boldsymbol{B}}\| = \|\boldsymbol{B} - \boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{B}\|.$$

Therefore, it suffices to solve the rangefinder problem.

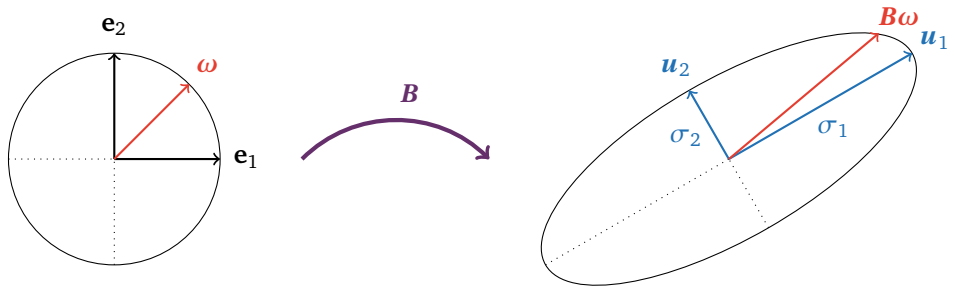


Figure 9.1 (Geometry of the randomized rangefinder). The SVD has a geometric meaning. The matrix $\boldsymbol{B}$ acts on the unit ball in $\mathbb{F}^n$ (left) by transforming it into an ellipsoid (right). The left singular vectors ($\boldsymbol{u}_1$ and $\boldsymbol{u}_2$) indicate the direction of the principal axes, while the corresponding singular values are the lengths of the semiaxes. If we input a random vector (red), its image is going to roughly align with the longer axes. Repeating this procedure several times and orthogonalizing the resulting vectors will produce a basis whose span is close to range($\boldsymbol{U}_1$). This picture is completely accurate for matrices with perfectly low rank. If this is not the case, the random vectors $\boldsymbol{B}\boldsymbol{\omega}_k$ will have some components along the shorter axes. By choosing additional random vectors ($\ell > k$), we can insulate the procedure against these devations.

9.3 The randomized rangefinder

So, how do we actually solve the rangefinder problem to execute phase **(A)**? It should come as no surprise that the answer involves randomness. The approach is based on geometric intuition, illustrated in Figure 9.1: If we multiply random vectors $\boldsymbol{\omega}_1, \dots, \boldsymbol{\omega}_\ell$ into $\boldsymbol{B}$ and orthogonalize, we get a subspace that aligns well with the range of $\boldsymbol{U}_1$. We can translate this intuition into a computational procedure, called the *randomized rangefinder*:

randomized rangefinder

1 Draw a random test matrix $\boldsymbol{\Omega} \in \mathbb{F}^{n \times \ell}$ (ℓ samples).
2 Compute $\boldsymbol{Y} = \boldsymbol{B}\boldsymbol{\Omega}$ (one $m \times n \times \ell$ matrix multiplication)
3 Orthogonalize: $\boldsymbol{Y} = \boldsymbol{Q}\boldsymbol{R}$ (one $m \times \ell$ economic QR).

The total cost is dominated by the single $m \times n \times \ell$ matrix multiplication. Typically, the additional cost $O(m\ell^2)$ is much smaller by comparison.

9.3.1 Implementation issues

This simple description of the randomized rangefinder masks a number of significant decisions about the implementation

What test matrix?

It turns out Gaussian test matrices work fabulously. But sometimes, one can use an SRFT (which works even better), or a sparse sign matrix (which works roughly the same) and other candidates for random embeddings. Gaussians are actually pretty natural in this setting.

What matrix multiplication?

Exploit advantageous structure in $\boldsymbol{B}$ (e.g., sparsity or a fast multiply) whenever possible. There are instances where structure in $\boldsymbol{\Omega}$ can be exploited as well. Unfortunately, the

second phase in the randomized SVD procedure involves the basis $\boldsymbol{Q}$, which is typically dense and unstructured.

How should we orthogonalize?

It is important to use stable procedures. Most of the vectors $\boldsymbol{B}\boldsymbol{\omega}_i$ will be aligned in similar directions. Use Householder orthogonalization, or double Gram–Schmidt, or another reliable method. Do not use (modified) Gram–Schmidt.

How much oversampling?

If k is known, it often suffices to take $\ell = k + p$ where $p = 5$ or $p = 10$ or maybe even $p = k$. The theory gives detailed justification for these choices. If k is not known, use randomized error estimation (based on randomized trace estimation), and adaptively increase ℓ until you get satisfactory results.

Remark 9.3 (Matlab). These ideas lend themselves to straightforward Matlab implementations. If you use the built-in functions, the randomized SVD strategy will work! This is one reason that the method has had a practical impact.

9.4 Analysis

We will present a detailed analysis of the randomized rangefinder problem in the next lecture. This argument leads to an informative result [HMT11].

Theorem 9.4 (Randomized rangefinder). Let $\boldsymbol{B} \in \mathbb{R}^{m\times n}$ be a matrix with singular values $\sigma_1 \geq \sigma_2 \geq \cdots$. Let $\boldsymbol{\Omega} \in \mathbb{R}^{n\times \ell}$ be standard normal. For each $k < \ell - 1$, the randomized rangefinder produces an orthogonal matrix $\boldsymbol{Q}$ with the following property:

$$\mathbb{E}_{\boldsymbol{\Omega}} \|\boldsymbol{B} - \boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{B}\| \leq \left(1 + \sqrt{\frac{k}{\ell - k - 1}}\right)\sigma_{k+1} + \frac{\mathrm{e}\sqrt{\ell}}{\ell - k}\left(\sum\nolimits_{j>k} \sigma_j^2\right)^{1/2}$$

This error bound features two constituents. The first term resembles the spectral-norm error from Mirsky's theorem; it shrinks slowly to zero as ℓ increases. The second term resembles the Frobenius-norm error from the Eckart–Young theorem; it decreases more quickly as the oversampling increases. If ℓ is very close to k, both contributions can be large. But, if we allow ℓ to be just a bit bigger than k, the second factor moderates quickly.

9.5 Randomized subspace iteration

The bounds in Theorem 9.4 suggest that the rangefinder procedure is most effective when the matrix $\boldsymbol{B}$ exhibits strong spectral decay; that is, $\sum_{j>k} \sigma_j^2$ is comparable with σ_{k+1}. This insight is correct. Moreover, the converse is also true: the rangefinder tends to produce large errors when the spectral tail has a lot of energy. In this case, we can improve the performance of the rangefinder by augmenting it with powering.

randomized subspace iteration

The following method is called *randomized subspace iteration*. It is based on a classic approach from numerical linear algebra. Let ℓ be the number of samples, and let q be a power parameter.

1 Draw a random test matrix $\boldsymbol{\Omega} \in \mathbb{F}^{m\times \ell}$, and set $\boldsymbol{Q} = \boldsymbol{\Omega}$. We have changed the dimension for reasons that will become clear in a moment.
2 For $i = 1, 2, \ldots, q$,

1 Construct $\boldsymbol{Y} = \boldsymbol{B}(\boldsymbol{B}^*\boldsymbol{Q})$
2 Compute $[\boldsymbol{Q}, \sim] = \texttt{qr\_econ}(\boldsymbol{Y})$

The idea behind this approach is the following insight. Effectively, the algorithm applies the randomized rangefinder to the matrix power. $|\boldsymbol{B}|^{2q} := (\boldsymbol{B}\boldsymbol{B}^*)^q$. Matrix powers suppress small eigenvalues very quickly. This procedure has the effective of reducing the energy in the spectral tail.

9.5.1 Analysis

The analysis of randomized subspace iteration follows as a corollary of the analysis of the randomized rangefinder. We need the following lemma.

Lemma 9.5 (Powering). Suppose that $\boldsymbol{P}$ is an orthogonal projector. For any matrix $\boldsymbol{B}$ with compatible dimensions,

$$\|(\mathbf{I} - \boldsymbol{P})\boldsymbol{B}\|^{2q} \le \|(\mathbf{I} - \boldsymbol{P})\,|\boldsymbol{B}|^{2q}\| \quad \text{for } q \ge 1.$$

Exercise 9.6 Prove this claim by using the Araki–Lieb–Thirring inequality.

As a consequence of Theorem 9.4 and Lemma 9.5, we immediately arrive at the following result.

Corollary 9.7 (Randomized subspace iteration). Instate the hypotheses of Theorem 9.4. Let $\boldsymbol{Q}$ be the orthonormal basis computed by the randomized subspace iterations after q steps. Then

$$\mathbb{E}\,\|\boldsymbol{B} - \boldsymbol{Q}\boldsymbol{Q}^*\| \le \left[\left(1 + \sqrt{\frac{k}{\ell + k}}\right)\sigma_{k+1}^{2q} + \frac{\mathrm{e}\sqrt{\ell}}{\ell - k}\left(\sum\nolimits_{j>k}\sigma_j^{4q}\right)^{1/2}\right]^{1/(2q)}.$$

This statement implies that powering drives the error in the rangefinder to σ_{k+1} exponentially fast as the power q increases. It is always sufficient to take $q = \log(m \wedge n)$, because at that point everything else gets "destroyed," and one copy of σ_{k+1} remains (with a constant close to one). In practice, $q = 2$ or $q = 3$ is entirely adequate for most examples.

9.6 Randomized Krylov methods

To achieve very high accuracy, randomized subspace iteration may not be sufficient. Instead, we can use a *randomized block Krylov* method. We give a brief summary of the ideas here, but implementations require further attention.

randomized block Krylov

The block Krylov method collects all of the information acquired during the execution of randomized subspace iteration:

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

Then we compute a QR factorization of the sample matrix: $\boldsymbol{Y} = \boldsymbol{Q}\boldsymbol{R}$. The matrix $\boldsymbol{Q}$ is very well aligned with the dominant left singular subspace of $\boldsymbol{B}$. We can use this matrix $\boldsymbol{Q}$ to compute an approximate k-truncated SVD $\hat{\boldsymbol{B}}$ of the input matrix.

A detailed analysis of this approach is beyond the scope of this course. The singular values of the approximation satisfy heuristic error bounds

$$|\sigma_j(\boldsymbol{B}) - \sigma_j(\hat{\boldsymbol{B}})| \lessapprox \frac{\sigma_j(\boldsymbol{B})}{q^2} \quad \text{for } j < \ell.$$

In contrast, randomized subspace iteration only yields a bound like

$$|\sigma_j(\boldsymbol{B}) - \sigma_j(\hat{\boldsymbol{B}})| \lessapprox \frac{\sigma_j(\boldsymbol{B})}{q} \quad \text{for } j < \ell.$$

This improvement is similar to what we saw when comparing the randomized power method with the randomized Krylov method for computing the maximum eigenvalue. It indicates that we can obtain much better relative accuracy by means of a Krylov method.

Lecture bibliography

[HMT11] N. Halko, P. G. Martinsson, and J. A. Tropp. "Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions". In: *SIAM Rev.* 53.2 (2011), pages 217–288. ISSN: 0036-1445. DOI: 10.1137/090771806.

10. Randomized Rangefinder: Analysis

Date: 06 February 2020 **Scribe:** Richard Kueng

Last time we discussed the randomized SVD algorithm. At the core of this technique is the *randomized rangefinder*. Today, we will discuss this important subroutine and will derive strong performance guarantees. These are based on a deterministic analysis using Schur complements. Randomness will then allow us to find simple and intuitive bounds on the relevant parameters.

Agenda:

1. RRF proof strategy
2. Schur complements
3. Deterministic bounds
4. Gaussian test matrix

10.1 The randomized rangefinder (RFF)

randomized rangefinder (RRF)

Recall that the *randomized rangefinder (RRF)* is designed to capture the action of a matrix. Here is a quick review of the procedure and the main result that we will establish.

10.1.1 Procedure

Let $\boldsymbol{B} \in \mathbb{F}^{m \times n}$ be a fixed target / input matrix. Draw a random test matrix $\boldsymbol{\Omega} \in \mathbb{F}^{m \times \ell}$, where ℓ is the number of samples. When computing a truncated SVD, the number ℓ is typically close to the rank of the truncated SVD. Form the sample matrix $\boldsymbol{Y} = \boldsymbol{B}\boldsymbol{\Omega} \in \mathbb{F}^{m \times \ell}$, and compute its QR decomposition: $\boldsymbol{Y} = \boldsymbol{Q}\boldsymbol{R}$, where $\boldsymbol{Q} \in \mathbb{F}^{m \times \ell}$ is an orthonormal basis for the range of $\boldsymbol{Y}$.

10.1.2 Error bounds

To quantify how well the rangefinder (RRF) works, that is, how well it captures the action of $\boldsymbol{B}$, we need a bound of the form

$$\|(\mathbf{I} - \boldsymbol{P}_{\boldsymbol{Y}})\boldsymbol{B}\|$$

where $\boldsymbol{P}_{\boldsymbol{Y}} = \boldsymbol{Q}\boldsymbol{Q}^*$ is the orthoprojector onto the range of $\boldsymbol{Y}$. This type of estimate controls the performance of the RRF, and it also leads directly to estimates for the approximation error in the randomized SVD. The following result, developed by Halko et al. [HMT11], establishes strong bounds for Gaussian test matrices.

Theorem 10.1 (Randomized rangefinder). Let $\boldsymbol{B} \in \mathbb{R}^{m\times n}$ be a matrix with singular values $\sigma_1 \geq \sigma_2 \geq \cdots$. Let $\boldsymbol{\Omega} \in \mathbb{R}^{n\times \ell}$ be standard normal. For each $k < \ell - 1$, the randomized rangefinder produces an orthogonal matrix $\boldsymbol{Q}$ such that the orthoprojector $\boldsymbol{P_Y} = \boldsymbol{QQ}^*$ obeys

$$\mathbb{E}_{\boldsymbol{\Omega}} \|(\boldsymbol{I} - \boldsymbol{P_Y})\boldsymbol{B}\| \leq \left(1 + \sqrt{\frac{k}{\ell - k - 1}}\right)\sigma_{k+1} + \frac{\mathrm{e}\sqrt{\ell}}{\ell - k}\left(\sum\nolimits_{j>k}\sigma_j^2\right)^{1/2}$$

This bound has two constituents: a spectral-norm type error and a Frobenius-norm type error. It turns out that this mixed error bound is unavoidable, even if the left hand side only features the spectral norm.

10.1.3 Proof strategy

We will prove most of Theorem 10.1, as well as some related things. The overall proof strategy is simple:

1. Understand the form of the error and single out extremal target matrices.
2. Compute a deterministic bound on the error.
3. Use random matrix theory to get explicit results for Gaussian test matrices.

We will phrase most of the argument for Frobenius norm errors, not the operator norm. The analysis is slightly simpler than the spectral norm case, and the two results are similar in spirit.

10.2 Step 1: Schur complements and extreme cases

The rangefinder procedure finds its most natural expression using the language of Schur complements. For the moment, we will work in either the real or complex field.

Definition 10.2 (Schur complement). Let $\boldsymbol{A} \in \mathbb{H}_n$ be a psd matrix and let $\boldsymbol{X} \in \mathbb{F}^{n\times k}$ be a test matrix. The *Schur complement* is the matrix

Schur complement

$$\boldsymbol{A}/\boldsymbol{X} = \boldsymbol{A} - (\boldsymbol{AX})\,(\boldsymbol{X}^*\boldsymbol{AX})^\dagger\,(\boldsymbol{AX})^*, \tag{10.1}$$

where $\dagger$ denotes the pseudo-inverse.

There are a lot of things to say about Schur complements and their properties; see the homework for an introduction. Schur complements occur naturally in block Gaussian elimination and Cholesky decompositions. Other applications include partial least squares, as well as Nyström decompositions.

We can write the error in the rangefinder procedure in terms of a Schur complement.

Proposition 10.3 (Rangefinder error). Fix a target $\boldsymbol{B} \in \mathbb{F}^{m\times n}$ and let $\boldsymbol{X} \in \mathbb{F}^{n\times \ell}$ be a test matrix. Define the (matrix) error in the rangefinder:

$$\boldsymbol{E} := \boldsymbol{E}(\boldsymbol{B}, \boldsymbol{X}) = (\mathbf{I} - \boldsymbol{P_{BX}})\,\boldsymbol{B} \tag{10.2}$$

Then the squared error matrix takes the form

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

Proof. Recall that the orthogonal projector onto the range of $\boldsymbol{Y}$ can be written as $\boldsymbol{P_Y} = \boldsymbol{Y}\,(\boldsymbol{Y}^*\boldsymbol{Y})^\dagger\,\boldsymbol{Y}^*$. Hence,

$$\boldsymbol{P_{BX}} = (\boldsymbol{BX})\,(\boldsymbol{X}^*\boldsymbol{B}^*\boldsymbol{BX})^\dagger\,(\boldsymbol{BX})^*.$$

Computing the squared absolute value reveals a Schur complement (10.1):

$$\begin{aligned} \boldsymbol{E}^*\boldsymbol{E} &= \boldsymbol{B}^* (\mathbf{I} - \boldsymbol{P}_{\boldsymbol{BX}}) \boldsymbol{B} = \boldsymbol{B}^*\boldsymbol{B} - \boldsymbol{B}^*\boldsymbol{P}_{\boldsymbol{BX}}\boldsymbol{B} \\ &= \boldsymbol{B}^*\boldsymbol{B} - (\boldsymbol{B}^*\boldsymbol{B}\boldsymbol{X}) (\boldsymbol{X}^*\boldsymbol{B}^*\boldsymbol{B}\boldsymbol{X})^\dagger (\boldsymbol{B}^*\boldsymbol{B}\boldsymbol{X})^* \\ &= (\boldsymbol{B}^*\boldsymbol{B}) / \boldsymbol{X}. \end{aligned}$$

This is the required result. ∎

This reformulation implies several interesting properties. First, observe that the error in the rangefinder is a monotone increasing function of the square of the input matrix.

Corollary 10.4 (Rangefinder: Monotonicity). Suppose that $\boldsymbol{B}^*\boldsymbol{B} \preccurlyeq \boldsymbol{C}^*\boldsymbol{C}$. Then $|\boldsymbol{E}(\boldsymbol{B}, \boldsymbol{X})|^2 \preccurlyeq |\boldsymbol{E}(\boldsymbol{C}, \boldsymbol{X})|^2$.

Proof. The Schur complement with respect to a fixed test matrix $\boldsymbol{X}$ is operator monotone. See the homework. ∎

This monotonicity result allows us to identify *extremals*, or worst-case target matrices, for the rangefinder. *extremals*

Corollary 10.5 (Extremals). Let $\boldsymbol{B} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^*$, where $\boldsymbol{\Sigma} = \operatorname{diag}(\sigma_1, \sigma_2, \ldots)$. Fix k and define $\boldsymbol{C} = \boldsymbol{U}\bar{\boldsymbol{\Sigma}}\boldsymbol{V}^*$, where $\bar{\boldsymbol{\Sigma}} = \operatorname{diag}(\sigma_1, \ldots, \sigma_1, \sigma_{k+1}, \sigma_{k+2}, \ldots)$. Then

$$|\boldsymbol{E}(\boldsymbol{B}, \boldsymbol{X})|^2 \preccurlyeq |\boldsymbol{E}(\boldsymbol{C}, \boldsymbol{X})|^2.$$

Proof. By construction, $\boldsymbol{B}^*\boldsymbol{B} \preccurlyeq \boldsymbol{C}^*\boldsymbol{C}$. ∎

We conclude this section with another fact about Schur complements that will play a role in the analysis.

Fact 10.6 (Schur complements: Inclusion). Fix a psd matrix $\boldsymbol{A}$. Let $\boldsymbol{X}, \boldsymbol{Z}$ be test matrices. If $\operatorname{range}(\boldsymbol{Z}) \subset \operatorname{range}(\boldsymbol{X})$, then $\boldsymbol{A}/\boldsymbol{X} \preccurlyeq \boldsymbol{A}/\boldsymbol{Z}$. ∎

In other words, if $\boldsymbol{Z}$ is smaller than $\boldsymbol{X}$, then the Schur complement with respect to $\boldsymbol{Z}$ removes less stuff from $\boldsymbol{A}$ than the Schur complement with respect to $\boldsymbol{X}$.

10.3 Step 2: Deterministic bounds

In this section, we will develop a deterministic bound for the rangefinder that operates for an arbitrary test matrix. In the next section, we will instantiate this result using a random test matrix.

This approach requires a fair amount of notation. Let $\boldsymbol{B} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^*$ be a SVD of the target matrix. For a parameter $k \le \ell$, write

$$\boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_1 & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Sigma}_2 \end{bmatrix} \quad \text{with} \quad \boldsymbol{\Sigma}_1 \in \mathbb{R}^{k \times k} \quad \text{and} \quad \boldsymbol{\Sigma}_2 \in \mathbb{R}^{(n-k)\times(n-k)}.$$

Equivalently, $\boldsymbol{\Sigma}_1 = \operatorname{diag}(\sigma_1, \ldots, \sigma_k)$ and $\boldsymbol{\Sigma}_2 = \operatorname{diag}(\sigma_{k+1}, \sigma_{k+2}, \ldots)$. We conformally decompose the singular vector matrices as

$$\boldsymbol{U} = \begin{bmatrix} \boldsymbol{U}_1 & \boldsymbol{U}_2 \end{bmatrix} \quad \text{and} \quad \boldsymbol{V} = \begin{bmatrix} \boldsymbol{V}_1 & \boldsymbol{V}_2 \end{bmatrix},$$

and we extend this partition to the test matrix $\boldsymbol{X}$:

$$\boldsymbol{Y} = \boldsymbol{B}\boldsymbol{X} = \begin{bmatrix} \boldsymbol{U}_1 & \boldsymbol{U}_2 \end{bmatrix} \begin{bmatrix} \boldsymbol{\Sigma}_1 & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Sigma}_2 \end{bmatrix} \begin{bmatrix} \boldsymbol{V}_1^*\boldsymbol{X} \\ \boldsymbol{V}_2^*\boldsymbol{X} \end{bmatrix} =: \begin{bmatrix} \boldsymbol{U}_1 & \boldsymbol{U}_2 \end{bmatrix} \begin{bmatrix} \boldsymbol{\Sigma}_1 & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Sigma}_2 \end{bmatrix} \begin{bmatrix} \boldsymbol{X}_1 \\ \boldsymbol{X}_2 \end{bmatrix}.$$

To be more explicit, we have defined

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

From now on, we assume that $\boldsymbol{X}_1 \in \mathbb{F}^{k \times \ell}$ has full *row* rank. This assumption is natural (and essential).

The first matrix $\boldsymbol{X}_1$ tells us how much the test matrix is aligned with the k dominant right singular vectors of $\boldsymbol{B}$; this is the important part of $\boldsymbol{B}$. The second matrix $\boldsymbol{X}_2$ tells us how well $\boldsymbol{X}$ is aligned with the tail singular vectors of $\boldsymbol{B}$; this is the unimportant part of $\boldsymbol{B}$.

The following deterministic theorem relates the rangefinder to the interplay between $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$. This statement and proof are essentially due to Halko et al. [HMT11]. The argument has been streamlined using some more recent insights.

Theorem 10.7 (Deterministic error bound). The error (10.2) in the rangefinder satisfies the following bound:

$$\|\boldsymbol{E}(\boldsymbol{B}, \boldsymbol{X})\|_\xi^2 \le \|\boldsymbol{\Sigma}_2\|_\xi^2 + \|\boldsymbol{\Sigma}_2 \boldsymbol{X}_2 \boldsymbol{X}_1^\dagger\|_\xi^2.$$

Here, $\|\cdot\|_\xi$ denotes either the Frobenius or the spectral norm.

Note that the first term exactly reproduces the error in the best rank-k approximation of the matrix $\boldsymbol{B}$. The second term reflects the energy of $\boldsymbol{X}$ contained in the tail subspace range$(\boldsymbol{V}_2)$, weighted by the associated singular values listed in $\boldsymbol{\Sigma}_2$. The pseudo-inverted matrix reflects the conditioning of $\boldsymbol{X}_1$.

Example 10.8 (Ideal case). If $\boldsymbol{X} = \boldsymbol{V}_1$, then $\|\boldsymbol{\Sigma}_2 \boldsymbol{X}_2 \boldsymbol{X}_1^\dagger\|_\xi^2 = 0$. ∎

The proof of Theorem 10.7 is based on *perturbation theory*. We first isolate an ideal test matrix case and treat the concrete instance as a perturbation of the ideal.

perturbation theory

Proof of Theorem 10.7. We already know that $|\boldsymbol{E}(\boldsymbol{B}, \boldsymbol{X})|^2 = (\boldsymbol{B}^* \boldsymbol{B})/\boldsymbol{X}$. Let us start with some simplifications. Without loss, assume that $\boldsymbol{B}^* \boldsymbol{B}$ is diagonal; that is, $\boldsymbol{B}^* \boldsymbol{B} = \boldsymbol{\Sigma}^* \boldsymbol{\Sigma} = \operatorname{diag}(\sigma_1^2, \sigma_2^2, \ldots)$. Next, by Corollary 10.5, we can assume the worst-case distribution of singular values: $\sigma_1 = \sigma_2 = \cdots = \sigma_k$. Finally, by homogeneity of the error bound, we can rescale the matrix $\boldsymbol{B}$ so that $\sigma_1 = 1$.

In short, it is sufficient to produce a semidefinite upper bound for the matrix

$$\left(\mathbf{I}_k \oplus \boldsymbol{\Sigma}_2^2\right)/\boldsymbol{X} \quad \text{where} \quad \boldsymbol{\Sigma}_2^2 = \operatorname{diag}\left(\sigma_{k+1}^2, \sigma_{k+2}^2, \ldots\right).$$

The direct sum operator $\oplus$ constructs a block-diagonal matrix from its arguments.

For intuition about how to proceed, we consult the ideal case. If $\boldsymbol{X} = \boldsymbol{V}$, then

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{X}_1 \\ \boldsymbol{X}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{I} \\ \mathbf{0} \end{bmatrix}.$$

We can treat the actual situation as a perturbation of this idealized case. Define

$$\boldsymbol{Z} = \boldsymbol{X} \boldsymbol{X}_1^\dagger = \begin{bmatrix} \mathbf{I}_k \\ \boldsymbol{F} \end{bmatrix} \quad \text{where} \quad \boldsymbol{F} = \boldsymbol{X}_2 \boldsymbol{X}_1^\dagger.$$

This construction ensures range$(\boldsymbol{Z}) \subset$ range$(\boldsymbol{X})$. We can invoke Fact 10.6 to conclude

$$\left(\mathbf{I}_k \oplus \boldsymbol{\Sigma}_2^2\right)/\boldsymbol{X} \preccurlyeq \left(\mathbf{I}_k \oplus \boldsymbol{\Sigma}_2^2\right)/\boldsymbol{Z}.$$

This final Schur complement can be computed explicitly.

In the next calculation, we use the symbol $\square$ to indicate a block matrix that does not play a role in the argument:

$$\begin{aligned}\left(\mathbf{I}_k \oplus \boldsymbol{\Sigma}_2^2\right) / \boldsymbol{Z} &= \begin{bmatrix} \mathbf{I}_k & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Sigma}_2^2 \end{bmatrix} - \begin{bmatrix} \mathbf{I}_k \\ \boldsymbol{\Sigma}_2^2 \boldsymbol{F} \end{bmatrix} \left(\mathbf{I}_k + \boldsymbol{F}^* \boldsymbol{\Sigma}_2^2 \boldsymbol{F}\right)^{-1} \begin{bmatrix} \mathbf{I}_k \\ \boldsymbol{\Sigma}_2^2 \boldsymbol{F} \end{bmatrix}^* \\ &= \begin{bmatrix} \mathbf{I}_k - \left(\mathbf{I}_k + \boldsymbol{F}^* \boldsymbol{\Sigma}_2^2 \boldsymbol{F}\right)^{-1} & \square \\ \square & \boldsymbol{\Sigma}_2 (\mathbf{I}_k - \cdots) \boldsymbol{\Sigma}_2 \end{bmatrix} \\ &\preccurlyeq \begin{bmatrix} \boldsymbol{F}^* \boldsymbol{\Sigma}_2^2 \boldsymbol{F} & \square \\ \square & \boldsymbol{\Sigma}_2^2 \end{bmatrix}.\end{aligned}$$

The matrix abbreviated as $\cdots$ is psd, so the bottom-right block increases when we drop this term. We bound the top-left block by applying the numerical fact $1 - (1+a)^{-1} \le a$, valid for $a \ge 0$, to the eigenvalues of this block.

The Frobenius norm bound follows readily:

$$\|\boldsymbol{E}(\boldsymbol{B}, \boldsymbol{X})\|_F^2 \le \operatorname{tr}\left(\mathbf{I}_k \oplus \boldsymbol{\Sigma}_2^2\right) / \boldsymbol{Z} \le \operatorname{tr} \boldsymbol{\Sigma}_2^2 + \operatorname{tr} \boldsymbol{F}^* \boldsymbol{\Sigma}_2^2 \boldsymbol{F}.$$

Indeed, the trace is operator monotone. The spectral norm bound requires an additional argument, which we leave as an exercise. ∎

10.4 Random test matrices

Theorem 10.7 provides us with crucial insights about the rangefinder performance for any test matrix $\boldsymbol{X}$. It turns out that using a random test matrix leads to near-optimal bounds, regardless of the singular-value spectrum of the target matrix $\boldsymbol{B}$.

To obtain probabilistic results, we work with a standard normal test matrix $\boldsymbol{\Omega} \in \mathbb{F}^{n \times \ell}$. To make the computations, we will exploit the fact that $\boldsymbol{\Omega}_1 = \boldsymbol{V}_1^* \boldsymbol{\Omega}$ and $\boldsymbol{\Omega}_2 = \boldsymbol{V}_2^* \boldsymbol{\Omega}$ are statistically independent standard normal matrices. Moreover, the random matrix $\boldsymbol{\Omega}_1$ is usually well-conditioned if $k \ll \ell$. These insights already suffice to deduce a strong Frobenius-norm error bound. The next result is drawn from Halko et al. [HMT11].

Theorem 10.9 (Randomized rangefinder: Frobenius-norm error). Suppose that the test matrix $\boldsymbol{\Omega}$ is standard normal. For all $k < \ell - \alpha$, the error in the randomized rangefinder approximation obeys

$$\mathbb{E}_{\boldsymbol{\Omega}} \|(\mathbf{I} - \boldsymbol{P}_{\boldsymbol{Y}})\boldsymbol{B}\|_{\mathrm{F}}^2 \le \left(1 + \frac{k}{\ell - k - \alpha}\right) \left(\sum\nolimits_{j>k} \sigma_j^2\right).$$

The field parameter $\alpha = 1$ when $\mathbb{F} = \mathbb{R}$, while $\alpha = 0$ when $\mathbb{F} = \mathbb{C}$.

Proof. We begin with the bound from Theorem 10.7:

$$\mathbb{E}_{\boldsymbol{\Omega}} \|\boldsymbol{E}(\boldsymbol{B}, \boldsymbol{\Omega})\|_F^2 \le \|\boldsymbol{\Sigma}\|_{\mathrm{F}}^2 + \mathbb{E} \|\boldsymbol{\Sigma}_2 \boldsymbol{\Omega}_2 \boldsymbol{\Omega}_1^\dagger\|_{\mathrm{F}}^2.$$

We compute the expectation in two steps. First, use independence of $\boldsymbol{\Omega}_1$ and $\boldsymbol{\Omega}_2$ to take the expectation with respect to $\boldsymbol{\Omega}_2$. An easy computation (using rotational invariance) reveals that

$$\mathbb{E}_{\boldsymbol{\Omega}_2} \|\boldsymbol{\Sigma}_2 \boldsymbol{\Omega}_2 \boldsymbol{\Omega}_1^\dagger\|_{\mathrm{F}}^2 = \|\boldsymbol{\Sigma}_2\|_{\mathrm{F}}^2 \|\boldsymbol{\Omega}_1^\dagger\|_{\mathrm{F}}^2.$$

The remaining expectation equals the expected trace of a $k \times k$ inverted Wishart matrix with ℓ degrees of freedom. This expectation is a classical computation from multivariate statistics:

$$\mathbb{E}_{\boldsymbol{\Omega}_1} \|\boldsymbol{\Omega}_1^\dagger\|_{\mathrm{F}}^2 = \frac{k}{\ell - k - \alpha}.$$

Combine the displays to complete the argument. ∎

In the real case ($\mathbb{F} = \mathbb{R}$), we can also establish a probabilistic spectral-norm error bound. For all matrices with compatible dimensions, Chevet's theorem states that

$$\mathbb{E}_{\boldsymbol{\Omega}_2} \|\boldsymbol{S}\boldsymbol{\Omega}_2\boldsymbol{T}\|^2 \le \left(\|\boldsymbol{S}\|\|\boldsymbol{T}\|_{\mathrm{F}} + \|\boldsymbol{S}\|_{\mathrm{F}}\|\boldsymbol{T}\|\right)^2.$$

Using this result in the previous argument leads to the following result.

Theorem 10.10 (Randomized rangefinder: Spectral-norm error). Fix $\mathbb{F} = \mathbb{R}$. Let $\boldsymbol{\Omega} \in \mathbb{R}^{n \times \ell}$ be standard Gaussian, and define $W = \|\boldsymbol{\Omega}_1^\dagger\|^2$. For all $k < \ell - 1$,

$$\mathbb{E}_{\boldsymbol{\Omega}} \|(\mathbf{I} - \boldsymbol{P}_{\boldsymbol{Y}})\boldsymbol{B}\|^2 \le \sigma_{k+1}^2 + \left(\sqrt{\frac{k}{\ell - k - 1}}\sigma_{k+1} + (\mathbb{E}\, W)^{1/2} \left(\sum\nolimits_{j>k} \sigma_j^2\right)^{1/2}\right)^2.$$

The expectation of the random variable W takes some effort to compute. The follow bound is always valid:

$$\mathbb{E}\, W \le \frac{\mathrm{e}\sqrt{\ell}}{\ell - k} \quad \text{for all } 2 \le k < \ell.$$

When $k \ll \ell$, this estimate can be somewhat wasteful. In this setting, the following heuristic is more accurate:

$$\mathbb{E}\, W \approx \frac{1}{\sqrt{\ell} - \sqrt{k}}, \quad \text{when } k \ll \ell.$$

Introducing these estimates into Theorem 10.10, we obtain detailed information about the performance of the randomized rangefinder procedure with respect to the spectral norm.

Lecture bibliography

[HMT11] N. Halko, P. G. Martinsson, and J. A. Tropp. "Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions". In: *SIAM Rev.* 53.2 (2011), pages 217–288. ISSN: 0036-1445. DOI: 10.1137/090771806.

11. Streaming SVD

Date: 11 February 2020 **Scribe:** Richard Kueng

Agenda:

1. Turnstile streaming model
2. Randomized linear sketches
3. Sketchy SVD
4. Implementation / Analysis
5. Examples

In this lecture, we are going to be interested in computing a truncated SVD approximately when we can only look at the matrix once, or the matrix is evolving. This is called a *streaming model* and occurs in a variety of applications. It may seem surprising that we can do this at all. But, we actually can approximately compute an SVD without ever visiting the matrix twice.

11.1 Turnstile streaming problem

Let $\boldsymbol{B} \in \mathbb{F}^{m \times n}$ be a matrix that is presented as a sequence of linear updates:

$$\boldsymbol{B} = \boldsymbol{H}_1 + \boldsymbol{H}_2 + \boldsymbol{H}_3 + \cdots \quad \text{where } \boldsymbol{H}_i \in \mathbb{F}^{m \times n}.$$

Typically, the innovations $\boldsymbol{H}_i$ are very simple objects. For example, they might be sparse or low-rank matrices. We think about the ambient dimensions m and n as very large, or enormous. Even if we see the entire matrix, we do not want to store a complete representation of $\boldsymbol{B}$ in memory. Algorithmically, this means that we must process each innovation and then discard it. This general model is called a termturnstile streaming model to differentiate it from other types of simpler streaming model.

Significant examples that fit into this model include

1. $\boldsymbol{H}_j = \boldsymbol{b}_j \mathbf{e}_j^*$: updates are individual matrix columns.
2. $\boldsymbol{H}_j = \mathbf{e}_i \boldsymbol{b}_j^*$: updates are individual matrix rows.
3. $\boldsymbol{H}_{ij} = h_{ij} \mathbf{E}_{ij}$: updates are individual matrix entries.

We can also consider cases where the columns/rows/entries are updated in an arbitrary order or where they are updated repeatedly.

This streaming formulation is often motivated in terms of *inventory models*, where the number of items in an inventory can go down (sales) or up (returns or replenishment). More recently, turnstile streaming has been used to model *one-pass data access models*:

inventory models

one-pass data access models

If a big matrix $\boldsymbol{B}$ is stored on tape drive, random or repeated access can be prohibitively expensive. Another prominent application is *scientific simulation*. The numerical solution of a PDE often proceeds iteratively, computing the next solution field from the previous ones. In many situations, each solution field is very large, and the computation is quite expensive.

scientific simulation

The goal for this lecture is to develop an approximate SVD algorithm that can operate in the turnstile streaming model. For a rank parameter k, given *a priori*, we want to compute an approximate k-truncated SVD of $\boldsymbol{B}$ at a given time T. We need to control the working storage, the time for individual updates, and the time for the SVD computation.

> **Warning 11.1** We are going to focus on the *single-shot case* where we only compute the approximate SVD once. If we need to compute the truncated SVD repeatedly, there are additional complications that arise because of failure probabilities and lack of independence among the approximations. ∎

single-shot case

11.2 Randomized linear sketching

Instead of storing the entire matrix $\boldsymbol{B}$, we are going to maintain a random linear image of $\boldsymbol{B}$. More precisely, let $\mathcal{S} : \mathbb{F}^{m\times n} \to \mathbb{F}^d$ be a linear map from matrices to vectors. We are going to track

$$\mathcal{S}(\boldsymbol{B}) = \mathcal{S}(\boldsymbol{H}_1) + \mathcal{S}(\boldsymbol{H}_2) + \mathcal{S}(\boldsymbol{H}_3) + \cdots$$

Linearity allows us to sketch each innovation and add it to the existing sketch.

The embedding dimension d is still a free parameter. If $d \ll mn$, then $\mathcal{S}$ must have a substantial nullspace. However, if we choose $\mathcal{S}$ at *random*, it is likely that the sketch works for any particular matrix $\boldsymbol{B}$ that is statistically independent from $\mathcal{S}$. After we see the whole data stream, we can use the sketch $\mathcal{S}(\boldsymbol{B})$ to compute the SVD. We get a guarantee that holds at a particular time T with high probability. Nevertheless, we cannot use the computed SVD to make decisions that affect the data stream without sullying the randomness in the sketch.

The idea of sketching is due to Noga Alon et al. [AMS99; Alo+02] who were interested in scalar quantities, like the energy in a data stream. The first SVD algorithm that can operate in the streaming setting was proposed by Wolfe et al. [Woo+08]. Later, Clarkson and Wodruff [CW09] developed the basic theory of sketching for numerical linear algebra computations. For the turnstile model, randomized linear sketches are essentially the only kind of algorithm; see [LNW14].

11.3 The SketchySVD algorithm

With randomized linear sketching at hand, the obvious question becomes: how do we design a streaming SVD algorithm? Let us start with the randomized SVD algorithm in its simplest form and see where it breaks.

11.3.1 Review of the randomized SVD

First, use the randomized rangefinder to find a basis $\boldsymbol{Q}$ that captures the range of $\boldsymbol{B}$:

$$\boldsymbol{B\Omega} = \boldsymbol{Y} = \boldsymbol{QR}$$

The matrix $\boldsymbol{\Omega} \in \mathbb{F}^{m\times\ell}$ is random. The second step is to compress $\boldsymbol{B}$ to the range of $\boldsymbol{Q}$. Then we perform a dense SVD computation:

$$\boldsymbol{C} = \boldsymbol{Q}^*\boldsymbol{B} = \hat{\boldsymbol{U}}\hat{\boldsymbol{\Sigma}}\hat{\boldsymbol{V}}^*.$$

Finally, we obtain a factored SVD:

$$\boldsymbol{B} = \left(\boldsymbol{Q}\hat{\boldsymbol{U}}\right)\hat{\boldsymbol{\Sigma}}\hat{\boldsymbol{V}}^*.$$

On the positive side, we collect linear data $\boldsymbol{Y} = \boldsymbol{B}\boldsymbol{\Omega}$. This already may be viewed as a (linear) sketch. On the negative side, we have to look at $\boldsymbol{B}$ again to perform the second step.

11.3.2 Avoiding the second pass

We can bypass this problem by sketching the matrix $\boldsymbol{B}$ on the left too. After all, the action of the matrix and its adjoint can be quite different. More precisely, we maintain

$$\boldsymbol{Y} = \boldsymbol{B}\boldsymbol{\Omega} \in \mathbb{F}^{m\times\ell} \quad \text{and} \quad \boldsymbol{X} = \boldsymbol{\Upsilon}\boldsymbol{B} \in \mathbb{F}^{\ell\times m}.$$

These are sketches for range and co-range, respectively. To compute an SVD, we first perform two QR decompositions:

$$\begin{aligned}\boldsymbol{Y} &= \boldsymbol{Q}\boldsymbol{R}_1 \quad \text{where } \boldsymbol{Q} \in \mathbb{F}^{m\times\ell} \text{ orthonormal;}\\ \boldsymbol{X}^* &= \boldsymbol{P}\boldsymbol{R}_2 \quad \text{where } \boldsymbol{P} \in \mathbb{F}^{n\times\ell} \text{ orthonormal.}\end{aligned}$$

The columns of $\boldsymbol{Q}$ capture the range of $\boldsymbol{B}$, while the columns of $\boldsymbol{P}$ capture the co-range. Heuristically,

$$\boldsymbol{B} \approx \boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{B}\boldsymbol{P}\boldsymbol{P}^* = \boldsymbol{Q}\left(\boldsymbol{Q}^*\boldsymbol{B}\boldsymbol{P}\right)\boldsymbol{P}^* =: \boldsymbol{Q}\boldsymbol{C}\boldsymbol{P}^* \quad \text{where} \quad \boldsymbol{C} = \boldsymbol{Q}^*\boldsymbol{C}\boldsymbol{P}.$$

Indeed, we have the intuition that the matrix $\boldsymbol{B}$ is well-approximated by its projection onto the range of $\boldsymbol{Q}$ on the left and its projection onto the range of $\boldsymbol{P}$ on the right. We have also defined the core matrix $\boldsymbol{C} = \boldsymbol{Q}^*\boldsymbol{C}\boldsymbol{P}^*$.

If we had access to $\boldsymbol{C}$ directly, we would be done. But this is not the case, because we can only query $\boldsymbol{B}$ once and we already used $\boldsymbol{B}$ to compute $\boldsymbol{Q}$ and $\boldsymbol{P}$. To bypass this problem, let us tabulate the information we already have:

$$\begin{aligned}\boldsymbol{Y} &= \boldsymbol{B}\boldsymbol{\Omega} \approx \boldsymbol{Q}\boldsymbol{C}\boldsymbol{P}^*\boldsymbol{\Omega},\\ \boldsymbol{X} &= \boldsymbol{\Upsilon}\boldsymbol{B} \approx \boldsymbol{\Upsilon}\boldsymbol{Q}\boldsymbol{C}\boldsymbol{P}^*.\end{aligned}$$

The matrices on the left-hand side are known, and the only unknown on the right-hand sides is the core matrix $\boldsymbol{C}$. Hence, we can use these formulas to find $\boldsymbol{C}$ by solving a least-squares problem.

This works reasonably well, as pointed out by Wolfe et al. [Woo+08] and Halko et al. [HMT11]. In fact, we can obtain significantly better performance if $\boldsymbol{\Omega}$ has ℓ columns while $\boldsymbol{\Upsilon}$ has, say, 2ℓ or 4ℓ rows [CW09; Tro+17a].

11.3.3 Adding a core sketch

Today, we are going to pursue a more elaborate algorithm that collects additional information to approximate the core matrix $\boldsymbol{C}$ more accurately. Draw and fix two additional random matrices

$$\boldsymbol{\Phi} \in \mathbb{F}^{s\times m} \quad \text{and} \quad \boldsymbol{\Psi} \in \mathbb{F}^{m\times s} \quad \text{where } s \geq \ell.$$

We use these additional matrices to maintain a $s \times s$ core sketch

$$\boldsymbol{Z} = \boldsymbol{\Phi}\boldsymbol{B}\boldsymbol{\Psi} \in \mathbb{F}^{s\times s}.$$

Heuristically, we expect

$$\boldsymbol{Z} \approx (\boldsymbol{\Phi}\boldsymbol{Q})\boldsymbol{C}(\boldsymbol{P}^*\boldsymbol{\Psi}).$$

We can use this relation to approximate the core matrix as follows.

$$\hat{\boldsymbol{C}} = (\boldsymbol{\Phi}\boldsymbol{Q})^\dagger \boldsymbol{Z} \, (\boldsymbol{P}^*\boldsymbol{\Psi})^\dagger$$

by solving a least-squares problem.

This approach has a crucial advantage. The sketch for the core is statistically independent from the sketches for the row and column spaces. If the test matrices are Gaussian, the approximated core matrix is even an unbiased estimator for the true core matrix. The idea of adding a core sketch is due to Upadhyay [Upa16], with subsequent improvements by Tropp et al. [Tro+17b].

11.3.4 The procedure

The *SketchySVD* procedure implements the following steps: *SketchySVD*

1. Draw and fix random matrices $\boldsymbol{\Omega} \in \mathbb{F}^{n\times\ell}$, $\boldsymbol{\Psi} \in \mathbb{F}^{n\times s}$, $\boldsymbol{\Upsilon} \in \mathbb{F}^{\ell\times m}$ and $\boldsymbol{\Phi} \in \mathbb{F}^{s\times m}$ with the only requirement $s \geq \ell$.
2. Maintain three sketches:

$$\begin{aligned} \boldsymbol{Y} &= \boldsymbol{B}\boldsymbol{\Omega} && \text{(range sketch)}, \\ \boldsymbol{X} &= \boldsymbol{\Upsilon}\boldsymbol{B} && \text{(co-range sketch)}, \\ \boldsymbol{Z} &= \boldsymbol{\Phi}\boldsymbol{B}\boldsymbol{\Psi} && \text{(core sketch)}. \end{aligned}$$

   These sketches are all linear, so they can be maintained in the turnstile streaming model.
3. Perform two QR factorizations: $\boldsymbol{Y} = \boldsymbol{Q}\boldsymbol{R}_1$ and $\boldsymbol{X}^* = \boldsymbol{P}\boldsymbol{R}_2$.
4. Compute a core approximation: $\hat{\boldsymbol{C}} = (\boldsymbol{\Phi}\boldsymbol{Q})^\dagger \boldsymbol{Z} \, (\boldsymbol{P}^*\boldsymbol{\Psi})^\dagger$.
5. Form an SVD of the core: $\hat{\boldsymbol{C}} = \hat{\boldsymbol{U}}\hat{\boldsymbol{\Sigma}}\hat{\boldsymbol{V}}^\dagger \in \mathbb{F}^{s\times s}$
6. Set $\hat{\boldsymbol{B}} = (\boldsymbol{Q}\hat{\boldsymbol{U}})\hat{\boldsymbol{\Sigma}}(\boldsymbol{P}\hat{\boldsymbol{V}})^*$
7. Truncate: $\hat{\boldsymbol{\Sigma}} \to [\![\hat{\boldsymbol{\Sigma}}]\!]_k$ with truncation rank k.

11.3.5 Storage, runtime and analysis

The storage cost of this procedure is $O(s^2 + \ell(m+n))$; one can use structured random matrices to nearly eliminate the cost of storing the sketching operators themselves. The arithmetic cost is $O(s^3 + \ell^2(m+n))$ for the SVD computation; this is dominated by the cost of the QR decompositions.

A serious drawback of such methods is that we can only query $\boldsymbol{B}$ once. If we make a mistake, we cannot hope to rectify it by revisiting $\boldsymbol{B}$. The implications of this limitation are two-fold:

1. We need good parameter choices (s, ℓ). This requires serious theory.
2. *A posteriori* validation is important.

We will quickly discuss these points in the following subsections.

11.4 *A priori* analysis for parameter choices

The following rigorous theorem is due to Tropp *et al.* [Tro+17b], and it provides guidance for how to choose the sketching parameters s and ℓ.

Theorem 11.2 (Tropp et al. 2019). Assume $\mathbb{F} = \mathbb{C}$. If the test matrices are iid standard normal and $s \geq 2\ell$, then

$$\mathbb{E}\,\|\boldsymbol{B} - \boldsymbol{Q}\hat{\boldsymbol{C}}\boldsymbol{P}^*\|_{\mathrm{F}}^2 \leq \frac{s}{s-\ell} \min_{k<\ell} \frac{\ell+k}{\ell-k} \left(\sum\nolimits_{j>k} \sigma_j^2(\boldsymbol{B}) \right). \tag{11.1}$$

Although this bound seems complicated, it can provide good advice on choosing the sketching parameters. To use it, we need to have an estimate for the target rank k, and we require some knowledge about the spectral decay of $\boldsymbol{B}$. In many cases, the choices $\ell = 4k$ and $s = 2\ell$ are effective.

Warning 11.3 Unusually for us, the bound (11.1) reports an estimate for the Frobenius-norm error. It turns out to be impossible to get relative-error spectral-norm error bounds for turnstile streaming algorithms. ∎

11.5 *A posteriori* error validation

To validate the quality of the computed SVD, we can maintain another sketch of $\boldsymbol{B}$ to evaluate the approximation error. To do so, we draw and fix another Gaussian matrix $\boldsymbol{\Theta} \in \mathbb{F}^{n \times q}$. We also form the sketch

$$\boldsymbol{W} = \boldsymbol{B}\boldsymbol{\Theta}.$$

Given any approximation $\hat{\boldsymbol{B}}$ that is statistically independent from $\boldsymbol{\Theta}$, the Frobenius-norm error in the approximation obeys

$$\|\boldsymbol{B} - \hat{\boldsymbol{B}}\|_{\mathrm{F}}^2 = \mathbb{E}\left[\frac{1}{q} \|\boldsymbol{W} - \boldsymbol{B}\hat{\boldsymbol{\Theta}}\|_F^2 \right].$$

This result is drawn from our discussion about randomized trace estimation. Perhaps surprisingly, an extremely small sketch suffices for this validation step. The choice $q = 5$ is typically good enough.

We can also use the empirical estimate of the error to try to determine a suitable value for truncation rank k.

Lecture bibliography

[AMS99] N. Alon, Y. Matias, and M. Szegedy. "The space complexity of approximating the frequency moments". In: volume 58. 1, part 2. Twenty-eighth Annual ACM Symposium on the Theory of Computing (Philadelphia, PA, 1996). 1999, pages 137–147. DOI: 10.1006/jcss.1997.1545.

[Alo+02] N. Alon et al. "Tracking join and self-join sizes in limited storage". In: volume 64. 3. Special issue on PODS 1999 (Philadelphia, PA). 2002, pages 719–747. DOI: 10.1006/jcss.2001.1813.

[CW09] K. L. Clarkson and D. P. Woodruff. "Numerical Linear Algebra in the Streaming Model". In: *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*. STOC '09. Bethesda, MD, USA: Association for Computing Machinery, 2009, pages 205–214. DOI: 10.1145/1536414.1536445.

[HMT11] N. Halko, P. G. Martinsson, and J. A. Tropp. "Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions". In: *SIAM Rev.* 53.2 (2011), pages 217–288. ISSN: 0036-1445. DOI: 10.1137/090771806.

[LNW14] Y. Li, H. L. Nguyen, and D. P. Woodruff. "Turnstile streaming algorithms might as well be linear sketches". In: *STOC'14—Proceedings of the 2014 ACM Symposium on Theory of Computing*. ACM, New York, 2014, pages 174–183.

[Tro+17a] J. A. Tropp et al. "Fixed-Rank Approximation of a Positive-Semidefinite Matrix from Streaming Data". In: *Advances in Neural Information Processing Systems 30*. Edited by I. Guyon et al. Curran Associates, Inc., 2017, pages 1225–1234. URL: http://papers.nips.cc/paper/6722-fixed-rank-approximation-of-a-positive-semidefinite-matrix-from-streaming-data.pdf.

[Tro+17b] J. A. Tropp et al. "Practical sketching algorithms for low-rank matrix approximation". In: *SIAM J. Matrix Anal. Appl.* 38.4 (2017), pages 1454–1485. DOI: 10.1137/17M1111590.

[Upa16] J. Upadhyay. "Fast and space-optimal low-rank factorization in the streaming model with application in differential privacy". In: *arXiv preprint arXiv:1604.01429* (2016).

[Woo+08] F. Woolfe et al. "A fast randomized algorithm for the approximation of matrices". In: *Appl. Comput. Harmon. Anal.* 25.3 (2008), pages 335–366. DOI: 10.1016/j.acha.2007.12.002.

12. Finding Natural Bases

Date: 13 February 2020 **Scribe:** Chung-Yi Lin

This lecture explores how to find a low-rank approximation that contains a subset of columns or rows of the original matrix. This type of factorization admits a more natural interpretation, so it can be valuable for practical data analysis. To that end, we will first develop deterministic methods to compute row and column-based matrix factorizations. We will then introduce randomness to make the algorithms more efficient and scalable.

Agenda:

1. Natural bases
2. Row/column factorization
3. Row/column approximation
4. CPQR
5. Randomized ID

12.1 Motivation: Natural bases versus eigenbases

To begin, let us introduce the concept of a factor model. Then we discuss the relative merits of factor models constructed from the SVD, versus models where the factors are data points.

12.1.1 Factor models

The truncated singular value decomposition (SVD) and its randomized variants (discussed in Lecture 9) find many applications in exploratory data analysis. In particular, principal component analysis (PCA) applies the truncated SVD to a standardized data covariance matrix. The computed singular values and singular vectors give information are interpreted as latent structures that appear in the data. Applications of such *factor models* include identifying "meta-genomes" (e.g., regional, environmental attributes) from genetic marker data in biology, classifying documents into different "topics" according to the term frequency (e.g., the TF-IDF measure) in natural language processing, detecting "communities" in a social media platform based on the connection (e.g., friendship, common sharing of a post) among the users, etc. We refer the reader to [MD09] for more data analysis applications.

factor models

In a factor model, we want to obtain a low-rank approximation

$$\underset{m\times n}{\boldsymbol{B}} \approx \underset{m\times k}{\boldsymbol{F}}\ \underset{k\times n}{\boldsymbol{W}} \tag{12.1}$$

at some target rank $k \ll \min\{m, n\}$, or equivalently

$$\boldsymbol{b}_j \approx \sum_{i=1}^{m} w_{ij} \boldsymbol{f}_i \quad j = 1, \ldots, n \tag{12.2}$$

for each column of $\boldsymbol{B}$. In the expression (12.2), the columns $\boldsymbol{f}_i$ of $\boldsymbol{F}$ are called *factors*, which are viewed as latent variables that can summarize the data. The entries w_{ij} of $\boldsymbol{W}$ are called *weights / loadings*, and they specify how the factors are combined together to represent each column $\boldsymbol{b}_j$ of the original data matrix.

factors

weights / loadings

12.1.2 Reification and structural issues with the SVD

Unfortunately, while the SVD produces a rank-k matrix factorization (12.1) that has minimal error, it can be challenging to interpret the obtained factors in terms of the data or the underlying data-generating mechanism. More seriously, the computed factors may not correspond with any object that actually exists in the world. The latter issue is called *reification*, and it is one major reason that data analysis can lead to fallacious or inappropriate conclusions.

reification

For example, a movie recommendation system might construct a factor that describes a category of movies:

$$\begin{bmatrix} (1/2) & \text{comedy content} \\ (-1/\sqrt{2}) & \text{action content} \\ (1/2) & \text{a blockbluster} \end{bmatrix}$$

In this case, we can view the positive values as correlations and negative values as *anti*-correlations. But there may be no such thing as a funny-placid-blockbuster movie.

The same approach can be deeply problematic for other kinds of data, like genomic data or text data. Indeed, we can formulate the concept of a "meta-genome" or a "topic" as being a linear combination of a given set of data points, but it is questionable to try to interpret a negative SNP in genetic data or a negative word count in text data. As quoted in [KPS01], "While very efficient basis vectors, the (singular) vectors themselves are completely artificial and do not correspond to actual [DNA expression] profiles. ... Thus, it would be interesting to try to find basis vectors for all experiment vectors, using actual experiment vectors and not artificial bases that offer little insight."

Apart from the reification issue, there are also *computational challenges* that can arise because factor models may not preserve or exploit matrix structure. For instance, given a sparse matrix $\boldsymbol{B}$ in (12.1), the factored matrices $\boldsymbol{F}$ and $\boldsymbol{W}$ may no longer be sparse.

computational challenges

To improve the interpretability and computational profile of a matrix decomposition, we will study factor models that respect the coordinate structure. The idea is to build low-rank matrix approximations where the factors are columns of the original matrix. In other words, we factorize a given dataset by identifying *exemplars*, or "typical" members of the population. This corresponds to finding individuals that can serve as representatives for larger populations (in genetic data), or to finding documents that epitomize specific topics (in text data).

exemplars

12.2 Row/column-based factorizations

We proceed to a more formal development. This section describes several types of matrix factorizations induced by rows or columns. For simplicity, we focus on the case where the input matrix has low rank. Afterward, we turn to questions about approximating a matrix that may not have low-rank, and we discuss computational issues.

12.2.1 Interpolative decompositions

We start with a column-based factorization when the rank of the original matrix is known *a priori*.

Definition 12.1 (Column ID). Let $\boldsymbol{B} \in \mathbb{F}^{m\times n}$ be a fixed matrix with exact rank k. A *column interpolative decomposition (ID)* is a factorization of the form

column interpolative decomposition (ID)

$$\underset{m\times n}{\boldsymbol{B}} = \underset{m\times k}{\boldsymbol{C}}\ \underset{k\times n}{\boldsymbol{Z}}, \tag{12.3}$$

where the matrix $\boldsymbol{C}$ comprises a subset of the columns of $\boldsymbol{B}$ and the matrix $\boldsymbol{Z}$ contains the $k \times k$ identity matrix. In other words,

$$\boldsymbol{C} = \boldsymbol{B}(:, J_s) \quad \text{and} \quad \boldsymbol{Z}(J_s, J_s) = \mathbf{I}_k$$

where $J_s \subset \{1, \dots, n\}$ indexes the columns and $|J_s| = k$.

In the decomposition (12.3), those columns of $\boldsymbol{Z}$ that are outside J_s specify how to represent the remaining columns of $\boldsymbol{B}$ as linear combinations of the distinguished columns, indexed by J_s. The fact that such a decomposition exists is a direct consequence of the definition of rank. Moreover, we can ensure that the decomposition (12.3) is *well-conditioned*, in the sense that entries of the coefficient matrix $\boldsymbol{Z}$ are bounded.

Parallel to the column ID, we can also define a factorization that uses a subset of rows of the original matrix $\boldsymbol{B}$ to span its co-range.

Definition 12.2 (Row ID). Let $\boldsymbol{B} \in \mathbb{F}^{m\times n}$ be a fixed matrix with exact rank k. A *row interpolative decomposition* is a factorization of the form

row interpolative decomposition

$$\underset{m\times n}{\boldsymbol{B}} = \underset{m\times k}{\boldsymbol{X}}\ \underset{k\times n}{\boldsymbol{R}}, \tag{12.4}$$

where

$$\boldsymbol{R} = \boldsymbol{B}(I_s, :) \quad \text{and} \quad \boldsymbol{X}(I_s, I_s) = \mathbf{I}_k$$

for some row-index set $I_s \subset \{1, \dots, m\}$ with $|I_s| = k$.

We can even consider a two-sided factorization that extracts both rows and columns of $\boldsymbol{B}$. Such factorizations, however, are less interpretable.

Definition 12.3 (Two-sided ID). Let $\boldsymbol{B} \in \mathbb{F}^{m\times n}$ be a fixed matrix with exact rank k. A *two-sided interpolative decomposition* is a factorization of the form

two-sided interpolative decomposition

$$\underset{m\times n}{\boldsymbol{B}} = \underset{m\times k}{\boldsymbol{X}}\ \underset{k\times k}{\boldsymbol{B}_s}\ \underset{k\times n}{\boldsymbol{Z}}, \tag{12.5}$$

where

$$\boldsymbol{B}_s = \boldsymbol{B}(I_s, J_s).$$

Here, the matrix $\boldsymbol{Z}$ and the index set J_s are the same as in Definition 12.1, while $\boldsymbol{X}$ and I_s are the same as in Definition 12.2.

Remark 12.4 (Conditioning and Computation). There exists a factorization (12.3) for which no entry of $\boldsymbol{Z}$ has magnitude larger than one. We refer the reader to [Pan00] for a constructive proof, but [CM109] points out that it is NP-hard to compute this kind of factorization. Nevertheless, if we relax the requirement and search for a factor $\boldsymbol{Z}$ whose entries have magnitude no larger than two, there are stable and efficient algorithms for computing the ID. See [GE96; Che+05].

12.2.2 CUR decomposition

The final decomposition that we introduce in this lecture has a similar flavor to the two-sided ID.

Definition 12.5 (CUR decomposition). Let $\boldsymbol{B} \in \mathbb{F}^{m\times n}$ be a fixed matrix with exact rank k. A *CUR decomposition* is a factorization

CUR decomposition

$$\underset{m\times n}{\boldsymbol{B}} = \underset{m\times k}{\boldsymbol{C}}\ \underset{k\times k}{\boldsymbol{U}}\ \underset{k\times n}{\boldsymbol{R}} \tag{12.6}$$

where $\boldsymbol{C}$ and $\boldsymbol{R}$ are the matrices that appear in (12.3) and (12.4).

Here, $\boldsymbol{U}$ is a small *core / linkage matrix* that ties the matrices $\boldsymbol{C}$ and $\boldsymbol{R}$ together. In this subsection, where $\boldsymbol{B}$ is assumed to have exact rank k, we can translate between the two-sided ID (12.5) and the CUR decomposition (12.6) through the relation

core / linkage matrix

$$\boldsymbol{B} = \boldsymbol{X}\boldsymbol{B}_s\boldsymbol{Z} = \underbrace{(\boldsymbol{X}\boldsymbol{B}_s)}_{\boldsymbol{C}}\ \underbrace{\boldsymbol{B}_s^{-1}}_{\boldsymbol{U}}\ \underbrace{(\boldsymbol{B}_s\boldsymbol{Z})}_{\boldsymbol{R}}. \tag{12.7}$$

Remark 12.6 (Naming history). These types of approximations date back at least as far as the paper [GZT97], which uses the term *skeleton* decompositions to refer to interpolative decompositions. The same paper also discusses CUR decompositions, which are referred to as *pseudo-skeleton* decomposionts. The subscript s on the index vectors I_s and J_s stands for *skeleton*, in recognition of this history.

skeleton

pseudo-skeleton

12.2.3 Comparisons: CUR vs. two-sided ID

The CUR decomposition (12.6) has an advantage that it requires less storage than the two-sided ID (12.5). In addition to the linkage matrix $\boldsymbol{U}$, storing the row index I_s and the column index J_s suffices to reconstruct the matrices $\boldsymbol{R}$ and $\boldsymbol{C}$ in a CUR decomposition, provided that the original matrix $\boldsymbol{B}$ is directly accessible. When it is not, we can still exploit structure of $\boldsymbol{B}$ (e.g., sparsity) to achieve efficient storage. In contrast, the coefficient matrices $\boldsymbol{Z}$ and $\boldsymbol{X}$ in a two-sided ID does not preserve the matrix structure as the sub-matrices $\boldsymbol{R}$ and $\boldsymbol{C}$ do.

A disadvantage of the CUR decomposition is that the matrix $\boldsymbol{U}$ is ill-conditioned when the original matrix $\boldsymbol{B}$ has a fast-decaying spectrum. In such cases, the singular values of the $k \times k$ sub-matrix $\boldsymbol{B}_s$ often approximate the k dominant singular values of $\boldsymbol{B}$. Hence, the relationship $\boldsymbol{U} = \boldsymbol{B}_s^{-1}$ in (12.7) implies that $\boldsymbol{U}$ will have elements in the order of $1/\sigma_k(\boldsymbol{B}_s)$. On the other hand, the two-sided ID tends to be more stable and numerically benign since we can find an interpolative decomposition where the coefficient matrices $\boldsymbol{Z}$ and $\boldsymbol{X}$ have entries no larger than one in modulus, as we point out in Remark 12.4.

12.3 Row/column-based approximations

In a real-world application, we typically do not know the precise rank of a data matrix $\boldsymbol{B}$ in advance. Nevertheless, the singular values of $\boldsymbol{B}$ usually decay fast enough so that it is reasonable to form a low-rank approximation. This section describes row/column-based matrix approximations that are applicable in this setting.

12.3.1 Low-rank approximations

Let $\boldsymbol{B} \in \mathbb{F}^{m\times n}$ be a fixed tall (i.e., $m \geq n$) matrix. Recall a fact from Lecture 9: The truncated k-SVD $[\![\boldsymbol{B}]\!]_k$ gives the best rank-k approximation with respect to the spectral

norm in the sense that

$$\min_{\operatorname{rank}(\hat{\boldsymbol{B}})=k} \|\boldsymbol{B}-\hat{\boldsymbol{B}}\| = \|\boldsymbol{B}-[\![\boldsymbol{B}]\!]_k\| = \sigma_{k+1}(\boldsymbol{B}), \tag{12.8}$$

which is attributed to Mirsky [Mir60].

12.3.2 Approximation error of IDs

The situation is somewhat different for row/column-based approximations. Let $k < \min\{m, n\}$ be given, and consider a k-column ID approximation:

$$\underset{m\times n}{\boldsymbol{B}} \approx \underset{m\times k}{\boldsymbol{C}}\ \underset{k\times n}{\boldsymbol{Z}} \tag{12.9}$$

It is known that

$$\|\boldsymbol{B}-\boldsymbol{C}\boldsymbol{Z}\| \le \sqrt{k(n-k)+1}\cdot\sigma_{k+1}(\boldsymbol{B}) \tag{12.10}$$

in the worst case [Lib+07]. We can see from (12.10) that column-based approximations (12.9) pay an extra multiplicative factor in the order of $\sqrt{k(n-k)}$, which may be much worse than the k-truncated SVD.

If $\boldsymbol{B}$ exhibits fast (e.g., exponential) spectral decay, then the approximation error (12.10) attained by a column ID is roughly the same as the optimal one (12.8) since $\sigma_{k+1}(\boldsymbol{B})$ is *tiny*. When the singular values of $\boldsymbol{B}$ decay slowly, then the number of columns that we need to achieve an error close to the optimal value $\sigma_{k+1}(\boldsymbol{B})$ can be much larger than k. Similar problems arise for row ID approximations of the form (12.4) and for two-sided-ID approximations of the form (12.5).

12.3.3 CUR approximations

For a CUR approximation of the form (12.6) when $\operatorname{rank}(\boldsymbol{B})$ is not known *a priori*, it is common practice to identify the representative index sets (I_s, J_s) first and then find a proper linkage matrix $\boldsymbol{U}$. We will leave the first problem to the next section and discuss the second problem of finding $\boldsymbol{U}$ here.

In principle, we can use the relation $\boldsymbol{U} = \boldsymbol{B}_s^{-1}$ once the indices are determined. This is not a good idea in practice, however, because the matrix $\boldsymbol{B}_s$ may not even be invertible. One reason is that the common practice of oversampling can easily make $\boldsymbol{B}_s$ singular or ill-conditioned. Therefore, a better construction of $\boldsymbol{U}$ is to solve a least-squares problem based on (12.6):

$$\boldsymbol{U} = \boldsymbol{C}^{\dagger}\boldsymbol{B}\boldsymbol{R}^{\dagger}. \tag{12.11}$$

We refer the reader to [VM17] for an error analysis of the CUR approximations.

12.4 CPQR: A deterministic ID algorithm

We continue with a discussion of classical deterministic algorithms for computing interpolative decompositions.

12.4.1 Classical solution

Now, let us return to the question of how to find a set of columns that span the range of a given matrix $\boldsymbol{B}$. As we discussed after Definition 12.1, a column ID exists by the very definition of rank. Hence, a natural way to compute it is through a *column-pivoted QR (CPQR)* decomposition. For instance, we can use the (double) Gram–Schmidt process with greedy pivoting, which always chooses to orthogonalize the column of

column-pivoted QR (CPQR)

$\boldsymbol{B}$ that has the largest Euclidean norm. After k iterations, the CPQR returns a partial decomposition of the matrix $\boldsymbol{B}$:

$$\underset{m\times n}{\boldsymbol{B}}\ \underset{n\times n}{\boldsymbol{\Pi}} = \underset{m\times k}{\boldsymbol{Q}}\ \underset{k\times n}{\boldsymbol{T}} + \underset{m\times n}{\boldsymbol{E}}\,. \tag{12.12}$$

Here, the columns of $\boldsymbol{Q}$ forms an orthonormal basis for the space spanned by the k selected columns of $\boldsymbol{B}$, and $\boldsymbol{T}$ is an upper triangular matrix that specifies how these k columns of $\boldsymbol{Q}$ are related to the original k columns of $\boldsymbol{B}$. Moreover, the matrix $\boldsymbol{\Pi}$ is a permutation matrix that ensures the k columns picked are the first k columns of $\boldsymbol{B}\boldsymbol{\Pi}$, and $\boldsymbol{E}$ is a residual matrix that holds the information on those $n-k$ columns that are yet to be processed.

12.4.2 Conversion from CPQR to ID

To see how CPQR gives a column ID algorithm, we split the matrix $\boldsymbol{T}$ in (12.12) into panels:

$$\underset{k\times n}{\boldsymbol{T}} = \begin{bmatrix} \underset{k\times k}{\boldsymbol{T}_1} & \underset{k\times(n-k)}{\boldsymbol{T}_2} \end{bmatrix}$$

where $\boldsymbol{T}_1$ is upper-triangular and corresponds to the most important k columns of $\boldsymbol{B}$. Multiplying both sides of (12.12) by the inverse permutation matrix $\boldsymbol{\Pi}^*$ on the right and factoring $\boldsymbol{T}_1$ out front, we have

$$\boldsymbol{B} = \underbrace{\boldsymbol{Q}\boldsymbol{T}_1}_{\boldsymbol{C}}\ \underbrace{[\,\mathbf{I}_k \quad \boldsymbol{T}_1^{-1}\boldsymbol{T}_2]\boldsymbol{\Pi}^*}_{\boldsymbol{Z}} + \boldsymbol{E}\boldsymbol{\Pi}^* \tag{12.13}$$

where the term $\boldsymbol{E}\boldsymbol{\Pi}^*$ is the error in approximating $\boldsymbol{B}$.

A row ID (12.4) can be obtained similarly by applying the Gram–Schmidt procedure to the rows of $\boldsymbol{B}$, or by applying the CPQR to the adjoint $\boldsymbol{B}^*$. To get a two-sided ID (12.5), we can perform another row ID computation on the matrix $\boldsymbol{Z}$ obtained by the column ID algorithm (12.13). Finally, to get a CUR decomposition (12.6), we can use (12.11) to compute the linkage matrix $\boldsymbol{U}$.

12.4.3 Complexity and quality

The CPQR algorithm that computes a k-column approximate ID of an $m \times n$ matrix $\boldsymbol{B}$ has the following time complexity: $O(k^2 m)$ for the QR decomposition and $O(mn)$ for pivoting at each iteration, which leads to a total cost of $O(k^2 m + mnk)$. For large matrices (i.e., large m or n), this could be very expensive because of the pivoting steps.

We should also mention that CPQR can fail. Indeed, Kahan constructed a famous example where CPQR performs very poorly [Kah66]. A completely reliable counterpart to CPQR is the *rank-revealing QR* (RRQR) decomposition algorithm [GE96], which is guaranteed to produce near-optimal results at a slightly higher complexity. Practically speaking, CPQR works well. The major practical issue is that CPQR requires random access to $\boldsymbol{B}$ because the pivot choices cannot be computed in advance.

rank-revealing QR

12.5 Randomized interpolative decompositions

We are now prepared to discuss randomized algorithms for computing interpolative decompositions more efficiently.

12.5.1 Row/column IDs for large matrices

As we have discussed at the end of the previous section, the CPQR algorithm is not so efficient for large data sets. In this section, we describe how to use randomized methods to improve the computational profile of an ID algorithm.

In particular, consider the task of constructing a column ID of an $m \times n$ matrix $\boldsymbol{B}$, and assume that $\operatorname{rank}(\boldsymbol{B}) = k$ for simplicity. The idea is to first build a smaller matrix $\boldsymbol{Y}$ that spans the co-range of $\boldsymbol{B}$ well. Then, we find the representative column index set J_s of $\boldsymbol{Y}$ by applying CPQR to this smaller matrix. Then we argue that the resulting column ID of $\boldsymbol{Y}$ is a column ID of the original matrix $\boldsymbol{B}$.

First, let us examine how the computed ID of $\boldsymbol{Y}$ leads to an ID of $\boldsymbol{B}$. Suppose we have determined an $k \times n$ matrix $\boldsymbol{Y}$ whose rows span the co-range of $\boldsymbol{B}$. We obtain a factorization

$$\underset{m\times n}{\boldsymbol{B}} = \underset{m\times k}{\boldsymbol{F}} \; \underset{k\times n}{\boldsymbol{Y}} \tag{12.14}$$

for some coefficient matrix $\boldsymbol{F}$. We then proceed to compute a column ID of Y using CPQR:

$$\underset{k\times n}{\boldsymbol{Y}} = \underset{k\times k}{\boldsymbol{Y}(:, J_s)} \; \underset{k\times n}{\boldsymbol{Z}} \, . \tag{12.15}$$

The following fact claims that, to compute a column ID of a matrix $\boldsymbol{B}$, it suffices to compute a column ID of a smaller matrix $\boldsymbol{Y}$ whose columns fully spans range($\boldsymbol{B}$).

Proposition 12.7 (Transfer principle for ID). The pair $\{J_s, \boldsymbol{Z}\}$ in (12.15) determines a column ID of the original matrix $\boldsymbol{B}$.

Proof. Observe that

$$\begin{aligned} \boldsymbol{B}(:, J_s)\boldsymbol{Z} &= \boldsymbol{F}\boldsymbol{Y}(:, J_s)\boldsymbol{Z} && \text{by (12.14) restricted to columns in } J_s \\ &= \boldsymbol{F}\boldsymbol{Y} && \text{by (12.15)} \\ &= \boldsymbol{B}. \end{aligned}$$

This is the required statement. ∎

Exercise 12.8 (Drop the exact-rank assumption). Prove a similar claim to Fact 12.7 when $\boldsymbol{B}$ is not rank-deficient.

12.5.2 Old friends who well span the range

Finally, we return to the problem of finding a matrix $\boldsymbol{Y}$ that captures the co-range of $\boldsymbol{B}$. Recall from Lecture 9 and Lecture 10 that the randomized rangefinder (RRF) gives us enough information to choose significant rows/columns of the original matrix $\boldsymbol{B}$, which serves our purpose here. Combining the RRF with the column ID algorithm that we discussed in the previous subsection, we arrive at the following randomized ID algorithm.

Algorithm 12.1 has a complexity of $O(kmn + k^2 n)$, due to the matrix–matrix multiplications and the column ID steps. For practical implementations, we can take the oversampling parameter p to be a small constant (e.g., $p = 5$). Moreover, we can use the structured random embeddings introduced in Lecture 7 to obtain a random embedding with $O(mn \log k)$ complexity.

Remark 12.9 (Coordinate sampling). We refer the reader to Section 13.5 in the reference survey [MT20] for a discussion of techniques based on random coordinate sampling. Although it may seem appealing, we recommend against using coordinate sampling to compute IDs. The methods described here are significantly more effective in practice.

Algorithm 12.1 Randomized ID.

Input: Matrix $\boldsymbol{B} \in \mathbb{F}^{m\times n}$, target rank k, and oversampling parameter p
Output: A column ID $[J_s : \boldsymbol{Z}]$ of the matrix $\boldsymbol{B}$ such that $\boldsymbol{B} = \boldsymbol{B}(:, J_s)\boldsymbol{Z}$

1 **function** RandomizedID($\boldsymbol{B}$; k; p)
2 Draw a random (e.g., Gaussian) test matrix $\boldsymbol{S} \in \mathbb{F}^{(k+p)\times n}$
3 Form the sample matrix $\boldsymbol{Y} = \boldsymbol{S}\boldsymbol{B} \in \mathbb{F}^{(k+p)\times n}$ ▷ may use Power/Krylov method
4 Compute a column ID $[J_s : \boldsymbol{Z}]$ of $\boldsymbol{Y}$ with CPQR ▷ may use RRQR to improve accuracy

Lecture bibliography

[Che+05] H. Cheng et al. "On the Compression of Low Rank Matrices". In: *SIAM Journal on Scientific Computing* 26.4 (2005), pages 1389–1404.

[CMI09] A. Civril and M. Magdon-Ismail. "On selecting a maximum volume sub-matrix of a matrix and related problems". In: *Theoretical Computer Science* 410.47 (2009), pages 4801 –4811.

[GZT97] S. A. Goreinov, N. Zamarashkin, and E. E. Tyrtyshnikov. "Pseudo-skeleton approximations by matrices of maximal volume". In: *Mathematical Notes* 62 (1997), pages 515–519.

[GE96] M. Gu and S. C. Eisenstat. "Efficient Algorithms for Computing a Strong Rank-Revealing QR Factorization". In: *SIAM Journal on Scientific Computing* 17.4 (1996), pages 848–869.

[Kah66] W. Kahan. "Numerical Linear Algebra". In: *Canadian Mathematical Bulletin* 9.5 (1966), pages 757–801.

[KPS01] F. G. Kuruvilla, P. J. Park, and S. L. Schreiber. "Vector algebra in the analysis of genome-wide expression data". In: *Genome Biology* 3 (2001), research0011.1 –research0011.11.

[Lib+07] E. Liberty et al. "Randomized algorithms for the low-rank approximation of matrices". In: *Proceedings of the National Academy of Sciences* 104.51 (2007), pages 20167–20172.

[MD09] M. W. Mahoney and P. Drineas. "CUR matrix decompositions for improved data analysis". In: *Proceedings of the National Academy of Sciences* 106.3 (2009), pages 697–702.

[MT20] P.-G. Martinsson and J. Tropp. "Randomized Numerical Linear Algebra: Foundations & Algorithms". In: *Acta Numerica, 2020*. Cambridge Univ. Press, 2020, pages 403–572. arXiv: 2002.01387.

[Mir60] L. Mirsky. "Symmetric guage functions and unitarily invariant norms". In: *The Quarterly Journal of Mathematics* 11.1 (Jan. 1960), pages 50–59.

[Pan00] C.-T. Pan. "On the existence and computation of rank-revealing LU factorizations". In: *Linear Algebra and its Applications* 316.1 (2000). Special Issue: Conference celebrating the 60th birthday of Robert J. Plemmons, pages 199 –222.

[VM17] S. Voronin and P.-G. Martinsson. "Efficient Algorithms for CUR and Interpolative Matrix Decompositions". In: *Adv. Comput. Math.* 43.3 (June 2017), pages 495–516.

13. Kernel Methods

Date: 18 February 2020 **Scribe:** Po-Chih Chen

This lecture begins our discussion of kernel methods, which appear in a wide range of modern machine learning algorithms. First, we introduce the notion of positive-definite kernels and their interpretation as (nonlinear) features. We give several canonical examples of positive-definite kernels. We outline the method of kernel principal component analysis (KPCA). Finally, we mention some computational issues that arise from kernel methods. The idea of using randomized linear algebra to implement kernel methods more efficiently will be the main topic of the next few lectures.

Agenda:

1. Positive-definite kernels
2. Feature space
3. Examples
4. Kernel PCA
5. Computation

13.1 Positive-definite kernels

Let $\mathcal{X}$ be a set, called the input space or the data space. Suppose we acquire observations $\{x_1, \dots, x_n\} \subset \mathcal{X}$. Our goal is to use this data for learning tasks.

To this end, we introduce a *kernel function* $k : \mathcal{X} \times \mathcal{X} \to \mathbb{F}$. The value $k(x, y)$ is a measure of similarity between observations x and y. The $n \times n$ *kernel matrix* $\boldsymbol{K}$ tabulates the similarity for each pair of observations:

kernel function
kernel matrix

$$(\boldsymbol{K})_{ij} = k(x_i, x_j) \quad \text{for } i, j = 1, \dots, n.$$

The kernel matrix is an analog of the Gram matrix of data points in $(\mathbb{F}^d, \ell_2)$.

Definition 13.1 (Positive-definite kernel). A kernel function k is *positive definite (pd)* if, for each $n \in \mathbb{N}$ and each $\{x_1, \dots, x_n\} \subset \mathcal{X}$,

positive definite (pd)

$$\boldsymbol{K} = [k(x_i, x_j)]_{i,j=1,\dots,n} \in \mathbb{H}_n$$

is a positive-semidefinite matrix. The kernel k is said to be *strictly positive definite* if $\boldsymbol{K}$ is positive definite.

strictly positive definite

A positive-definite kernel necessarily has several properties:

- $k(x, x) \geq 0$ for all $x \in \mathcal{X}$.

- $k(x, y) = k(y, x)^*$ for all $x, y \in \mathcal{X}$.

The properties of the associated kernel matrix mirror the properties of the Gram matrix of Euclidean data.

13.1.1 Feature space

A kernel function can be interpreted as reporting the inner product between features derived from the data. Let $\mathcal{F}$ be a Hilbert space, which is called the *feature space*. Consider a function from data into the feature space $\Phi : \mathcal{X} \to \mathcal{F}$, which is called the *feature map*. Heuristically, Φ extracts numerical information from a data point that is relevant for leaning goals (although the features are usually not directly interpretable).

feature space

feature map

Under mild conditions, the feature map induces a pd kernel:

$$k(x, y) = \langle \Phi(x),\ \Phi(y) \rangle_{\mathcal{F}} \quad \text{for all } x, y \in \mathcal{X}.$$

Conversely, a pd kernel k always induces a feature map from $\mathcal{X}$ into an appropriate feature space $\mathcal{F}$.

> **Idea:** Kernels get more information out of the data. We can select, tune, or design the kernel function.

13.1.2 Examples

Let us introduce some of the most important positive-definite kernels.

Example 13.2 (Inner product kernel). Let $\mathcal{X} = \mathbb{F}^d$. Then, the *inner product kernel*

inner product kernel

$$k(x, y) = \langle x,\ y \rangle_{\mathbb{F}^d} \quad \text{for } x, y \in \mathcal{X}$$

is a pd kernel. ▪

Proof. Positive semidefiniteness follows from the fact that the kernel matrix $\boldsymbol{K}$ is the ordinary Gram matrix. ■

Example 13.3 (Angular similarity kernel). Let $\mathcal{X} = \mathbb{S}^{d-1}(\mathbb{R})$. Then, the *angular similarity kernel*

angular similarity kernel

$$k(x, y) = \frac{2}{\pi} \arcsin \langle x,\ y \rangle = 1 - \frac{2}{\pi} \theta(x, y) \quad \text{for } x, y \in \mathcal{X}$$

is a pd kernel. ▪

Proof. Let $\boldsymbol{G} = \boldsymbol{X}^* \boldsymbol{X}$ be the Gram matrix of $\boldsymbol{X} = [\boldsymbol{x}_1 \cdots \boldsymbol{x}_n]$. Then

$$\boldsymbol{K} = \frac{2}{\pi} \arcsin[\boldsymbol{G}] = \frac{2}{\pi} \sum\nolimits_{p=0}^{\infty} \frac{(2p-1)!!}{(2p)!!} \cdot \frac{1}{2p+1} \cdot \boldsymbol{G}^{[2p+1]}.$$

We use brackets to denote an entrywise matrix function, so $\arcsin[\boldsymbol{G}]$ applies the arcsine function to each entry of $\boldsymbol{G}$. The notation $\boldsymbol{G}^{[k]}$ refers to the kth entrywise power. By the Schur product theorem, each matrix $\boldsymbol{G}^{[k]}$ is psd. Since all coefficients in the series are positive, $\boldsymbol{K}$ is also psd. ■

Example 13.4 (Polynomial kernel). Let $\mathcal{X}$ be a subset of $\mathbb{F}^d$. For $p \in \mathbb{N}$, the inhomogeneous *polynomial kernel*

polynomial kernel

$$k(x, y) = (1 + \langle x,\ y \rangle)^p \quad \text{for } x, y \in \mathcal{X}$$

is a pd kernel. ▪

Proof. The proof is left as an exercise. ∎

Example 13.5 (Gaussian kernel). Let $\mathcal{X} = \mathbb{F}^d$. For a bandwidth $\sigma > 0$, the *Gaussian kernel* is pd: *Gaussian kernel*

$$k(x, y) = \exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right) \quad \text{for } x, y \in \mathcal{X}$$

This is a specific example of a radial basis function kernel. ▪

Proof. See the next lecture. ∎

13.1.3 The kernel trick

The reason that kernels are valuable is summarized in the following principle.

> **Idea: (The kernel trick).** "Given an algorithm which is formulated in terms of a pd kernel k, one can construct an alternative algorithm by replacing k with another pd kernel $\tilde{k}$." [SS01, Rem. 2.8].

In particular, any algorithm that can be formulated using the Gram matrix (as the only access to the data) can be ported to any other pd kernel. This approach leads to kernel-based algorithms for processing images, text, DNA, and data from many other domains!

13.2 Kernel PCA

Kernel principal component analysis (KPCA) is a method for extracting the directions of maximum variance of the data in feature space. To derive this algorithm, we first develop a dual description of ordinary principal component analysis for Euclidean data. *Kernel principal component analysis (KPCA)*

Let $\{\boldsymbol{x}_1, \dots, \boldsymbol{x}_n\} \subset \mathbb{F}^d$ be data in an ℓ_2 space. For simplicity, suppose the dataset is already centered (that is, sums to zero). We form the covariance matrix

$$\boldsymbol{C} = \frac{1}{n} \sum\nolimits_{i=1}^{n} \boldsymbol{x}_i \boldsymbol{x}_i^* = \frac{1}{n} \boldsymbol{X}\boldsymbol{X}^* \in \mathbb{H}_d.$$

The dominant eigenvector $\boldsymbol{u} \in \mathbb{F}^d$ of the matrix $\boldsymbol{C}$ is called the *first principal component*. It satisfies *first principal component*

$$\lambda \boldsymbol{u} = \boldsymbol{C}\boldsymbol{u}, \quad \text{where } \lambda > 0 \text{ and } \|\boldsymbol{u}\| = 1.$$

Since $\lambda \neq 0$, the eigenvector $\boldsymbol{u}$ is contained in $\operatorname{span}\{\boldsymbol{x}_1, \dots, \boldsymbol{x}_n\}$, so

$$\lambda \langle \boldsymbol{x}_i, \boldsymbol{u} \rangle = \langle \boldsymbol{x}_i, \boldsymbol{C}\boldsymbol{u} \rangle \quad \text{for } i = 1, \dots, n.$$

Furthermore,

$$\boldsymbol{u} = \sum\nolimits_{j=1}^{n} \alpha_j \boldsymbol{x}_j \quad \text{for } \boldsymbol{\alpha} \in \mathbb{F}^n.$$

Combining all of these relations, we obtain the following identity for each $i = 1, \dots, n$:

$$\lambda \sum\nolimits_{j=1}^{n} \alpha_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle = \frac{1}{n} \sum\nolimits_{j=1}^{n} \alpha_j \sum\nolimits_{l=1}^{n} \langle \boldsymbol{x}_i, \boldsymbol{x}_l \rangle \langle \boldsymbol{x}_l, \boldsymbol{x}_j \rangle.$$

We may rewrite these conditions in terms of the Gram matrix $\boldsymbol{K} = \boldsymbol{X}^*\boldsymbol{X}$:

$$\lambda \boldsymbol{K}\boldsymbol{\alpha} = \frac{1}{n} \boldsymbol{K}^2 \boldsymbol{\alpha}.$$

Equivalently,

$$\lambda \boldsymbol{\alpha} = \frac{1}{n} \boldsymbol{K} \boldsymbol{\alpha}.$$

This is called the *dual eigenvalue problem*. In other words, the coefficient vector $\boldsymbol{\alpha}$ that describes the first principal component as a linear combination of data points can be found by solving the dual eigenvalue problem. *dual eigenvalue problem*

We still need to determine the correct scaling for the coefficient vector $\boldsymbol{\alpha}$. To do so, note that

$$1 = \langle \boldsymbol{u}, \boldsymbol{u} \rangle = \sum_{i,j=1}^{n} \alpha_i \alpha_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle = \boldsymbol{\alpha}^* \boldsymbol{K} \boldsymbol{\alpha} = n\lambda \|\boldsymbol{\alpha}\|^2.$$

The *normalization constraint* $\|\boldsymbol{u}\| = 1$ is equivalent to demanding *normalization constraint*

$$\|\boldsymbol{\alpha}\|^2 = \frac{1}{n\lambda}.$$

This formula tells us how to rescale $\boldsymbol{\alpha}$ so we obtain a unit-norm eigenvector..

Now, given a new data point $\boldsymbol{x}$, we can evaluate its first principal component:

$$\mathrm{PC}_1(\boldsymbol{x}) = \langle \boldsymbol{u}, \boldsymbol{x} \rangle = \sum_{j=1}^{n} \alpha_j \langle \boldsymbol{x}_j, \boldsymbol{x} \rangle.$$

Observe that we can compute the coefficient vectors $\boldsymbol{\alpha}$ using only the Gram matrix $\boldsymbol{K}$. Furthermore, we can evaluate the projection of the data point $\boldsymbol{x}$ onto the first principal component using only inner products. We can also repeat the same derivation to obtain formulas for more principal components; the associated coefficient vectors are eigenvectors attached to the largest eigenvalues of the kernel matrix.

Invoking the kernel trick, we can do exactly the same thing in the kernel setting:

1. Let $\boldsymbol{K} \in \mathbb{H}_n$ be the kernel matrix.
2. Compute the dominant eigenpair $(\lambda, \boldsymbol{\alpha})$ of $\boldsymbol{K}$.
3. Scale so that $\|\boldsymbol{\alpha}\|^2 = 1/(n\lambda)$.
4. Given a new data point $\boldsymbol{x} \in \mathbb{F}^d$, its first principal component is
$$\mathrm{PC}_1(\boldsymbol{x}) = \sum_{j=1}^{n} \alpha_j k(\boldsymbol{x}_j, \boldsymbol{x}).$$
5. Similarly, we compute more principal components by computing more eigenvectors of $\boldsymbol{K}$.

This construction has a nice interpretation. In feature space, $\sum_{j=1}^{n} \alpha_j \Phi(\boldsymbol{x}_j)$ is a direction of maximum variance of the features induced by the data.

Remark 13.6 (Centering). The interpretation of principal components as directions of maximum variance relies on the assumption that the dataset is centered. It is possible to center the (implicit) feature vectors $\Phi(\boldsymbol{x}_i)$ associated with the dataset using only kernel evaluations.

13.3 Computational issues and outlook

Kernel methods are widely applicable and there are many kernelizable algorithms, including kernel nearest neighbors, kernel k-means, and kernel ridge regression. However, when implementing these approaches, we encounter a serious problem: it is very expensive to construct kernel matrices.

Indeed, if $\mathcal{X}$ has a d-dimensional parameterization, then it often takes $O(d)$ arithmetic operations per kernel evaluation, or $O(n^2 d)$ operations to compute the full kernel matrix. Moreover, after computing the kernel matrix, we still have to do some more linear algebra. For example, we anticipate that it will require $O(n^2 \ell)$ operations to compute ℓ kernel principal components.

Idea: Use randomized linear algebra to implement kernel methods more efficiently.

One challenge is that the access to the kernel matrix is very restricted; we can only compute one entry at a time. In the upcoming lectures we will introduce two approaches to address this difficulty: *Random features* and *Nyström approximation by column sampling*.

Lecture bibliography

[SS01] B. Scholkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT press, 2001.

14. Random Features

Date: 20 February 2020 **Scribe:** Yifan Chen

In the last lecture, we introduced kernel methods. The bottleneck for kernel methods is their computational cost, which calls for efficient approximation strategies. In this lecture, we will discuss a Monte Carlo method, called random features, for approximating a kernel matrix.

As a motivating example, we begin with the angular similarity kernel, and then we continue to the abstract definition of random features. Two useful examples, translation-invariant kernels and dot-product kernels, will be discussed in detail. Finally, we will touch on the streaming KPCA algorithm, which allows us to process random features with minimal storage.

Agenda:

1. Warmup: Angular similarity
2. Abstract random features
3. Translation-invariant kernels
4. Dot-product kernels
5. Streaming KPCA

14.1 Introduction

Let $k : \mathcal{X} \times \mathcal{X} \to \mathbb{F}$ be a positive-definite kernel on the input space $\mathcal{X}$. Suppose we have a family of observations $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_n\} \subset \mathcal{X}$. The associated *kernel matrix* is the psd matrix

kernel matrix

$$\boldsymbol{K} = [k(\boldsymbol{x}_i, \boldsymbol{x}_j)]_{i,j=1,...,n} \in \mathbb{H}_n.$$

We can also interpret the kernel matrix as the Gram matrix for (possibly infinite-dimensional) feature vectors extracted the observations.

Issue: Computing the kernel matrix is very expensive. Moreover, it takes a lot of storage.

This issue is very real and motivates the following question: Can we approximate $\boldsymbol{K}$ more efficiently? An effective approximation is beneficial for speeding up computations. What is more, the approximation can also act as a form of regularization, which is good from a learning point of view. The challenge for approximation is that we only have restricted access to the kernel matrix.

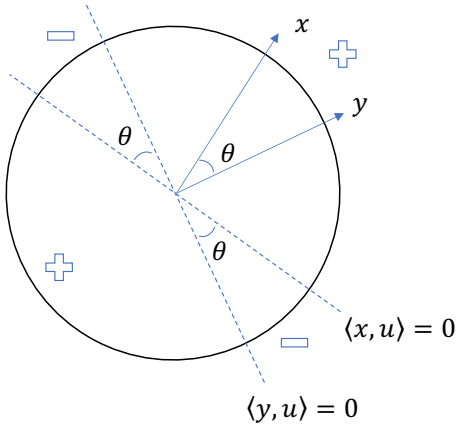


Figure 14.1 (Grothendieck identity). Proposition 14.1 in 2 dimensions: The sectors where the inner products $\langle \boldsymbol{u}, \boldsymbol{x} \rangle$ and $\langle \boldsymbol{u}, \boldsymbol{y} \rangle$ have opposite signs subtend the same angle as the sector generated by $\boldsymbol{x}$ and $\boldsymbol{y}$.

In this lecture, we will discuss a Monte Carlo method for approximating kernel matrices. We will see that there are close similarities to randomized approaches for matrix multiplication, the topic of Lecture 4.

14.2 Motivating example: Angular similarity kernels

We warm up with a concrete example: the *angular similarity kernel*. For $\mathcal{X} = \mathbb{S}^{d-1}(\mathbb{R})$, this kernel assumes the following form:

angular similarity kernel

$$k(\boldsymbol{x}, \boldsymbol{y}) = \frac{2}{\pi} \arcsin\langle \boldsymbol{x}, \boldsymbol{y} \rangle = 1 - \frac{2}{\pi}\theta(\boldsymbol{x}, \boldsymbol{y}) \quad \text{for all} \quad \boldsymbol{x}, \boldsymbol{y} \in \mathcal{X} = \mathbb{S}^{d-1}(\mathbb{R}).$$

We can relate this kernel to an expectation involving Gaussian random vectors.

Proposition 14.1 (Grothendieck identity). Let $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{S}^{d-1}(\mathbb{R})$, and draw $\boldsymbol{g} \sim \text{NORMAL}(\mathbf{0}, \mathbf{I}_d)$. Then

$$k(\boldsymbol{x}, \boldsymbol{y}) = \mathbb{E}_{\boldsymbol{g}}[\operatorname{sgn}\langle \boldsymbol{x}, \boldsymbol{g} \rangle \cdot \operatorname{sgn}\langle \boldsymbol{y}, \boldsymbol{g} \rangle].$$

Proof. By rotational invariance and linearity, we can reduce the problem to two dimensions only. Moreover, the signum function is scale-invariant, so we can pass to a random vector $\boldsymbol{u}$ that is uniformly distributed on the unit circle in $\mathbb{R}^2$. We can understand this case by looking at an illustration; see Figure 14.1.

Let $\theta := \arcsin\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ denote the angle between $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^2$. We see that the proportion of random vectors $\boldsymbol{u}$ where the product $\operatorname{sgn}\langle \boldsymbol{x}, \boldsymbol{u} \rangle \cdot \operatorname{sgn}\langle \boldsymbol{y}, \boldsymbol{u} \rangle = -1$ equals $(2\theta)/(2\pi)$. Therefore, the expectation satisfies

$$\mathbb{E}_{\boldsymbol{u}}[\operatorname{sgn}\langle \boldsymbol{x}, \boldsymbol{u} \rangle \cdot \operatorname{sgn}\langle \boldsymbol{y}, \boldsymbol{u} \rangle] = (+1) \cdot \left[1 - \frac{\theta}{\pi}\right] + (-1) \cdot \frac{\theta}{\pi} = 1 - \frac{2}{\pi}\theta.$$

This is the required result. ∎

Proposition 14.1 has many applications in computational mathematics. For instance, it features prominently in the randomized rounding procedure for approximately

solving MaxCut. For the task at hand, Proposition 14.1 allows us to recast the kernel matrix as an expectation value.

Corollary 14.2 (Angular similarity kernel matrix). Consider a set $\{\boldsymbol{x}_1, \dots, \boldsymbol{x}_n\} \subset \mathbb{S}^{d-1}(\mathbb{R})$, and draw $\boldsymbol{g} \sim \text{normal}(\boldsymbol{0}, \mathbf{I}_d)$. Construct a vector $\boldsymbol{z} \in \mathbb{R}^n$ with entries $z_i = \operatorname{sgn}\langle \boldsymbol{x}_i, \boldsymbol{g} \rangle$. Then

$$\boldsymbol{K} = \mathbb{E}[\boldsymbol{z}\boldsymbol{z}^*].$$

The random vector $\boldsymbol{z} \in \mathbb{R}^n$ appearing in Corollary 14.2 is called a *random feature*. *random feature*

Proof. For each pair (i, j), we calculate

$$(\boldsymbol{K})_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \mathbb{E}_{\boldsymbol{g}}[z_i \cdot z_j] = \mathbb{E}_{\boldsymbol{g}}[(\boldsymbol{z}\boldsymbol{z}^*)_{ij}] = (\mathbb{E}_{\boldsymbol{g}}[\boldsymbol{z}\boldsymbol{z}^*])_{ij}.$$

This result implies that the two matrices are equal. ∎

Corollary 14.2 asserts that each random features $\boldsymbol{z}$ leads to an unbiased estimator of the angular similarity kernel matrix. The computation of a random feature vector is relatively cheap. As a consequence, we can compute an iid family of random features $\{\boldsymbol{z}_1, \dots, \boldsymbol{z}_s\} \subset \mathbb{R}^n$ and approximate the kernel matrix by a sample average:

$$\widehat{\boldsymbol{K}}_s = \frac{1}{s} \sum\nolimits_{i=1}^{s} \boldsymbol{z}_i \boldsymbol{z}_i^* \quad \text{such that} \quad \mathbb{E}\, \widehat{\boldsymbol{K}}_s = \boldsymbol{K}$$

The hope is that we can achieve an accurate estimate with a moderate number s of samples, where $s \ll n$. The cost associated with this Monte Carlo approximation is $O(sdn)$, which can be *much* cheaper than the naive cost $O(dn^2)$ for computing the full kernel matrix.

An alternative view reveals close connections to *randomized matrix multiplication*. *randomized matrix multiplication*
Rewrite the kernel matrix as

$$\boldsymbol{K} = \boldsymbol{B}\boldsymbol{D}\boldsymbol{B}^*,$$

where $\boldsymbol{B} \in \{\pm 1\}^{n \times 2^n}$ tabulates all possible sign vectors and the diagonal matrix $\boldsymbol{D}$ records sampling probabilities. Effectively, we aggregate all possible choices of signs in the matrix $\boldsymbol{B}$. The sample probability matrix tells us what proportion of each sign patterns we need to represent the kernel matrix. Subsequently, we just sample from this complicated distribution using Corollary 14.2. This approach is parallel to the randomized matrix–matrix multiplication procedure we discussed in Lecture 4.

14.3 Abstract random features

Let us continue with an abstract generalization of the ideas in the previous section.

14.3.1 Random feature maps

Let $\mathcal{X}$ be an input space with a positive-definite kernel function k. Let $\mathcal{W}$ be a probability space with probability measure μ.

Definition 14.3 (Random feature map). A function $\psi : \mathcal{X} \times \mathcal{W} \to \mathbb{F}$ is called a *random feature map* for the kernel k if *random feature map*

$$\begin{aligned} k(\boldsymbol{x}, \boldsymbol{y}) &= \int \psi(\boldsymbol{x}; \boldsymbol{w}) \cdot \psi(\boldsymbol{y}; \boldsymbol{w})^* \, \mathrm{d}\mu(\boldsymbol{w}) \\ &= \mathbb{E}_{\boldsymbol{w} \sim \mu}[\psi(\boldsymbol{x}; \boldsymbol{w}) \cdot \psi(\boldsymbol{y}; \boldsymbol{w})^*] \end{aligned}$$

for all $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{X}$. That is, a random feature map reproduces the kernel evaluation in expectation.

This definition generalizes the simple and intuitive concepts that featured in our discussion of angular similarity kernels.

Example 14.4 (Angular similarity kernel) Let $\mathcal{X} = \mathbb{S}^{d-1}$. Let $\mathcal{W} = (\mathbb{R}^d, \gamma)$, where γ is the standard normal measure. The associated random feature map is $\psi(\boldsymbol{x}; \boldsymbol{w}) = \operatorname{sgn}\langle \boldsymbol{x}, \boldsymbol{w} \rangle$.

14.3.2 Kernel matrix approximation

Using the random feature map, we can develop a strategy for approximating the kernel matrix:

1. Given observations $\{\boldsymbol{x}_1, \dots, \boldsymbol{x}_n\} \subset X$, construct a random feature $\boldsymbol{z} \in \mathbb{F}^n$ by drawing $\boldsymbol{w} \sim \mu$ and computing $z_i = \psi(\boldsymbol{x}_i; \boldsymbol{w})$ for $i = 1, \dots, n$. Note, that we use the *same* random vector $\boldsymbol{w}$ for all data points.
2. By construction, $\boldsymbol{K} = \mathbb{E}_{\boldsymbol{w}}[\boldsymbol{z}\boldsymbol{z}^*]$. In other words, the random feature reproduces the kernel matrix on average over the randomness in $\boldsymbol{w}$.
3. To approximate the kernel, we draw iid random features $\boldsymbol{z}_1, \dots, \boldsymbol{z}_s \in \mathbb{F}^n$ and form the Monte Carlo approximation

$$\widehat{\boldsymbol{K}}_s = \frac{1}{s} \sum\nolimits_{i=1}^{s} \boldsymbol{z}_i \boldsymbol{z}_i^*.$$

This construction also demonstrates that feature maps only exist for positive-definite kernel functions.

14.3.3 Quality of approximation

Let us talk briefly about the quality of such an approximation. We can use the matrix Monte Carlo theorem (based on Matrix Bernstein) to derive rigorous *convergence guarantees*. Consider a kernel that obeys $k(x, x) = 1$ for all $x \in \mathcal{X}$, and suppose that the feature map is bounded: $|\psi| \leq b$. Then

convergence guarantees

$$s \geq 2b\varepsilon^{-2} \operatorname{intdim}(\boldsymbol{K}) \log(2n) \quad \text{implies} \quad \frac{\mathbb{E} \, \|\widehat{\boldsymbol{K}}_s - \boldsymbol{K}\|}{\|\boldsymbol{K}\|} \leq \varepsilon + \varepsilon^2.$$

In this setting, $\operatorname{intdim}(\boldsymbol{K}) = n/\|\boldsymbol{K}\|$, which can be much smaller than n when the data points carry some redundant information. If this is the case, we can approximate the kernel with $s \ll n$ random features.

14.3.4 Cost

If the data has a d-dimensional representation, we anticipate that it will take $O(sdn)$ arithmetic operations to compute s random features. Using a randomized SVD algorithm, we can perform kernel PCA to compute ℓ kernel principal components at an extra cost of $O(\ell s n)$.

14.4 Translation invariant kernels

In this section, we discuss an important class of kernels for which random feature maps can be constructed.

Definition 14.5 **(Translation invariant kernels).** Let $\mathcal{X} = \mathbb{F}^d$. We say that a kernel k is *translation invariant* if there is a function $\varphi : \mathbb{F}^d \to \mathbb{F}$ such that $k(\boldsymbol{x}, \boldsymbol{y}) = \varphi(\boldsymbol{x} - \boldsymbol{y})$ for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}^d$.

translation invariant

Note that the angular similarity kernel is not translation invariant, but some other prominent examples are.

Example 14.6 (Gaussian kernel). The kernel $k_\sigma(\boldsymbol{x}, \boldsymbol{y}) = \exp(-\|\boldsymbol{x} - \boldsymbol{y}\|^2/(2\sigma^2))$ is translation invariant. ∎

Translation-invariant kernels are characterized completely by a classical result due to Bochner.

Theorem 14.7 (Bochner, 1930). A continuous translation invariant kernel $k(\boldsymbol{x}, \boldsymbol{y}) = \varphi(\boldsymbol{x} - \boldsymbol{y})$ is positive definite if and only if

$$k(\boldsymbol{x}, \boldsymbol{y}) = \text{const} \cdot \int_{\mathbb{F}^d} \mathrm{e}^{\mathrm{i}\langle \boldsymbol{x},\, \boldsymbol{w}\rangle} \mathrm{e}^{-\mathrm{i}\langle \boldsymbol{y},\, \boldsymbol{w}\rangle} \,\mathrm{d}\mu(\boldsymbol{w})$$

for some Borel probability measure μ on $\mathbb{F}^d$. That is,

$$\varphi(\boldsymbol{u}) = \text{const} \cdot \int_{\mathbb{F}^d} \mathrm{e}^{\mathrm{i}\langle \boldsymbol{u},\, \boldsymbol{w}\rangle} \,\mathrm{d}\mu(\boldsymbol{w})$$

is the Fourier transform of a probability measure.

One direction of Bochner's theorem is easy. (In fact, we already proved it.) The converse direction is moderately hard. In practice, we typically use the easy direction. Indeed, if we set $\mathscr{W} = (\mathbb{F}^d, \mu)$, then the function

$$\psi(\boldsymbol{x}; \boldsymbol{w}) = \sqrt{\text{const}} \cdot \mathrm{e}^{\mathrm{i}\langle \boldsymbol{x},\, \boldsymbol{w}\rangle} \quad \text{for } \boldsymbol{x} \in \mathbb{F}^d$$

is a random feature map for the kernel $k(\boldsymbol{x}, \boldsymbol{y}) = \varphi(\boldsymbol{x} - \boldsymbol{y})$.

Example 14.8 (Gaussian kernel). For the Gaussian kernel, the probability measure promised by Bochner's theorem is $\mu \sim \textsc{normal}(\mathbf{0}, \sigma^{-2}\mathbf{I}_d)$. We leave the details of the argument as an exercise. ∎

14.5 Dot-product kernels

In this section, we briefly touch on a second class of kernels that admit random feature maps.

Definition 14.9 (Dot-product kernels). A kernel k is called a *dot-product kernel* if $k(\boldsymbol{x}, \boldsymbol{y}) = f(\langle \boldsymbol{x},\ \boldsymbol{y}\rangle)$ for some scalar-valued function f. *dot-product kernel*

Example 14.10 (Angular similarity kernel). The angular similarity kernel is a dot-product kernel with $f(t) = (2/\pi)\arcsin(t)$ for $t \in [-1, 1]$. ∎

Example 14.11 (Inhomogeneous polynomial kernel). This kernel is generated by the function $f(t) = (1 + t)^p$ for some $p \in \mathbb{N}$. ∎

Theorem 14.12 (Schoenberg, 1942). A dot-product kernel $k(\boldsymbol{x}, \boldsymbol{y}) = f(\langle \boldsymbol{x},\ \boldsymbol{y}\rangle)$ is positive definite on $B(0; r) \subset \mathbb{F}^d$ if and only if the function $f(t) = \sum_{p=0}^{\infty} a_p t^p$ where $a_p \geq 0$ and the series converges for $|t| \leq r$.

One direction is easy and follows from the Schur product theorem. This is how we checked that the aforementioned examples are positive-definite kernels. The converse direction is moderately hard. It is a nontrivial problem to construct a random feature map using Schoenberg's theorem, but it can be done [KK12].

14.6 Streaming KPCA

We still have not talked about the storage issues. Indeed, the random features and the associated kernel matrix approximation can take a substantial amount of space. Let us discuss an approach to control the space complexity of performing KPCA.

Idea: Use a streaming PCA algorithm to process random features.

This approach is called *Streaming Kernel PCA* [GPP16]. We will outline a variant proposed in [MT20, Sec. 19]. *Streaming Kernel PCA*

Let $\boldsymbol{z}_1, \boldsymbol{z}_2, \boldsymbol{z}_3, \ldots$ be a sequence of random features for a dataset. Define the kernel matrix approximations

$$\widehat{\boldsymbol{K}}_s = \frac{1}{s} \sum\nolimits_{i=1}^{s} \boldsymbol{z}_i \boldsymbol{z}_i^* \quad \text{for } s \in \mathbb{N}.$$

Observe that we can rewrite this formula as a dynamical system

$$\widehat{\boldsymbol{K}}_{s+1} = \left[1 - \frac{1}{s+1}\right] \widehat{\boldsymbol{K}}_s + \frac{1}{s+1} \boldsymbol{z}_{s+1} \boldsymbol{z}_{s+1}^* \quad \text{for each } s \in \mathbb{N}. \tag{14.1}$$

This is a linear update rule. As a consequence, we can use a streaming SVD method to reduce the storage! Moreover, since $\widehat{\boldsymbol{K}}_s$ is psd, we can use a specialized algorithm to improve the performance.

Here is a summary of the procedure:

1. Draw and fix a random test matrix $\boldsymbol{\Omega} \in \mathbb{F}^{n \times \ell}$, where ℓ is roughly twice the number of principal components to be computed.
2. Maintain $\boldsymbol{Y}_s = \widehat{\boldsymbol{K}}_s \boldsymbol{\Omega}$, and update it using the linear update rule (14.1).
3. Once we have processed a sufficient number s of random features, we compute a *Nyström approximation* of $\widehat{\boldsymbol{K}}_s$: *Nyström approximation*

$$\widetilde{\boldsymbol{K}}_s = \boldsymbol{Y}_s (\boldsymbol{\Omega}^* \boldsymbol{Y}_s)^\dagger \boldsymbol{Y}_s^* = (\widehat{\boldsymbol{K}}_s \boldsymbol{\Omega})(\boldsymbol{\Omega}^* \widehat{\boldsymbol{K}}_s \boldsymbol{\Omega})^\dagger (\widehat{\boldsymbol{K}}_s \boldsymbol{\Omega})^*.$$

This method offers a *very* space-efficient technique for kernel PCA. The storage cost is only $O(n\ell)$.

To analyze this algorithm, observe that

$$\widehat{\boldsymbol{K}}_s - \widetilde{\boldsymbol{K}}_s = \widehat{\boldsymbol{K}}_s / \boldsymbol{\Omega}$$

is a Schur complement. This identity suggests that we can exploit the analysis of the randomized rangefinder (Lecture 10) to understand the performance of this approximation. See [MT20, Sec. 14] for more discussion, or see the paper [Tro+17a] for details.

Lecture bibliography

[GPP16] M. Ghashami, D. J. Perry, and J. Phillips. "Streaming Kernel Principal Component Analysis". In: *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*. Edited by A. Gretton and C. C. Robert. Volume 51. Proceedings of Machine Learning Research. Cadiz, Spain: PMLR, 2016, pages 1365–1374. URL: http://proceedings.mlr.press/v51/ghashami16.html.

[KK12] P. Kar and H. Karnick. "Random Feature Maps for Dot Product Kernels". In: *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*. Edited by N. D. Lawrence and M. Girolami. Volume 22. Proceedings of Machine Learning Research. La Palma, Canary Islands: PMLR, 2012, pages 583–591. URL: http://proceedings.mlr.press/v22/kar12.html.

[MT20] P.-G. Martinsson and J. Tropp. "Randomized Numerical Linear Algebra: Foundations & Algorithms". In: *Acta Numerica, 2020*. Cambridge Univ. Press, 2020, pages 403–572. arXiv: 2002.01387.

[Tro+17a] J. A. Tropp et al. "Fixed-Rank Approximation of a Positive-Semidefinite Matrix from Streaming Data". In: *Advances in Neural Information Processing Systems 30*. Edited by I. Guyon et al. Curran Associates, Inc., 2017, pages 1225–1234. URL: http://papers.nips.cc/paper/6722-fixed-rank-approximation-of-a-positive-semidefinite-matrix-from-streaming-data.pdf.

15. Kernel Sampling

Date: 25 February 2020 **Scribe:** Zongyi Li

In the past two lectures, we have discussed kernel methods, random features maps, and streaming kernel PCA. In this lecture, we will discuss a more direct approach for approximating the kernel matrix by sampling the coordinates. This approach relies on a matrix approximation that dates back to work of Nyström [Nys+30] on integral equations.

We begin with a recap of kernel matrices and the computational challenges that arise. To address these challenges, we consider the idea of sampling coordinates and using the sampled coordinates to approximate the matrix. Afterward, we discuss a regularization method that can improve the quality of approximations. This regularization leads us to consider a specific set of sampling probabilities, called ridge leverage scores. Although this sampling distribution is mathematically appealing, it is quite hard to compute. We give a brief discussion of some of the methods that have been proposed for this task, although it remains a subject of ongoing research.

Agenda:

1. Sampling kernel matrices
2. Coordinate Nyström Approximation
3. Regularized Nyström Approximation
4. Ridge leverage score sampling
5. Approximating ridge leverage score

15.1 Motivation: Kernel approximation

Let $\mathcal{X}$ be an input space and $k : \mathcal{X} \times \mathcal{X} \to \mathbb{F}$ a positive-definite kernel. Let $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_n\} \subset \mathcal{X}$ be a dataset with the associated psd kernel matrix

$$\boldsymbol{K} = [k(\boldsymbol{x}_i, \boldsymbol{x}_j)]_{i,j=1,...,n} \in \mathbb{H}_n.$$

In most situations, we have to pay for each individual entry of the kernel matrix that we access. As a consequence, working with kernel matrices can be very expensive.

When the data points have a d-dimensional parameterization, the cost of forming $\boldsymbol{K}$ is typically $O(dn^2)$ operations because we need to compute $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ for each pair (i, j). Furthermore, the kernel matrix requires $O(n^2)$ storage. It is natural to seek methods for working with kernel matrices that do not require us to form the entire matrix.

Idea: Can we approximate a general kernel matrix without forming it explicitly?

In the last lecture, we studied random feature maps. For kernels that admit a random feature map, we can compute s random features using $O(sdn)$ operations plus $O(sn)$ storage. The number s of random features we need is proportional to the intrinsic dimension of the kernel matrix.

In this lecture, we discuss a more direct approach to approximate kernel matrices by sampling columns. Unfortunately, it is complicated to design a theoretically sound procedure for coordinate sampling, and the analysis is not very transparent. The goal of this lecture is to offer an introduction to this circle of ideas, without too many details.

15.2 The Nyström method

The Nyström method is the most natural way to construct a low-rank approximation of a psd matrix. We begin with the general construction, which is closely related to the rangefinder problem. Afterward, we specialize to the case of column sampling.

15.2.1 Abstract Nyström approximation

Consider a psd input matrix $\boldsymbol{A} \in \mathbb{H}_n$. Let $\boldsymbol{\Omega} \in \mathbb{F}^{n \times s}$ be an arbitrary test matrix, and form the sample matrix $\boldsymbol{Y} = \boldsymbol{A}\boldsymbol{\Omega} \in \mathbb{F}^{n \times s}$. The *Nyström approximation* of the input matrix $\boldsymbol{A}$ with respect to (the range of) the test matrix $\boldsymbol{\Omega}$ is the matrix

$$\boldsymbol{A}\langle\boldsymbol{\Omega}\rangle := \boldsymbol{Y}(\boldsymbol{\Omega}^*\boldsymbol{Y})^\dagger\boldsymbol{Y}^* = (\boldsymbol{A}\boldsymbol{\Omega})(\boldsymbol{\Omega}^*\boldsymbol{A}\boldsymbol{\Omega})^\dagger(\boldsymbol{A}\boldsymbol{\Omega})^*. \tag{15.1}$$

As usual, $\dagger$ denotes the pseudoinverse. The error incurred by approximating $\boldsymbol{A}$ with $\boldsymbol{A}\langle\boldsymbol{\Omega}\rangle$ is precisely the Schur complement of $\boldsymbol{A}$ with respect to $\boldsymbol{\Omega}$:

$$\boldsymbol{A} - \boldsymbol{A}\langle\boldsymbol{\Omega}\rangle = \boldsymbol{A}/\boldsymbol{\Omega}.$$

This observation shows that there is a close connection between Nyström approximations and the rangefinder problem. In particular, for random test matrices, we can invoke our analysis of the randomized SVD to obtain bounds for the error in the Nyström approximation.

Before moving on to discuss an important special case, let us quickly review the arithmetic cost associated with a general Nyström approximation. The cost is dominated by the matrix–matrix multiply $\boldsymbol{A}\boldsymbol{\Omega}$, which requires $O(n^2 s)$ arithmetic operations. This cost can be reduced if either $\boldsymbol{A}$ or $\boldsymbol{\Omega}$ admits a fast multiply. The subsequent approximation steps involve $O(ns^2)$ arithmetic operations. Meanwhile, storage costs are $O(ns)$.

15.2.2 Column Nyström approximation

The Nyström approximation (15.1) can be constructed for any test matrix $\boldsymbol{\Omega}$. Historically, these approximations were constructed using test matrices that select columns from the input matrix. This special case is also the focus of today's lecture. Indeed, we cannot easily multiply a kernel matrix by an arbitrary vector to form an abstract Nyström approximation, but we can extract individual columns by evaluating a relatively small number of entries.

Consider a test matrix of the form

$$\boldsymbol{\Omega} = \boldsymbol{S} = \begin{bmatrix} \mathbf{e}_{j_1} & \dots & \mathbf{e}_{j_s} \end{bmatrix} \quad \text{where} \quad J = \{j_1, \dots, j_s\} \subset \{1, \dots, n\}.$$

In this case, the sample matri $\boldsymbol{Y} = \boldsymbol{A}\boldsymbol{S} = \boldsymbol{A}(:, J)$ extracts precisely s columns of the input matrix $\boldsymbol{A}$. The compression $\boldsymbol{S}^*\boldsymbol{A}\boldsymbol{S} = A(J, J)$ is a $s \times s$ submatrix of $\boldsymbol{A}$. Pictorially,

$$\boldsymbol{S} = \underbrace{\begin{bmatrix} * & & \\ & & \\ & * & \\ & & * \end{bmatrix}}_{s} \Bigg\} n \quad \text{and} \quad \boldsymbol{S}^*\boldsymbol{A}\boldsymbol{S} = \boldsymbol{A}(J, J) = \begin{bmatrix} & * & * & * \\ & & & \\ & * & * & * \\ & * & * & * \end{bmatrix}.$$

For this special case, the Nyström approximation only depends on the index set J. We introduce special notation:

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

Computing a column Nyström approximation is relatively fast, because it only requires evaluation of s individual columns from $\boldsymbol{A}$.

Let us emphasize the connection with Schur complements. Without loss, we may permute the columns of the input matrix so that the index set $J = \{1, \dots, s\}$ contains the first s column indices. Write the psd matrix $\boldsymbol{A}$ as a conformal block matrix:

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{21}^* \\ \boldsymbol{A}_{21} & \boldsymbol{A}_{22} \end{bmatrix} \quad \text{such that} \quad \boldsymbol{A}(:, J) = \begin{bmatrix} \boldsymbol{A}_{11} \\ \boldsymbol{A}_{21} \end{bmatrix}, \quad \text{and} \quad \boldsymbol{A}(J, J) = \boldsymbol{A}_{11}.$$

If $\boldsymbol{A}_{11}$ is invertible, the Nyström approximation becomes

$$\boldsymbol{A}\langle J \rangle = \begin{bmatrix} \boldsymbol{A}_{11} \\ \boldsymbol{A}_{21} \end{bmatrix} \boldsymbol{A}_{11}^{-1} \begin{bmatrix} \boldsymbol{A}_{11} \\ \boldsymbol{A}_{21} \end{bmatrix}^* = \begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{21}^* \\ \boldsymbol{A}_{21} & \boldsymbol{A}_{21}\boldsymbol{A}_{11}^{-1}\boldsymbol{A}_{12} \end{bmatrix}.$$

The approximation error is precisely the Schur complement:

$$\boldsymbol{A} - \boldsymbol{A}\langle J \rangle = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{A}_{22} - \boldsymbol{A}_{21}\boldsymbol{A}_{11}^{-1}\boldsymbol{A}_{12} \end{bmatrix} = \boldsymbol{A}/\boldsymbol{A}_{11} = \boldsymbol{A}/\boldsymbol{A}(J, J).$$

Equivalently, we apply Gaussian elimination to remove the first s rows and columns of the matrix $\boldsymbol{A}$, which results in the Schur complement.

15.3 Regularized Nyström approximation

Numerically, we must take care in forming a Nyström approximation to avoid inversion of a badly conditioned matrix. An alternative approach is to regularize the approximation to improve the conditioning. This modification is closely related to ridge regression (or Tykhonov regularization), and it can also be valuable for improving the statistical properties of kernel computations.

For a parameter $\varepsilon > 0$, the regularized Nyström approximation adds The scalar matrix $\varepsilon\mathbf{I}$ before computing the inverse. More precisely, we consider the approximations

$$\begin{aligned} \boldsymbol{A}\langle \boldsymbol{\Omega} \rangle_\varepsilon &:= (\boldsymbol{A}\boldsymbol{\Omega})(\boldsymbol{\Omega}^*\boldsymbol{A}\boldsymbol{\Omega} + \varepsilon\mathbf{I})^{-1}(\boldsymbol{A}\boldsymbol{\Omega})^*; \\ \boldsymbol{A}\langle J \rangle_\varepsilon &:= \boldsymbol{A}(:, J)(\boldsymbol{A}(J, J) + \varepsilon\mathbf{I})^{-1}\boldsymbol{A}(J, :). \end{aligned}$$

The first formula is the regularization of an abstract Nyström approximation, and the second describes regularization of a column Nyström approximation.

Proposition 15.1 (Monotonicity). For any test matrix $\boldsymbol{\Omega}$, the mapping $\varepsilon \mapsto \boldsymbol{A}\langle\boldsymbol{\Omega}\rangle_\varepsilon$ is operator monotone decreasing. That is,

$$\varepsilon \le \varepsilon' \quad \text{implies} \quad \boldsymbol{A}\langle\boldsymbol{\Omega}\rangle_{\varepsilon'} \preccurlyeq \boldsymbol{A}\langle\boldsymbol{\Omega}\rangle_\varepsilon.$$

In particular, $\boldsymbol{A}\langle\boldsymbol{\Omega}\rangle_\varepsilon \preccurlyeq \boldsymbol{A}$ for any $\varepsilon > 0$

Exercise 15.2 (Monotonicity). Prove Proposition 15.1.

To understand the behavior of the regularized Nyström approximation, we define the *smoothed projection matrix*

smoothed projection matrix

$$\boldsymbol{P}_\varepsilon = \boldsymbol{A}(\boldsymbol{A} + \varepsilon\mathbf{I})^{-1}.$$

This matrix filters the eigenvalues of $\boldsymbol{A}$ to produce an approximate projector on a dominant eigenspace of $\boldsymbol{A}$.

Proposition 15.3 (Smoothed projection matrix). The smoothed projection matrix $\boldsymbol{P}_\varepsilon$ satisfies $\mathbf{0} \preccurlyeq \boldsymbol{P}_\varepsilon \preccurlyeq \mathbf{I}$. More precisely,

$$\lambda_i(\boldsymbol{P}_\varepsilon) = \frac{\lambda_i(\boldsymbol{A})}{\lambda_i(\boldsymbol{A}) + \varepsilon} \quad \text{for each } i = 1, \dots, n.$$

In particular,

$$\boldsymbol{P}_\varepsilon \to \mathbf{0} \quad \text{as } \varepsilon \to \infty;$$
$$\boldsymbol{P}_\varepsilon \to \boldsymbol{P}_{\boldsymbol{A}} \quad \text{as } \varepsilon \to 0.$$

As usual, $\boldsymbol{P}_{\boldsymbol{A}}$ is the orthogonal projector onto the range of $\boldsymbol{A}$.

Exercise 15.4 (Smoothed projection matrix). Prove Proposition 15.3.

We are now prepared to develop an expression for the error in the regularized Nyström approximation in terms of the smoothed projector $\boldsymbol{P}_\varepsilon$ and the test matrix $\boldsymbol{\Omega}$.

Proposition 15.5 (Regularized Nyström: Approximation error). The error in the regularized Nyström approximation can be written in terms of the smoothed projection matrix:

$$\mathbf{0} \preccurlyeq \boldsymbol{A} - \boldsymbol{A}\langle\boldsymbol{\Omega}\rangle_\varepsilon = \varepsilon \cdot \boldsymbol{P}_\varepsilon^{1/2}\left(\mathbf{I} - \boldsymbol{P}_\varepsilon^{1/2}(\mathbf{I} - \boldsymbol{\Omega}\boldsymbol{\Omega}^*)\boldsymbol{P}_\varepsilon^{1/2}\right)^{-1}\boldsymbol{P}_\varepsilon^{1/2}$$

In particular,

$$\lambda_{\max}(\boldsymbol{P}_\varepsilon^{1/2}(\mathbf{I} - \boldsymbol{\Omega}\boldsymbol{\Omega}^*)\boldsymbol{P}_\varepsilon^{1/2}) \le \alpha \quad \text{implies} \quad \boldsymbol{A} - \boldsymbol{A}\langle\boldsymbol{\Omega}\rangle_\varepsilon \preccurlyeq \frac{\varepsilon}{1-\alpha}\boldsymbol{P}_\varepsilon.$$

Proof. Define the matrix $\boldsymbol{R} = \boldsymbol{A}^{1/2}\boldsymbol{\Omega}$. We can rewrite the approximation error as

$$\begin{aligned}\boldsymbol{A} - \boldsymbol{A}\langle\boldsymbol{\Omega}\rangle_\varepsilon &= \boldsymbol{A} - \boldsymbol{A}^{1/2}\boldsymbol{R}(\boldsymbol{R}^*\boldsymbol{R} + \varepsilon\mathbf{I})^{-1}\boldsymbol{R}^*\boldsymbol{A}^{1/2} \\ &= \boldsymbol{A}^{1/2}(\mathbf{I} - \boldsymbol{R}(\boldsymbol{R}^*\boldsymbol{R} + \varepsilon\mathbf{I})^{-1}\boldsymbol{R}^*)\boldsymbol{A}^{1/2}.\end{aligned}$$

Direct calculation reveals that

$$\mathbf{I} - \boldsymbol{R}(\boldsymbol{R}^*\boldsymbol{R} + \varepsilon\mathbf{I})^{-1}\boldsymbol{R}^* = \varepsilon \cdot (\boldsymbol{R}\boldsymbol{R}^* + \varepsilon\mathbf{I})^{-1}.$$

The inverted matrix admits an alternative expression:

$$\begin{aligned}\boldsymbol{R}^*\boldsymbol{R} + \varepsilon\mathbf{I} = \boldsymbol{A}^{1/2}\boldsymbol{\Omega}\boldsymbol{\Omega}^*\boldsymbol{A}^{1/2} + \varepsilon\mathbf{I} &= \boldsymbol{A} + \varepsilon\mathbf{I} - \boldsymbol{A}^{1/2}\left(\mathbf{I} - \boldsymbol{\Omega}\boldsymbol{\Omega}^*\right)\boldsymbol{A}^{1/2} \\ &= (\boldsymbol{A} + \varepsilon\mathbf{I})^{1/2}\left(\mathbf{I} - \boldsymbol{P}_\varepsilon^{1/2}\left(\mathbf{I} - \boldsymbol{\Omega}\boldsymbol{\Omega}^*\right)\boldsymbol{P}^{1/2}\right)(\boldsymbol{A} + \varepsilon\mathbf{I})^{1/2}.\end{aligned}$$

Combine these three displays, and identify copies of the matrix $\boldsymbol{P}_\varepsilon^{1/2}$ to complete the argument. ∎

15.4 Ridge leverage scores

Proposition 15.5 shows how to express the approximation error of a regularized Nyström approximation in terms of the largest eigenvalue of the matrix $\boldsymbol{P}_\varepsilon^{1/2}(\mathbf{I} - \boldsymbol{\Omega}\boldsymbol{\Omega}^*)\boldsymbol{P}_\varepsilon^{1/2}$. The smaller this eigenvalue, the better the approximation.

To that end, observe that the error matrix can be expressed in the form

$$\boldsymbol{P}_\varepsilon^{1/2} \cdot \boldsymbol{P}_\varepsilon^{1/2} - \boldsymbol{P}_\varepsilon^{1/2}\boldsymbol{\Omega}\boldsymbol{\Omega}^*\boldsymbol{P}_\varepsilon^{1/2}.$$

In other words, we need a test matrix $\boldsymbol{\Omega}$ that controls the error in an approximate matrix multiplication.

Idea: Use importance sampling to bound the error $\boldsymbol{P}_\varepsilon^{1/2}(\mathbf{I} - \boldsymbol{\Omega}\boldsymbol{\Omega}^*)\boldsymbol{P}_\varepsilon^{1/2}$.

In Lecture 4, we studied how to choose sampling distributions to control the error in approximate matrix multiplication. Using the insights from this analysis, we realize that the sampling probabilities should be proportional to the diagonal entries of the smoothed projector:

$$p_i \propto \|\boldsymbol{P}_\varepsilon^{1/2}\mathbf{e}_i\|^2 = \mathbf{e}_i^*\boldsymbol{P}_\varepsilon\mathbf{e}_i = (\boldsymbol{P}_\varepsilon)_{ii}.$$

It is convenient to introduce notation and terminology related to these quantities.

Definition 15.6 (Ridge leverage scores). For a regularization parameter $\varepsilon > 0$, the *ridge leverage scores (RLS)* of a psd matrix $\boldsymbol{K} \in \mathbb{H}_n$ are the quantities

ridge leverage scores (RLS)

$$\ell_i(\varepsilon) := (\boldsymbol{P}_\varepsilon)_{ii} = [\boldsymbol{K}(\boldsymbol{K} + \varepsilon\mathbf{I})^{-1}]_{ii} \qquad \text{for } i = 1, \dots, n.$$

The sum of the ridge leverage scores is called the *effective dimension* of the regularized projector:

effective dimension

$$d_{\text{eff}}(\varepsilon) := \sum\nolimits_{i=1}^{n} \ell_i(\varepsilon).$$

These quantities have a long history in statistics. Kernel ridge regression is an optimization problem of the form

$$\text{minimize} \quad \|\boldsymbol{K}^{1/2}(\boldsymbol{u} - \boldsymbol{b})\|_2^2 + \varepsilon \cdot \|\boldsymbol{u}\|_2^2.$$

The ith ridge leverage score $\ell_i(\varepsilon)$ is a measure of the influence of the ith data point on the solution to the ridge regression problem. The effective dimension $d_{\text{eff}}(\varepsilon)$ captures the number of degrees of freedom in the model.

Given the ridge leverage scores, we can construct a sampling distribution over the data points.

$$p_i = \frac{\ell_i(\varepsilon)}{d_{\text{eff}}(\varepsilon)} \quad \text{for } i = 1, \dots, n$$

If we sample enough columns from this distribution, then we can control the size of the error in the regularized Nyström approximation. Here is a more precise version of this claim.

Proposition 15.7 (RLS sampling). For a parameter $\beta > 0$, consider a probability distribution that satisfies

$$p_i \geq \beta \cdot \frac{\ell_i(\varepsilon)}{d_{\text{eff}}(\varepsilon)} \quad \text{for each } i = 1, \dots, n.$$

Let $\boldsymbol{S} \in \mathbb{F}^{n \times s}$ be a sampling matrix whose columns select s coordinates drawn independently from the probability distribution. For each $t \in (0, 1)$,

$$\mathbb{P}\left\{\lambda_{\max}(\boldsymbol{P}_\varepsilon^{1/2}(\mathbf{I} - \boldsymbol{S}\boldsymbol{S}^*)\boldsymbol{P}_\varepsilon^{1/2}) \geq t\right\} \leq n \exp\left(\frac{-st^2/2}{d_{\text{eff}}(\varepsilon/\beta + t/3)}\right).$$

On the complement of this event,

$$\mathbf{0} \preccurlyeq \boldsymbol{K} - \boldsymbol{K}\langle \boldsymbol{S} \rangle_\varepsilon \preccurlyeq \frac{\varepsilon}{1-t} \boldsymbol{P}_\varepsilon,$$

where $\boldsymbol{P}_\varepsilon = \boldsymbol{K}(\boldsymbol{K} + \varepsilon \mathbf{I})^{-1}$ is the smoothed projector.

Exercise 15.8 (RLS sampling). Prove Proposition 15.7

This result indicates that we can approximate the kernel matrix $\boldsymbol{K}$ by sampling about $d_{\mathrm{eff}}(\varepsilon)$ coordinates from the RLS distribution and forming a regularized Nyström approximation. This approach is efficient when the effective dimension is small.

15.5 Approximating ridge leverage scores

In the last section, we have seen that it is possible to approximate a psd matrix by sampling a moderate number of columns from the RLS distribution. This raises another question:

> How do we compute the ridge leverage scores?

In a sense, estimating the RLS distribution is just as hard as the original problem of approximating the input matrix. Although there are many papers [AM15; MM17; Rud+18] that contain algorithms for estimating the RLS distribution, there is still no fully satisfactory solution to the problem. In the remainder of this section, we outline some of the ideas that have been proposed for this task.

> **Idea:** Use uniform sampling to compute a coarse approximation of $\boldsymbol{K}$. Use the coarse approximation to estimate the RLS distribution.

Let us explain how one might implement and analyze this approach. Fix a sampling set J, and form the Nyström approximation $\boldsymbol{K}\langle J \rangle$. Compute a Cholesky decomposition: $\boldsymbol{K}\langle J \rangle = \boldsymbol{B}\boldsymbol{B}^*$. Then we can estimate the RLS using the formula

$$\hat{\ell}_i(\varepsilon) = \boldsymbol{b}_i^* (\boldsymbol{B}^* \boldsymbol{B} + \varepsilon \mathbf{I})^{-1} \boldsymbol{b}_i.$$

In this expression, $\boldsymbol{b}_i$ is the ith row of the Cholesky factor $\boldsymbol{B}$. The next result indicates why this estimate is sensible.

Proposition 15.9 (RLS estimates). Instate the prevailing notation. Then

$$\hat{\ell}_i(\varepsilon) = \big(\boldsymbol{K}\langle J \rangle \cdot (\boldsymbol{K}\langle J \rangle + \varepsilon \mathbf{I})^{-1} \big)_{ii}.$$

Exercise 15.10 (RLS estimates). Prove Proposition 15.9.

Proposition 15.11 (RLS estimates via diagonal sampling). Let $t > 0$ be a parameter. Suppose that we sample a set J consisting of $s \geq \operatorname{tr}(\boldsymbol{K})/(t\varepsilon)$ points from the probability distribution $p_i = \boldsymbol{K}_{ii} / \operatorname{tr} \boldsymbol{K}$. With high probability, the RLS estimates satisfy

$$\ell_i(\varepsilon) - 2t \leq \hat{\ell}_i(\varepsilon) \leq \ell_i(\varepsilon), \qquad \text{for all } i = 1, \dots, n.$$

Exercise 15.12 (RLS estimates via diagonal sampling). Prove Proposition 15.11.

This result indicates that we can obtain reasonable estimates for the large elements of the RLS distribution using a simple sampling scheme that involves minimal computation. Nevertheless, this approach is not powerful enough to estimate the smaller elements without an exorbitant number of samples.

Idea: Refine the RLS estimates by sampling repeatedly, using homotopy to reduce the parameter ε.

Here is how this approach works. For a large value of ε_0, we use diagonal sampling to obtain estimates $\hat{\ell}_i(\varepsilon_0)$ for the RLS at the scale ε_0. We reduce the scale by a constant factor: $\varepsilon_1 = c \cdot \varepsilon_0$ where $c < 1$. We use our RLS estimates $\hat{\ell}_i(\varepsilon_0)$ to sample more coordinates, and we obtain new estimates $\hat{\ell}_i(\varepsilon_1)$ for the RLS at the scale ε_1. This process is repeated about $\log(\varepsilon_0/\varepsilon)$ times, where ε is the smallest scale.

It is possible to justify this approach and provide estimates for the number of samples required at each steps. Algorithms based on this type of scheme have reasonable performance in practice, but they are still not fully satisfactory.

15.6 History

The Nyström approximation was originally developed in the context of integral equations [Nys+30]. It has had a substantial impact in machine learning, beginning with the work of Williams and Seeger [WS01] on the randomized low-rank approximation of kernel matrices. The paper of Bach [Bac13] clarified the role of sampling for approximating kernel matrices. Alaoui and Mahoney [AM15] proposed the sampling scheme based on approximate ridge leverage scores and developed the initial estimate based on diagonal sampling. Musco and Musco [MM17] introduced a recursive method for estimating the RLS distribution. The current state of the art for RLS sampling is the paper of Rudi et al.[Rud+18].

Lecture bibliography

[AM15] A. Alaoui and M. W. Mahoney. "Fast randomized kernel ridge regression with statistical guarantees". In: *Advances in Neural Information Processing Systems*. 2015, pages 775–783.

[Bac13] F. Bach. "Sharp analysis of low-rank kernel matrix approximations". In: *Conference on Learning Theory*. 2013, pages 185–209.

[MM17] C. Musco and C. Musco. "Recursive sampling for the nystrom method". In: *Advances in Neural Information Processing Systems*. 2017, pages 3833–3845.

[Nys+30] E. J. Nyström et al. "Über die praktische Auflösung von Integralgleichungen mit Anwendungen auf Randwertaufgaben". In: *Acta Mathematica* 54 (1930), pages 185–204.

[Rud+18] A. Rudi et al. "On fast leverage score sampling and optimal learning". In: *Advances in Neural Information Processing Systems*. 2018, pages 5672–5682.

[WS01] C. K. Williams and M. Seeger. "Using the Nyström method to speed up kernel machines". In: *Advances in neural information processing systems*. 2001, pages 682–688.

Bibliography

[AW02] R. Ahlswede and A. Winter. "Strong converse for identification via quantum channels". In: *IEEE Trans. Inform. Theory* 48.3 (2002), pages 569–579. DOI: 10.1109/18.985947.

[AM15] A. Alaoui and M. W. Mahoney. "Fast randomized kernel ridge regression with statistical guarantees". In: *Advances in Neural Information Processing Systems*. 2015, pages 775–783.

[AMS99] N. Alon, Y. Matias, and M. Szegedy. "The space complexity of approximating the frequency moments". In: volume 58. 1, part 2. Twenty-eighth Annual ACM Symposium on the Theory of Computing (Philadelphia, PA, 1996). 1999, pages 137–147. DOI: 10.1006/jcss.1997.1545.

[Alo+02] N. Alon et al. "Tracking join and self-join sizes in limited storage". In: volume 64. 3. Special issue on PODS 1999 (Philadelphia, PA). 2002, pages 719–747. DOI: 10.1006/jcss.2001.1813.

[AMT10] H. Avron, P. Maymounkov, and S. Toledo. "Blendenpik: supercharging Lapack's least-squares solver". In: *SIAM J. Sci. Comput.* 32.3 (2010), pages 1217–1236. DOI: 10.1137/090767911.

[Bac13] F. Bach. "Sharp analysis of low-rank kernel matrix approximations". In: *Conference on Learning Theory*. 2013, pages 185–209.

[Bai+98] Z. Bai et al. *Computing Partial Eigenvalue Sum in Electronic Structure Calculations*. Technical report. Stanford University, 1998. URL: https://web.cs.ucdavis.edu/~bai/publications/baifaheygolubetal98.pdf.

[BFG96] Z. Bai, M. Fahey, and G. Golub. "Some large-scale matrix computation problems". In: volume 74. 1-2. TICAM Symposium (Austin, TX, 1995). 1996, pages 71–89. DOI: 10.1016/0377-0427(96)00018-0.

[Bha97] R. Bhatia. *Matrix analysis*. Volume 169. Graduate Texts in Mathematics. Springer-Verlag, New York, 1997, pages xii+347. DOI: 10.1007/978-1-4612-0653-8.

[Bjö96] Å. Björck. *Numerical Methods for Least Squares Problems*. Society for Industrial and Applied Mathematics, Jan. 1996. DOI: 10.1137/1.9781611971484.

[Bub15] S. Bubeck. "Convex Optimization: Algorithms and Complexity". In: *Found. Trends Mach. Learn.* 8.3–4 (Nov. 2015), pages 231–357. DOI: 10.1561/2200000050.

[Car85] B. Carl. "Inequalities of Bernstein-Jackson-type and the degree of compactness of operators in Banach spaces". In: *Ann. Inst. Fourier (Grenoble)* 35.3 (1985), pages 79–118. URL: http://www.numdam.org/item?id=AIF_1985__35_3_79_0.

[Che+05] H. Cheng et al. "On the Compression of Low Rank Matrices". In: *SIAM Journal on Scientific Computing* 26.4 (2005), pages 1389–1404.

[CMI09] A. Civril and M. Magdon-Ismail. "On selecting a maximum volume sub-matrix of a matrix and related problems". In: *Theoretical Computer Science* 410.47 (2009), pages 4801 –4811.

[CW09] K. L. Clarkson and D. P. Woodruff. "Numerical Linear Algebra in the Streaming Model". In: *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*. STOC '09. Bethesda, MD, USA: Association for Computing Machinery, 2009, pages 205–214. DOI: 10.1145/1536414.1536445.

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

[Don+17] K. Dong et al. "Scalable Log Determinants for Gaussian Process Kernel Learning". In: *Advances in Neural Information Processing Systems 30*. Edited by I. Guyon et al. Curran Associates, Inc., 2017, pages 6327–6337. URL: http://papers.nips.cc/paper/7212-scalable-log-determinants-for-gaussian-process-kernel-learning.pdf.

[DKM06a] P. Drineas, R. Kannan, and M. W. Mahoney. "Fast Monte Carlo algorithms for matrices I: Approximating matrix multiplication". In: *SIAM Journal on Computing* 36.1 (2006), pages 132–157.

[DKM06b] P. Drineas, R. Kannan, and M. W. Mahoney. "Fast Monte Carlo algorithms for matrices. II. Computing a low-rank approximation to a matrix". In: *SIAM J. Comput.* 36.1 (2006), pages 158–183. ISSN: 0097-5397. DOI: 10.1137/S0097539704442696.

[DKM06c] P. Drineas, R. Kannan, and M. W. Mahoney. "Fast Monte Carlo algorithms for matrices. III. Computing a compressed approximate matrix decomposition". In: *SIAM J. Comput.* 36.1 (2006), pages 184–206. ISSN: 0097-5397. DOI: 10.1137/S0097539704442702.

[Efr79a] B. Efron. "Bootstrap methods: another look at the jackknife". In: *Ann. Statist.* 7.1 (1979), pages 1–26. ISSN: 0090-5364.

[Efr79b] B. Efron. "Computers and the theory of statistics: thinking the unthinkable". In: *SIAM Rev.* 21.4 (1979), pages 460–480. DOI: 10.1137/1021092.

[ET93] B. Efron and R. J. Tibshirani. *An introduction to the bootstrap*. Volume 57. Monographs on Statistics and Applied Probability. Chapman and Hall, New York, 1993, pages xvi+436. DOI: 10.1007/978-1-4899-4541-9.

[Fit+18] J. K. Fitzsimons et al. "Improved stochastic trace estimators using mutually unbiased bases". In: *Uncertainty in Artificial Intelligence: Proceedings of the Thirty-Fourth Conference*. Edited by A. Globerson and R. Silva. Monterey, CA: AUAI Press, 2018.

[Fol01] G. B. Folland. "How to integrate a polynomial over a sphere". In: *Amer. Math. Monthly* 108.5 (2001), pages 446–448. DOI: 10.2307/2695802.

[FKV04] A. Frieze, R. Kannan, and S. Vempala. "Fast Monte-Carlo algorithms for finding low-rank approximations". In: *J. ACM* 51.6 (2004), pages 1025–1041. ISSN: 0004-5411. DOI: 10.1145/1039488.1039494.

[GSO17] A. S. Gambhir, A. Stathopoulos, and K. Orginos. "Deflation as a method of variance reduction for estimating the trace of a matrix inverse". In: *SIAM J. Sci. Comput.* 39.2 (2017), A532–A558. DOI: 10.1137/16M1066361.

[GPP16] M. Ghashami, D. J. Perry, and J. Phillips. "Streaming Kernel Principal Component Analysis". In: *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*. Edited by A. Gretton and C. C. Robert. Volume 51. Proceedings of Machine Learning Research. Cadiz, Spain: PMLR, 2016, pages 1365–1374. URL: http://proceedings.mlr.press/v51/ghashami16.html.

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

[GM10] G. H. Golub and G. Meurant. *Matrices, moments and quadrature with applications*. Princeton Series in Applied Mathematics. Princeton University Press, Princeton, NJ, 2010, pages xii+363.

[GVL13] G. H. Golub and C. F. Van Loan. *Matrix computations*. Fourth. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, Baltimore, MD, 2013, pages xiv+756.

[GW69] G. H. Golub and J. H. Welsch. "Calculation of Gauss quadrature rules". In: *Math. Comp. 23 (1969), 221-230; addendum, ibid.* 23.106, loose microfiche suppl (1969), A1–A10. DOI: 10.2307/2004418.

[GZT97] S. A. Goreinov, N. Zamarashkin, and E. E. Tyrtyshnikov. "Pseudo-skeleton approximations by matrices of maximal volume". In: *Mathematical Notes* 62 (1997), pages 515–519.

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

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

[GE96] M. Gu and S. C. Eisenstat. "Efficient Algorithms for Computing a Strong Rank-Revealing QR Factorization". In: *SIAM Journal on Scientific Computing* 17.4 (1996), pages 848–869.

[HCH12] E. Haber, M. Chung, and F. Herrmann. "An effective method for parameter estimation with PDE constraints with multiple right-hand sides". In: *SIAM J. Optim.* 22.3 (2012), pages 739–757. DOI: 10.1137/11081126X.

[HMT11] N. Halko, P. G. Martinsson, and J. A. Tropp. "Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions". In: *SIAM Rev.* 53.2 (2011), pages 217–288. ISSN: 0036-1445. DOI: 10.1137/090771806.

[HPIM12] S. Har-Peled, P. Indyk, and R. Motwani. "Approximate nearest neighbor: towards removing the curse of dimensionality". In: *Theory Comput.* 8 (2012), pages 321–350. DOI: 10.4086/toc.2012.v008a014.

[Har18] M. Hardt. *Convex optimization and approximation*. Course website, Berkeley EE 227C (Spring 2018). 2018. URL: https://ee227c.github.io.

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

[JL84] W. B. Johnson and J. Lindenstrauss. "Extensions of Lipschitz mappings into a Hilbert space". In: *Conference in modern analysis and probability (New Haven, Conn., 1982)*. Volume 26. Contemp. Math. Amer. Math. Soc., Providence, RI, 1984, pages 189–206. DOI: 10.1090/conm/026/737400.

[Kah66] W. Kahan. "Numerical Linear Algebra". In: *Canadian Mathematical Bulletin* 9.5 (1966), pages 757–801.

[KK12] P. Kar and H. Karnick. "Random Feature Maps for Dot Product Kernels". In: *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*. Edited by N. D. Lawrence and M. Girolami. Volume 22. Proceedings of Machine Learning Research. La Palma, Canary Islands: PMLR, 2012, pages 583–591. URL: http://proceedings.mlr.press/v22/kar12.html.

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

[KPS01] F. G. Kuruvilla, P. J. Park, and S. L. Schreiber. "Vector algebra in the analysis of genome-wide expression data". In: *Genome Biology* 3 (2001), research0011.1 –research0011.11.

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

[LP19] J. Lacotte and M. Pilanci. *Faster Least Squares Optimization*. 2019. eprint: arXiv:1911.02675.

[LAH11] T. van Leeuwen, A. Y. Aravkin, and F. Herrmann. "Seismic waveform inversion by stochastic optimization". In: *Intl. J. Geophysics* (2011). Article ID 689041. DOI: 10.1155/2011/689041.

[LNW14] Y. Li, H. L. Nguyen, and D. P. Woodruff. "Turnstile streaming algorithms might as well be linear sketches". In: *STOC'14—Proceedings of the 2014 ACM Symposium on Theory of Computing*. ACM, New York, 2014, pages 174–183.

[Lib+07] E. Liberty et al. "Randomized algorithms for the low-rank approximation of matrices". In: *Proceedings of the National Academy of Sciences* 104.51 (2007), pages 20167–20172.

[Lie73] E. H. Lieb. "Convex trace functions and the Wigner-Yanase-Dyson conjecture". In: *Advances in Math.* 11 (1973), pages 267–288. DOI: 10.1016/0001-8708(73)90011-X.

[LLR95] N. Linial, E. London, and Y. Rabinovich. "The geometry of graphs and some of its algorithmic applications". In: *Combinatorica* 15.2 (1995), pages 215–245. DOI: 10.1007/BF01200757.

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

[MD09] M. W. Mahoney and P. Drineas. "CUR matrix decompositions for improved data analysis". In: *Proceedings of the National Academy of Sciences* 106.3 (2009), pages 697–702.

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

[MT20] P.-G. Martinsson and J. Tropp. "Randomized Numerical Linear Algebra: Foundations & Algorithms". In: *Acta Numerica, 2020*. Cambridge Univ. Press, 2020, pages 403–572. arXiv: 2002.01387.

[Mir60] L. Mirsky. "Symmetric guage functions and unitarily invariant norms". In: *The Quarterly Journal of Mathematics* 11.1 (Jan. 1960), pages 50–59.

[MM17] C. Musco and C. Musco. "Recursive sampling for the nystrom method". In: *Advances in Neural Information Processing Systems*. 2017, pages 3833–3845.

[Nys+30] E. J. Nyström et al. "Über die praktische Auflösung von Integralgleichungen mit Anwendungen auf Randwertaufgaben". In: *Acta Mathematica* 54 (1930), pages 185–204.

[Olv+10] F. W. J. Olver et al., editors. *NIST handbook of mathematical functions*. U.S. Department of Commerce, National Institute of Standards and Technology, Washington, DC; Cambridge University Press, Cambridge, 2010, pages xvi+951.

[OT18] S. Oymak and J. A. Tropp. "Universality laws for randomized dimension reduction, with applications". In: *Inf. Inference* 7.3 (2018), pages 337–446. DOI: 10.1093/imaiai/iax011.

[Pai82] M. A. Paige Christopher C. and. "LSQR: An Algorithm for Sparse Linear Equations and Sparse Least Squares". In: *ACM Trans. Math. Softw.* 8.1 (Mar. 1982), pages 43–71. DOI: 10.1145/355984.355989.

[Pan00] C.-T. Pan. "On the existence and computation of rank-revealing LU factorizations". In: *Linear Algebra and its Applications* 316.1 (2000). Special Issue: Conference celebrating the 60th birthday of Robert J. Plemmons, pages 199 –222.

[Pap+00] C. H. Papadimitriou et al. "Latent semantic indexing: a probabilistic analysis". In: volume 61. 2. Special issue on the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (Seattle, WA, 1998). 2000, pages 217–235. DOI: 10.1006/jcss.2000.1711.

[Par98] B. N. Parlett. *The symmetric eigenvalue problem*. Volume 20. Classics in Applied Mathematics. Corrected reprint of the 1980 original. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1998, pages xxiv+398. DOI: 10.1137/1.9781611971163.

[PW16] M. Pilanci and M. J. Wainwright. "Iterative Hessian Sketch: Fast and Accurate Solution Approximation for Constrained Least-Squares". In: *J. Mach. Learn. Res.* 17.1 (Jan. 2016), pages 1842–1879.

[PW17] M. Pilanci and M. J. Wainwright. "Newton Sketch: A Near Linear-Time Optimization Algorithm with Linear-Quadratic Convergence". In: *SIAM Journal on Optimization* 27.1 (Jan. 2017), pages 205–245. DOI: 10.1137/15m1021106.

[Pis80] G. Pisier. "Remarques sur un résultat non publié de B. Maurey". fr. In: *Séminaire d'Analyse fonctionnelle (dit "Maurey-Schwartz")* (1980-1981). talk:5. URL: http://www.numdam.org/item/SAF_1980-1981____A5_0/.

[RT08] V. Rokhlin and M. Tygert. "A fast randomized algorithm for overdetermined linear least-squares regression". In: *Proceedings of the National Academy of Sciences* 105.36 (Sept. 2008), pages 13212–13217. DOI: 10.1073/pnas.0804869105.

[Rud+18] A. Rudi et al. "On fast leverage score sampling and optimal learning". In: *Advances in Neural Information Processing Systems*. 2018, pages 5672–5682.

[Sar06] T. Sarlós. "Improved Approximation Algorithms for Large Matrices via Random Projections". In: *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*. 2006, pages 143–152. DOI: 10.1109/FOCS.2006.37.

[SS01] B. Scholkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT press, 2001.

[She94] J. R. Shewchuk. *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*. Technical report. USA, 1994.

[SAR17] M. Simchowitz, A. E. Alaoui, and B. Recht. "On the Gap Between Strict-Saddles and True Convexity: An Omega(log d) Lower Bound for Eigenvector Approximation". In: *CoRR* abs/1704.04548 (2017). arXiv: 1704.04548.

[SV09] T. Strohmer and R. Vershynin. "A randomized Kaczmarz algorithm with exponential convergence". In: *J. Fourier Anal. Appl.* 15.2 (2009), pages 262–278. DOI: 10.1007/s00041-008-9030-4.

[TOH14] C. Thrampoulidis, S. Oymak, and B. Hassibi. "The Gaussian min-max theorem in the presence of convexity". In: *arXiv preprint arXiv:1408.4837* (2014).

[TB97] L. N. Trefethen and D. Bau III. *Numerical linear algebra*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1997, pages xii+361. DOI: 10.1137/1.9780898719574.

[Tro19] J. A. Tropp. *Matrix concentration and computational linear algebra*. CMS Lecture Notes 2019-01. Pasadena, CA: Caltech, 2019.

[Tro12] J. A. Tropp. "User-friendly tail bounds for sums of random matrices". In: *Found. Comput. Math.* 12.4 (2012), pages 389–434. DOI: 10.1007/s10208-011-9099-z.

[Tro15] J. A. Tropp. "An Introduction to Matrix Concentration Inequalities". In: *Foundations and Trends® in Machine Learning* 8.1-2 (2015), pages 1–230. ISSN: 1935-8237. DOI: 10.1561/2200000048.

[Tro16] J. A. Tropp. "The expected norm of a sum of independent random matrices: an elementary approach". In: *High dimensional probability VII*. Volume 71. Progr. Probab. Springer, [Cham], 2016, pages 173–202. DOI: 10.1007/978-3-319-40519-3_8.

[Tro+17a] J. A. Tropp et al. "Fixed-Rank Approximation of a Positive-Semidefinite Matrix from Streaming Data". In: *Advances in Neural Information Processing Systems 30*. Edited by I. Guyon et al. Curran Associates, Inc., 2017, pages 1225–1234. URL: http://papers.nips.cc/paper/6722-fixed-rank-approximation-of-a-positive-semidefinite-matrix-from-streaming-data.pdf.

[Tro+17b] J. A. Tropp et al. "Practical sketching algorithms for low-rank matrix approximation". In: *SIAM J. Matrix Anal. Appl.* 38.4 (2017), pages 1454–1485. DOI: 10.1137/17M1111590.

[UCS17] S. Ubaru, J. Chen, and Y. Saad. "Fast estimation of $\mathtt{tr}(f(A))$ via stochastic Lanczos quadrature". In: *SIAM J. Matrix Anal. Appl.* 38.4 (2017), pages 1075–1099. DOI: 10.1137/16M1104974.

[Upa16] J. Upadhyay. "Fast and space-optimal low-rank factorization in the streaming model with application in differential privacy". In: *arXiv preprint arXiv:1604.01429* (2016).

[Ver18] R. Vershynin. *High-dimensional probability*. Volume 47. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, Cambridge, 2018, pages xiv+284. DOI: 10.1017/9781108231596.

[VM17] S. Voronin and P.-G. Martinsson. "Efficient Algorithms for CUR and Interpolative Matrix Decompositions". In: *Adv. Comput. Math.* 43.3 (June 2017), pages 495–516.

[WS01] C. K. Williams and M. Seeger. "Using the Nyström method to speed up kernel machines". In: *Advances in neural information processing systems*. 2001, pages 682–688.

[Woo14] D. P. Woodruff. "Sketching as a Tool for Numerical Linear Algebra". In: *Foundations and Trends in Theoretical Computer Science* 10.1-2 (2014), pages 1 –157. DOI: 10.1561/0400000060.

[Woo+08] F. Woolfe et al. "A fast randomized algorithm for the approximation of matrices". In: *Appl. Comput. Harmon. Anal.* 25.3 (2008), pages 335–366. DOI: 10.1016/j.acha.2007.12.002.