On Calculations Involving Perturbed Orthogonal Matrices

FOREST KOBAYASHI *[1] AND EVAN LIANG[†1,2]

[1]Department of Mathematics, Harvey Mudd College
[2]Department of Computer Science, Harvey Mudd College

December 18, 2018

Abstract

Many common algorithms employed in Data Science require the calculation of an Orthonormal Basis. Often, this is done using the Gram-Schmidt Algorithm, which has time complexity $O(n^3)$ in the standard case. For low-dimensional spaces, this does not pose a significant computational burden. However, for high-dimensional vector spaces, calculating such a basis can prove unfeasible, so it could be desirable to find faster algorithms that approximate Orthonormal Bases up to some error tolerance ε. In this paper, we examine bounds on error propagation if such an "ε-almost orthonormal basis" (defined below) is employed.

1. INTRODUCTION

1.1. MOTIVATION

Orthogonal matrices (and their complex counterparts Unitary matrices) are of vital importance in linear algebra, both pure and applied. In a sense, they are the least "disruptive" class of deformations we can apply to a vector space without fixing all of its elements — every linear isometry operator on a finite-dimensional inner product space can be represented by an orthogonal matrix, and conversely, every orthogonal matrix represents an isometry encoded with respect to some particular basis. Thus, every orthogonal matrix Q is necessarily invertible, with $Q^{-1} = Q^{\mathsf{T}}$. This is very nice property, for both theoretical and computational purposes.

From the theoretical perspective, this trivially guarantees the existence of an inverse, and from $1 = \det(I) = \det(QQ^{-1}) = \det(Q)\det(Q^{\mathsf{T}}) = \det(Q)^2$, we get that $\det(Q) = \pm 1$.[1] Additionally, because $Q^{-1}Q = I = QQ^{-1}$, orthogonal matrices encode normal operators, and the complex spectral theorem they are thus unitarily diagonalizable [Axl15]. In fact, we can guarantee an even stronger property — if Q is orthogonal, then the spectrum $\sigma(Q) \subseteq \{z \in \mathbb{C} \mid |z| = 1\}$. That is, every eigenvalue of Q lies on the unit circle in $\mathbb{C}$. As such, the group of $n \times n$ orthogonal matrices (denoted $\mathrm{O}(n)$) and its subgroups are of great interest from a theoretical perspective.

Orthogonal matrices also have many nice computational properties. Again, let Q be an orthogonal matrix. Then because all eigenvalues have unit modulus, orthogonal matrices cannot magnify errors in computations [TB97, p. 95]. Furthermore, because $Q^{-1} = Q^{\mathsf{T}}$, inverting Q is essentially trivial, as we can simply swap indexing schemes to yield a semantically faithful representation of Q^{-1}, without having to perform any major calculations. Finally, Orthogonal matrices are of great importance because of their occurrence in important matrix factorizations, such as QR decomposition.

At the time that this paper is being written, there is a large body of ongoing research in numerical linear algebra surrounding "sketching" methods — that is, fast, randomized algorithms for querying large matrices in a way that preserves relevant structure. One classic result in this vein is the *Johnson-Lindenstrauss Lemma* (hence referred to as the JL Lemma), which proves a bounds about the extent to which a family of randomized dimension-reduction algorithms can preserve metric structure. Such tools are necessitated by the ever-growing importance of Big Data and Big Data Analytics techniques in the tech industry.

For our project, we will examine how error resulting from a JL-like dimension reduction scheme can propagate through further computations. To do so, we will re-frame the problem in terms of examining how properties of orthogonal matrices are affected when we apply a small random perturbation matrix. We will examine both additive and multiplicative perturbations, and describe some possible connections to eigenvalue estimation problems.

But first, we will give an overview of any non-standard definitions and notation that we will make use of in our paper. As our randomized algorithms will make some use of concepts from Random Matrix Theory, we will also include an overview of some of the

*fkobayashi@g.hmc.edu
†eliang@g.hmc.edu

[1]note that the converse is not true — i.e., for some $n \times n$ matrix M with $\det(M) = \pm 1$, it is not necessarily true that M is orthogonal

relevant concepts in the space below.

1.2. Definitions and Notations

First, some general things: to make things easier for the reader to scan through our paper quickly, we will use $\triangle$ to denote the end of a theorem, definition, remark, or similar, and $\blacksquare$ to denote end of a proof. In general, bases for spaces will be given by $\mathcal{B}$, while the basis vectors themselves will be denoted $\mathbf{e}_i$. δ_{ij} will, as usual, refer to the Kronecker delta. In general, we will use the variables i, j, k and ℓ as indices variables, resorting to Greek symbols only if necessary. Unless otherwise stated, $\mathcal{U}, \mathcal{V}$ and $\mathcal{W}$ will refer to finite-dimensional inner product spaces. All other notation (except that described below) should be standard.

Definition 1.1 (Random Matrix Family). We'll use some nonstandard notation for random matrices in our paper. Let $\mathcal{A}$ be a probability distribution with parameters $x_1, \ldots, x_n$, and let $M \in M_{m\times n}(\mathbb{R})$. Then we say $M \sim \mathcal{A}(\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n)$ iff

$$m_{ij} \sim \mathcal{A}((\boldsymbol{X}_1)_{ij}, \ldots, (\boldsymbol{X}_n)_{ij}).$$

for all i, j. $\triangle$

Example 1.1.1 (Gaussian random matrix). Let $\boldsymbol{\mu} \in M_{m\times n}(\mathbb{R})$, and let $\boldsymbol{\sigma} \in M_{m\times n}(\mathbb{R}^{\geq 0})$. Then we say $\boldsymbol{\xi} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma})$ iff

$$\xi_{ij} \sim \mathcal{N}(\mu_{ij}, \sigma_{ij})$$

for all i, j. We will sometimes denote this by

$$\begin{bmatrix} \xi_{1,1} & \cdots & \xi_{1,n} \\ \vdots & \ddots & \vdots \\ \xi_{m,1} & \cdots & \xi_{m,n} \end{bmatrix} \sim \begin{bmatrix} \mathcal{N}(\mu_{1,1}, \sigma_{1,1}) & \cdots & \mathcal{N}(\mu_{1,n}, \sigma_{1,n}) \\ \vdots & \ddots & \vdots \\ \mathcal{N}(\mu_{m,1}, \sigma_{m,1}) & \cdots & \mathcal{N}(\mu_{m,n}, \sigma_{m,n}) \end{bmatrix}$$

if it is more convenient to do so. $\triangle$

Oftentimes, we'll be interested in matrices where all of the entries are i.i.d. Hence, we'll introduce the following notation:

Definition 1.2. Let $c \in \mathbb{R}$. Then we define

$$c_{m\times n} = c \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix} = \begin{bmatrix} c & \cdots & c \\ \vdots & \ddots & \vdots \\ c & \cdots & c \end{bmatrix}$$

Thus, we can define a $m \times n$ matrix of i.i.d. Gaussian random variables by $M \sim \mathcal{N}(\mu_{m\times n}, \sigma_{m\times n})$ for some $\mu, \sigma \in \mathbb{R}$ (where $\sigma \geq 0$). $\triangle$

Definition 1.3. Let $M \in M_{m\times n}(\mathbb{R})$. If we have $M_{ij} < \varepsilon$ for all i, j, then we write

$$M \prec \varepsilon$$

and say M is *an ε-bounded matrix.* We denote the set of all matrices M such that $-\varepsilon \prec M \prec \varepsilon$ by $B_\varepsilon(0_{m\times n})$, because under the metric induced by the max norm on $M_{m\times n}(\mathbb{R})$, the set of ε-bounded matrices is simply the open ball about the matrix $0_{m\times n}$.

In the case that we want $M_{ij} \leq \varepsilon$, we of course replace $\prec$ with a $\preceq$ symbol, and denote the set of all such matrices by $\overline{B_\varepsilon(0_{m\times n})}$. $\triangle$

Definition 1.4 (Orthogonal Group). The standard notation for the group of orthogonal matrices on $\mathbb{R}^n$ is $\mathrm{O}(n)$. However, this looks confusingly similar to the big-O notation $O(n)$. Hence, we will elect to denote the orthogonal group on $\mathbb{R}^n$ by $\mathrm{Orth}(n)$. $\triangle$

Definition 1.5. Let $n \in \mathbb{N}$, and let $M \in M_{n\times n}(\mathbb{R})$. Then if M can be expressed as the sum of an an orthogonal matrix Q and a matrix ξ with $-\varepsilon \prec \xi \prec \varepsilon$, then we call M a ε-almost orthogonal matrix. $\triangle$

Definition 1.6. Let $\mathcal{V}$ be a finite-dimensional inner product space. Let $\mathcal{B} = \{\mathbf{e}_1, \ldots, \mathbf{e}_n\}$ be a basis for $\mathcal{V}$, and suppose that $\forall \mathbf{e}_i, \mathbf{e}_j \in \mathcal{B}$ with $i \neq j$, we have

$$|\langle \mathbf{e}_i, \mathbf{e}_j \rangle| < \varepsilon.$$

and $\langle \mathbf{e}_i, \mathbf{e}_i \rangle < 1 + \varepsilon$ for every $\mathbf{e}_i$. Then we call call $\mathcal{B}$ an *ε-almost orthonormal basis* of V. Note that it will often be useful to restate the conditions above as $\forall \mathbf{e}_i, \mathbf{e}_j \in \mathcal{B}$,

$$|\langle \mathbf{e}_i, \mathbf{e}_j \rangle - \delta_{ij}| < \varepsilon$$

which will be particularly relevant when dealing with double summations. $\triangle$

To contrast orthonormal bases with ε-almost orthonormal bases, we will often use $\mathcal{B}_\perp$ to denote an orthonormal basis, and $\mathcal{B}_\varepsilon$ to denote an ε almost-orthonormal basis.

This should handle any outstanding notational quirks, so we will continue to the main body.

2. Results

Since we'll be building on results from the midterm project, we'll first reacquaint ourselves with the material therein. To avoid complete redundancy of content, we will take the section of this paper as opportunity to improve the quality and clarity of our exposition. In particular, we will add much greater justification and explanation for our proofs, as many were opaque and generally hard to read. Our analysis will be broken into two main parts: first, dealing with ε-almost orthonormal bases, and then later, dealing with ε-almost orthogonal matrices. These two topics are of course related, as we will detail later. The bulk of our interest will be on the latter, but we will include results about the former as well so as to show two different perspectives on the problem.

2.1. ε-almost orthonormal bases

Let $\mathcal{V}$ be an n-dimensional inner product space, and let $\mathcal{B}_\perp$ be an orthonormal basis for $\mathcal{V}$. One nice property of $\mathcal{B}_\perp$ is that $\forall x \in \mathcal{V}$, we have the representation

$$x = \sum_{i=1}^{n} \langle x, \mathbf{e}_i \rangle \mathbf{e}_i$$

and hence

$$\|x\|_2^2 = \sum_{i=1}^{n} |\langle x, \mathbf{e}_i \rangle|^2.$$

To help us build intuition for how it "feels" to work with an ε-almost orthonormal basis instead of an orthonormal basis, we will first prove a result about the extent to which the results above fail.

THEOREM 2.1. *Let $\mathcal{V}$ be finite-dimensional inner product space, and $n = \dim(\mathcal{V})$. Let $\varepsilon > 0$ be given, with the constraint that $\varepsilon < \frac{1}{n}$, and let $\mathcal{B}_\varepsilon$ be an ε-almost orthonormal basis of $\mathcal{V}$. Now, let $x \in \mathcal{V}$, and suppose that x has the representation*

$$x = \sum_{i=1}^{n} c_i \langle x, \mathbf{e}_i \rangle$$

where $c_i \in \mathbb{R}$, and $\mathbf{e}_i \in \mathcal{B}_\varepsilon$. Then we have

$$\sum_{i=1}^{n} |c_i| \le \frac{\|x\|_2}{\sqrt{\frac{1}{n} - \varepsilon}}$$

△

Proof. First, we will prove a lower bound for $\|x\|_2$ in terms of the c_i, and use this to obtain the desired result. Observe that by the triangle inequality, we have

$$\left|\sum_{i=1}^{n} c_i^2\right| - \left|\langle x, x \rangle - \sum_{i=1}^{n} c_i^2\right| \le |\langle x, x \rangle| = \|x\|_2^2.$$

Removing the superfluous absolute values, this is just

$$\sum_{i=1}^{n} c_i^2 - \left|\langle x, x \rangle - \sum_{i=1}^{n} c_i^2\right| \le \|x\|_2^2.$$

We seek a bound for the left-hand-side in terms of of the c_i. We'll work with the second term. Note that

$$\langle x, x \rangle = \sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j \langle \mathbf{e}_i, \mathbf{e}_j \rangle.$$

Hence we have

$$\begin{aligned}
\left|\langle x, x \rangle - \sum_{i=1}^{n} c_i^2\right| &= \left|\sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j (\langle \mathbf{e}_i, \mathbf{e}_j \rangle - \delta_{ij})\right| \\
&\le \sum_{i=1}^{n} \sum_{j=1}^{n} |c_i c_j (\langle \mathbf{e}_i, \mathbf{e}_j \rangle - \delta_{ij})| \\
&= \sum_{i=1}^{n} \sum_{j=1}^{n} |c_i c_j| |(\langle \mathbf{e}_i, \mathbf{e}_j \rangle - \delta_{ij})|
\end{aligned}$$

Recall that $\mathcal{B}_\varepsilon$ is an ε-almost orthonormal basis implies that $\forall \mathbf{e}_i, \mathbf{e}_j \in \mathcal{B}_\varepsilon$, we have $|\langle \mathbf{e}_i, \mathbf{e}_j \rangle - \delta_{ij}| < \varepsilon$, and hence we have

$$\begin{aligned}
&\le \sum_{i=1}^{n} \sum_{j=1}^{n} |c_i c_j| \varepsilon \\
&= \varepsilon \sum_{i=1}^{n} \sum_{j=1}^{n} |c_i c_j| \\
&= \varepsilon \left(\sum_{i=1}^{n} |c_i|\right)^2.
\end{aligned}$$

This gives us the inequality

$$\sum_{i=1}^{n} c_i^2 - \varepsilon \left(\sum_{i=1}^{n} |c_i|\right)^2 \le \sum_{i=1}^{n} c_i^2 - \left|\langle x, x \rangle - \sum_{i=1}^{n} c_i^2\right|, \quad (1)$$

and so we can chain together with our original bound to yield

$$\left(\sum_{i=1}^{n} c_i^2\right) - \varepsilon \left(\sum_{i=1}^{n} |c_i|\right)^2 \le \|x\|_2^2$$

By Hölder's inequality for sums, we have

$$\begin{aligned}
\left(\sum_{i=1}^{n} |c_i|\right)^2 &= \left(\sum_{i=1}^{n} c_i \operatorname{sgn}(c_i)\right)^2 \\
&\le \left(\sum_{i=1}^{n} c_i^2\right) \left(\sum_{i=1}^{n} \operatorname{sgn}(c_i)^2\right) \\
&= \left(\sum_{i=1}^{n} c_i^2\right) \cdot n
\end{aligned}$$

whence

$$\frac{1}{n} \left(\sum_{i=1}^{n} |c_i|\right)^2 \le \sum_{i=1}^{n} c_i^2$$

and so we can chain together with (1) to obtain

$$\frac{1}{n} \left(\sum_{i=1}^{n} |c_i|\right)^2 - \varepsilon \left(\sum_{i=1}^{n} |c_i|\right)^2 \le \|x\|_2^2$$

From which we obtain

$$\begin{aligned}
\left(\sum_{i=1}^{n} |c_i|\right)^2 &\le \frac{\|x\|_2^2}{\frac{1}{n} - \varepsilon} \\
\sum_{i=1}^{n} |c_i| &\le \frac{\|x\|_2}{\sqrt{\frac{1}{n} - \varepsilon}}
\end{aligned}$$

as desired. ■

As a corollary, we have the the following result:

COROLLARY 2.1.1. *With all variables quantified as above, we have*

$$|\langle x, \mathbf{e}_i\rangle - c_i| \leq \frac{\varepsilon\|x\|_2^2}{\sqrt{\frac{1}{n}-\varepsilon}}.$$

for every $i = 1, \ldots, n$. △

Proof. Observe that

$$\langle x, \mathbf{e}_i\rangle = \left\langle \sum_{j=1}^n c_j \mathbf{e}_j,\ \mathbf{e}_i \right\rangle$$

Thus we have

$$\begin{aligned}
|\langle x, \mathbf{e}_i\rangle - c_i| &= \left| \left\langle \sum_{j=1}^n c_j \mathbf{e}_j,\ \mathbf{e}_i \right\rangle - c_i \right| \\
&= \left| \left(\sum_{j=1}^n c_j \langle \mathbf{e}_j, \mathbf{e}_i\rangle \right) - c_i \right| \\
&= \left| \sum_{j=1}^n c_j (\langle \mathbf{e}_j, \mathbf{e}_i\rangle - \delta_{ij}) \right| \\
&\leq \varepsilon \left| \sum_{j=1}^n c_j \right| \\
&\leq \varepsilon \sum_{j=1}^n |c_j| \\
&\leq \frac{\varepsilon\|x\|_2^2}{\sqrt{\frac{1}{n}-\varepsilon}}
\end{aligned}$$

as desired. ■

REMARK. Note that if $\varepsilon \ll \frac{1}{n}$, then the error is essentially $O(\varepsilon)$.

2.2. ε-ALMOST ORTHOGONAL MATRICES

We now turn our attention to ε-almost orthogonal matrices. We begin by showing the connection to ε-almost orthonormal bases, and then introducing the problem that will be our main focus for this paper.

2.2.1. INTRODUCTION

THEOREM 2.2. *Let $\mathcal{V}$ be a a finite-dimensional inner product space with* $\dim(\mathcal{V}) = n$. *Let* $Q \in \mathrm{Orth}(n)$, *and* $\xi \in B_\varepsilon(0_{n\times n})$. *Then let* $M = Q + \xi$, *i.e.:*

$$M = \begin{bmatrix} q_{11} + \xi_{11} & \cdots & q_{1n} + \xi_{1n} \\ \vdots & \ddots & \vdots \\ q_{n1} + \xi_{n1} & \cdots & q_{nn} + \xi_{nn} \end{bmatrix}$$

Let $\mathbf{v}_i, \mathbf{v}_j$ be the i^{th} and j^{th} columns of M, respectively. Then

$$\langle \mathbf{v}_i \mathbf{v}_j\rangle \leq 2\sqrt{n}\varepsilon + n\varepsilon^2.$$

Note that this bound is not tight. In particular, the coefficient of $2\sqrt{n}$ on the ε term is somewhat crude and can be lowered. △

Proof. We have

$$\begin{aligned}
\langle \mathbf{v}_i, \mathbf{v}_j\rangle &= \sum_{k=1}^n (q_{ki} + \xi_{ki})(q_{kj} + \xi_{kj})\langle \mathbf{e}_k, \mathbf{e}_k\rangle \\
&= \sum_{k=1}^n q_{ki}q_{kj} + \xi_{ki}q_{kj} + q_{kj}\xi_{ki} + \xi_{ki}\xi_{kj} \\
&= \cancelto{0}{\langle \mathbf{q}_i, \mathbf{q}_j\rangle} + \sum_{k=1}^n \xi_{ki}q_{kj} + q_{kj}\xi_{ki} + \xi_{ki}\xi_{kj} \\
&\leq \sum_{k=1}^n \varepsilon(q_{kj} + q_{ki}) + \varepsilon^2 \\
&= \varepsilon\left(\sum_{k=1}^n (q_{kj} + q_{ki}) \right) + n\varepsilon^2 \\
&\leq \varepsilon\left(2\sum_{k=1}^n q_{kj} \right) + n\varepsilon^2 \\
&\leq \varepsilon\left(2\sum_{k=1}^n \frac{1}{\sqrt{n}} \right) + n\varepsilon^2 \qquad (*) \\
&= 2\sqrt{n}\varepsilon + n\varepsilon^2
\end{aligned}$$

As desired. Note that to obtain $(*)$, we do not require that $q_{kj} \leq \frac{1}{\sqrt{n}}$ for each k — we are bounding the sum, not the individual terms. ■

From the theorem, we see that the idea of an ε-almost orthogonal matrix is related to that of an ε-almost orthonormal basis, in that the columns of the ε-almost orthogonal matrix are an $O(\varepsilon)$ almost-orthonormal basis.

THEOREM 2.3. *Let $\varepsilon > 0$ be given, and let* $\xi \in B_\varepsilon(0_{n\times n})$. *Then* $\|\xi\|_F \leq \varepsilon n$. △

Proof. Observe that

$$\begin{aligned}
\|\xi\|_F &= \sqrt{\sum_{i,j\leq n} |\xi_{ij}|^2} \\
&\leq \sqrt{\sum_{i,j\leq n} \varepsilon^2} \\
&= \sqrt{\varepsilon^2 n^2} \\
&= \varepsilon n
\end{aligned}$$

as desired. ■

Now, we'll introduce the problem that will be the main focus of our paper today. An important property of orthogonal matrices is that they can be used to perform fast matrix exponentiation for symmetric matrices. That is, let M be a real symmetric matrix. Then by the spectral theorem, M can be diagonalized by $A\Lambda A^\mathsf{T}$, where A is an orthogonal matrix. Using this, we can perform rapid exponentiation of M by utilizing the fact $M^n = A\Lambda^n A^\mathsf{T}$, which can be computed quite rapidly. However, we can imagine situations in which we have access to an *almost* orthogonal matrix in some matrix exponentiation problem, but not an *orthogonal* matrix. For instance, imagine we have an IMU measuring the body frame of some machine, say, a UAV, encoded as a matrix B. Say there is measurement error $\xi \sim N(\mathbf{0}, \varepsilon_{m\times n})$ in the hardware, such that B is now an *ε-Almost Orthogonal Matrix*. In this situation, what kinds of errors can we expect in the exponentiation process if we treat B like it's orthogonal? That is, if we assume

$$(BDB^\mathsf{T})^n = BD^nB^\mathsf{T}$$

What kinds of errors will this introduce into our calculations? We examine this question below, and prove asymptotic bounds for the errors we might see.

2.2.2. Deterministic Matrices

We will begin by examining the problem in the restricted context of non-random matrices, as the proofs are much simpler. Here, we will be primarily interested in two error metrics, which we call δ and Δ, as defined below.

DEFINITION 2.1 (Error Metrics for Diagonalization). Let $\mathcal{V}$ be a finite-dimensional inner product space of dimension $n \in \mathbb{N}$, and let $\varepsilon > 0$ be given. Let $Q \in \mathrm{Orth}(n)$ and $\xi \in B_\varepsilon(0_{n\times n})$, and let $M = A + \xi$. Then we define our Diagonalization error metrics by

$$\boxed{\delta(k) = \left\|QD^kQ^\mathsf{T} - (MDM^\mathsf{T})^k\right\|_F^2}$$

and

$$\boxed{\Delta(k) = \left\|MD^kM^\mathsf{T} - (MDM^\mathsf{T})^k\right\|_F^2.}$$

We will refer to the stuff inside the norm in the expression for δ as $\mathbf{v}_\delta$, and the stuff inside the norm in the expression for Δ as $\mathbf{v}_\Delta$. △

Before we begin working with these error metrics, it will be useful to prove some technical lemmas to aid in proving our first theorem.

LEMMA 2.4. *Let Λ, D be diagonal $n \times n$ matrices, and let H be an orthogonal matrix. Then*

$$\mathrm{tr}(HD^kH^\mathsf{T}\Lambda^k) = \sum_{i,j}^n h_{ij}^2 \lambda_i^k(D)\lambda_j^k(\Lambda)$$

△

Proof. We'll start from the left-hand-side, first calculating HD^kH^T. Note that indexing variables might get a little wacky here, because we have a lot of matrices in the expression. Observe that, by the standard matrix multiplication formula $c_{ij} = \sum_{k=1}^n a_{ik}b_{kj}$, we have

$$\begin{aligned}(D^kH^\mathsf{T})_{ij} &= \sum_{\alpha=1}^n d_{i\alpha}h_{j\alpha} \\ &= \sum_{\alpha=1}^n \lambda_i^k(D)h_{j\alpha}\delta_{i\alpha}\end{aligned}$$

Where $\delta_{i\alpha}$ is the kronecker delta. Thus, applying this rule a second time, we have

$$\begin{aligned}(H(D^kH^\mathsf{T}))_{ij} &= \sum_{\beta=1}^n (H)_{i\beta}(D^kH^\mathsf{T})_{\beta j} \\ &= \sum_{\beta=1}^n h_{i\beta}\sum_{\alpha=1}^n \lambda_\beta^k(D)h_{j\alpha}\delta_{\beta\alpha}\end{aligned}$$

And so finally,

$$\begin{aligned}(HD^kH^\mathsf{T}\Lambda^k)_{ij} &= \sum_{\gamma=1}^n (HD^kH^\mathsf{T})_{i\gamma}(\Lambda^k)_{\gamma j} \\ &= \sum_{\gamma=1}^n \left(\sum_{\beta=1}^n h_{i\beta}\sum_{\alpha=1}^n \lambda_\beta^k(D)h_{\gamma\alpha}\delta_{\beta\alpha}\right)\lambda_\gamma^k(\Lambda)\delta_{\gamma j} \\ &= \sum_{\gamma=1}^n\sum_{\beta=1}^n\sum_{\alpha=1}^n h_{i\beta}h_{\gamma\alpha}\lambda_\beta^k(D)\lambda_\gamma^k(\Lambda)\delta_{\beta\alpha}\delta_{\gamma j}\end{aligned}$$

Thus, the trace is given by

$$\begin{aligned}\mathrm{tr}(HD^kH^\mathsf{T}\Lambda^k) &= \sum_{\ell=1}^n (HD^kH^\mathsf{T}\Lambda^k)_{\ell\ell} \\ &= \sum_{\ell=1}^n\sum_{\gamma=1}^n\sum_{\beta=1}^n\sum_{\alpha=1}^n h_{\ell\beta}h_{\gamma\alpha}\lambda_\beta^k(D)\lambda_\gamma^k(\Lambda)\delta_{\beta\alpha}\delta_{\gamma\ell} \\ &= \sum_{\ell=1}^n\sum_{m=1}^n h_{\ell m}h_{\ell m}\lambda_m^k(D)\lambda_\ell^k(\Lambda) \\ &= \sum_{\ell=1}^n\sum_{m=1}^n h_{\ell m}^2\lambda_m^k(D)\lambda_\ell^k(\Lambda)\end{aligned}$$

as desired. ■

COROLLARY 2.4.1. *Let Λ, D, H be as defined above. Then $\mathrm{tr}(\Lambda^kHD^kH^\mathsf{T}) = \mathrm{tr}(HD^kH^\mathsf{T}\Lambda^k)$.* △

Proof. This follows simply from the fact that trace is invariant under cyclic permutations. ■

Now, we proceed to the first of our main claims.

THEOREM 2.5. *Let $\mathcal{V}$ be a finite-dimensional inner product space with $\dim \mathcal{V} = n \in \mathbb{N}$, and let $\varepsilon > 0$ be given. Let $Q \in \operatorname{Orth}(n)$ and $\xi \in B_\varepsilon(0_{n\times n})$, and let $M = Q+\xi$. Now, let D be an $n\times n$ diagonal matrix, and let $\sigma_0 = \max\{|\sigma_i(D)|\}$, and $\tau_0 = \max\{|\sigma_j(MDM^\mathsf{T})|\}$ be the largest singular values of D and MDM^T, respectively. Then if $\nu = \max\{\sigma_0, \tau_0\}$, then*

$$\delta(k) \sim \alpha\nu^{2k}$$

for almost all cases, where α is a constant that depends on D and MDM^T. We list edge cases as follows:

(i) If the eigenvalues of D and MDM^T are equal, then $\delta(k) = 0$.

(ii) If the singular values of D and MDM^T are equal but some of the eigenvalues are not (i.e., the eigenvalues have the same magnitudes but some have different signs), then $\delta(k) = 0$ for even k and $\delta(k) = \alpha c^{2k}$ for odd k.

(iii) If the multiplicity of ν in $\sigma(D)$, $\sigma(\Lambda)$ is the same, then whenever k is even, $\delta(k) = o(\nu^{2k})$ (i.e., δ is subexponential in k).

△

Proof. Note that MDM^T is symmetric:

$$\begin{aligned}(MDM^\mathsf{T}) &= (M^\mathsf{T})^\mathsf{T} D^\mathsf{T} M^\mathsf{T}\\ &= MDM^\mathsf{T}\end{aligned}$$

Thus, by the real spectral theorem, MDM^T is orthogonally diagonalizable, and so there exists an orthogonal matrix P and a diagonal matrix Λ such that $MDM^\mathsf{T} = P\Lambda P^\mathsf{T}$. Hence,

$$\begin{aligned}\delta(k) &= \left\|QD^kQ^\mathsf{T} - (MDM^\mathsf{T})^k\right\|_F^2\\ &= \left\|QD^kQ^\mathsf{T} - (P\Lambda P^\mathsf{T})^k\right\|_F^2\\ &= \left\|QD^kQ^\mathsf{T} - P\Lambda^k P^\mathsf{T}\right\|_F^2\end{aligned}$$

Since the Frobenius Norm is unitary invariant, it follows that

$$\begin{aligned}&= \left\|P^T\left(QD^kQ^\mathsf{T} - P\Lambda^k P^\mathsf{T}\right)P\right\|_F^2\\ &= \left\|P^\mathsf{T}QD^kQ^\mathsf{T}P - \Lambda^k\right\|_F^2\\ &= \operatorname{tr}\Big(\big(P^\mathsf{T}QD^kQ^\mathsf{T}P - \Lambda^k\big)^2\Big)\end{aligned}$$

For the sake of concision, let $H = P^\mathsf{T}Q$. Then we can write this as

$$\begin{aligned}\delta(k) &= \operatorname{tr}\Big(\big(HD^kH^\mathsf{T} - \Lambda^k\big)^2\Big)\\ &= \operatorname{tr}\big(HD^{2k}H^\mathsf{T} - HD^kH^\mathsf{T}\Lambda^k - \Lambda^kHD^kH^\mathsf{T} + \Lambda^{2k}\big)\end{aligned}$$

by the Lemma and its Corollary, we have

$$\begin{aligned}&= \operatorname{tr}\big(HD^{2k}H^\mathsf{T}\big) + \operatorname{tr}\big(\Lambda^{2k}\big) - 2\operatorname{tr}\big(HD^kH^\mathsf{T}\Lambda^k\big)\\ &= \left(\sum_{i=1}^n \lambda_i^{2k}(D) + \lambda_i^{2k}(\Lambda)\right) - 2\sum_{i,j\le n} h_{ij}^2\lambda_i^k(D)\lambda_i^k(\Lambda).\end{aligned}$$

Let $\nu = \max\{\sigma_0, \tau_0\}$. Suppose that for all $i = 1, \ldots, i_0 \le n$, $j = 1, \ldots, j_0 \le n$, we have

$$\nu = |\lambda_i(D)| = |\lambda_j(\Lambda)|,$$

and without loss of generality, suppose that $i_0 \ge j_0$, and none of the other eigenvalues satisfy this equality. Then we have two cases: $j_0 < n$, and $j_0 = n$ (and hence $i_0 = n$ as well).

(i) Suppose $j_0 < n$. We evaluate the left-hand term in our expression for $\delta(k)$ first. Because ν was chosen to be the max of the singular values, we have

$$\lim_{k\to\infty} \frac{\sum_{i=1}^n \lambda_i^{2k}(D) + \lambda_i^{2k}(\Lambda)}{\nu^{2k}} = i_0 + j_0.$$

Now, we examine the right-hand term. By similar reasoning,

$$\begin{aligned}\text{RHS} &= \lim_{k\to\infty} -2\frac{\sum_{i,j\le n} h_{ij}^2\lambda_i^k(D)\lambda_i^k(\Lambda)}{\nu^{2k}}\\ &= -2\frac{\sum_{i,j\le j_0} h_{ij}^2\lambda_i^k(D)\lambda_i^k(\Lambda)}{\nu^{2k}}\\ &= -2\sum_{i,j\le j_0}(-1)^{\beta_{ij}}\end{aligned}$$

where β_{ij} is defined by

$$(-1)^{\beta_{ij}} = \frac{\lambda_i^k(D)\lambda_i^k(\Lambda)}{\nu^{2k}}.$$

(This is because for $i, j \le j_0$, we have $\left|\lambda_i^k(D)\lambda_i^k(\Lambda)\right| = \nu^{2k}$, by definition of j_0). Now, because H is orthogonal, the columns of H are orthonormal, hence we obtain the inequality

$$\begin{aligned}\sum_{i,j\le j_0}(-1)^{\beta_{ij}}h_{ij}^2 &< \sum_{i=1}^n\sum_{j=1}^{j_0} h_{ij}^2\\ &= \sum_{j=1}^{j_0}\|\mathbf{h}_j\|_2^2\\ &= j_0\end{aligned}$$

with the inequality being strict since $j_0 < n$. This yields the lower bound $i_0 + j_0 - 2j_0 = i_0 - j_0$:

$$0 \le i_0 - j_0 < \lim_{k\to\infty}\frac{\delta(k)}{\nu^{2k}}$$

we obtain an upper bound when we always have $\beta_{ij} = 1$, yielding

$$\lim_{k\to\infty}\frac{\delta(k)}{\nu^{2k}} < i_0 + 3j_0$$

thus exists a constant $\alpha > 0$ such that

$$\lim_{k\to\infty}\frac{\delta(k)}{\nu^{2k}} = \alpha$$

hence

$$\boxed{\delta(k) \sim \alpha\nu^{2k}}$$

as desired.

(ii) Now, suppose $i_0 = n = j_0$. Then all of the singular values of D and Λ are the same. Suppose that k of the eigenvalues are of the same sign. If k is even, then $\lambda_i^{2k}(D) = \lambda_i^{2k}(\Lambda)$ for all i, and hence $(-1)^{\beta_{ij}} = 1$. It follows that

$$\begin{aligned}\sum_{i,j\le n}(-1)^{\beta_{ij}}h_{ij}^2 &= \sum_{i,j\le n}h_{ij}^2\\ &= n\end{aligned}$$

hence we have

$$\begin{aligned}\delta(k) &= n + n - 2n\\ &= 0.\end{aligned}$$

In fact, in general, $\delta(k)$ is subexponential in k whenever the multiplicity of ν is the same in both $\sigma(D)$ and $\sigma(\Lambda)$.

If k is odd, then we have

$$\sum_{i,j\le n}(-1)^{\beta_{ij}} < h_{ij}^2 = n,$$

and so

$$\delta(k) = \left(2n - \sum_{i,j\le n}(-1)^{\beta_{ij}}h_{ij}^2\right)\nu^{2k} = \alpha\nu^{2k},$$

where

$$\alpha = \left(2n - \sum_{i,j\le n}(-1)^{\beta_{ij}}h_{ij}^2\right).$$

■

COROLLARY 2.5.1. *Quantify all variables as above. Then $\alpha \le 4n$.* △

Proof. Note that $\forall i,j,\ \left|2h_{ij}\lambda_i^k(D)\lambda_i^k(\Lambda)\right| \le 2h_{ij}^2\nu^{2k}$. Then because $\sum_{i,j\le n}h_{ij}^2 = n$, we have

$$\begin{aligned}\delta(k) &= \sum_{i=1}^n \lambda_i^{2k}(D) + \lambda_i^{2k}(\Lambda) - 2\sum_{i,j\le n}h_{ij}^2\lambda_i^k(D)\lambda_i^k(\Lambda)\\ &\le 2n\nu^{2k} + 2n\nu^{2k}\\ &= 4n\nu^{2k}\end{aligned}$$

as desired. ■

Before deriving some results for Δ, we should observe that nowhere in the proof above did we utilize the condition that $M = Q+\xi$. Surely, we conjecture, we should be able to use this fact to tighten our bounds. Indeed, this is the case. We outline the proof in the theorem below. But first, we have a small lemma.

LEMMA 2.6. *Let $A, B \in M_{n\times n}(\mathbb{R})$. Then AB and BA share the same eigenvalues.* △

Proof. Let v be an eigenvector of AB, with associated eigenvalue λ. Then we have

$$ABv = \lambda v.$$

Left multiplying each side by B, we have

$$\begin{aligned}B(ABv) &= B\lambda v\\ &= \lambda Bv\\ &= BA(Bv)\end{aligned}$$

hence Bv is an eigenvector of BA with associated eigenvalue λ. Thus, every eigenvalue of AB is an eigenvalue of BA. Conversely, let u be an eigenvector of BA, with associated eigenvalue τ. Then

$$\begin{aligned}A(BAu) &= A\tau u\\ &= \tau Au\\ &= AB(Au)\end{aligned}$$

hence Au is an eigenvector of AB with associated eigenvalue τ. Thus, AB and BA have the same spectrum. ■

COROLLARY 2.6.1. *Let $Q \in \mathrm{Orth}(n)$, and let $M \in M_{n\times n}(\mathbb{R})$. Then M and QMQ^T have the same spectrum.* △

Proof. Take $A = QM$ and $B = Q^\mathsf{T}$, then apply the lemma above. ■

In the theorem below, we will apply Weyl's inequality for perturbations, hence we give the statement below for completeness:

THEOREM 2.7 (Weyl). *Let $H, P \in M_{n\times n}(\mathbb{R})$, and let $M = H + P$. Suppose M has eigenvalues*

$$\mu_1 \ge \mu_2 \ge \cdots \ge \mu_n,$$

H has eigenvalues

$$\eta_1 \ge \eta_2 \ge \cdots \ge \eta_n,$$

and P has eigenvalues

$$\rho_1 \ge \rho_2 \ge \cdots \ge \rho_n.$$

Then if any two of M, H, P are Hermitian, $\forall i = 1, \ldots, n$, we have

$$\eta_i + \rho_n \le \mu_i \le \eta_i + \rho_1.$$

△

Now the theorem.

THEOREM 2.8. *Let $Q \in \mathrm{Orth}(n)$ and $D \in M_{n\times n}(\mathbb{R})$ be diagonal, and let $\xi \in B_\varepsilon(0_{n\times n})$. Now, let $M = Q + \xi$, and let $\lambda_1, \lambda_2, \ldots, \lambda_n$ be the eigenvalues of QDQ^T, and let $\tau_1, \tau_2, \ldots, \tau_n$ be the eigenvalues of MDM^T. If $\lambda = \max_i(|\lambda_i|)$, then*

$$|\lambda_i - \tau_i| \le \lambda(2\varepsilon n + \varepsilon^2 n^2)$$

△

Proof. For this proof, we will make use of the spectral norm, hence we first note some properties of the spectra of $QDQ^\intercal$ and $MDM^\intercal$. By the corollary above, note that the eigenvalues of $QDQ^\intercal$ are simply the entries of D. Similarly, note that the spectrum of $MDM^\intercal$ is equivalent to the spectrum of $M^\intercal MD$. That is,

$$\begin{aligned}\sigma(MDM^\intercal) &= \sigma(M^\intercal MD)\\ &= \sigma\big((Q+\xi)^\intercal(Q+\xi)D\big)\\ &= \sigma\big(D + \xi^\intercal QD + Q^\intercal\xi D + \xi^\intercal\xi D\big)\end{aligned}$$

and thus

$$\begin{aligned}\left\|QDQ^\intercal - MDM^\intercal\right\|_2 &= \left\|D - M^\intercal Md\right\|_2\\ &= \left\|\xi^\intercal QD + Q^\intercal\xi D + \xi^\intercal\xi D\right\|_2\end{aligned}$$

Since $QDQ^\intercal$ and $MDM^\intercal$ are symmetric (and thus hermetian), we can apply Weyl's inequality by taking $M = QDQ^\intercal$, $H = MDM^\intercal$, and $P = QDQ^\intercal - MDM^\intercal$ to obtain the following bound:

$$\begin{aligned}|\lambda_i - \tau_i| &\le \left\|D - M^\intercal MD\right\|_2\\ &= \left\|\xi^\intercal QD + Q^\intercal\xi D + \xi^\intercal\xi D\right\|_2\end{aligned}$$

by submultiplicativity of the spectral norm, we have

$$\begin{aligned}&\le \|D\|_2\left\|\xi^\intercal Q + Q^\intercal\xi + \xi^\intercal\xi\right\|_2\\ &= \lambda\left\|\xi^\intercal Q + Q^\intercal\xi + \xi^\intercal\xi\right\|_2\end{aligned}$$

and so by the triangle inequality,

$$\begin{aligned}&\le \lambda\big(\left\|\xi^\intercal Q\right\|_2 + \left\|Q^\intercal\xi\right\|_2 + \left\|\xi^\intercal\xi\right\|_2\big)\\ &\le \lambda\big(\left\|\xi^\intercal\right\|_2\|Q\|_2 + \left\|Q^\intercal\right\|_2\|\xi\|_2 + \left\|\xi^\intercal\right\|_2\|\xi\|_2\big)\\ &= \lambda\big(\left\|\xi^\intercal\right\|_2 + \|\xi\|_2 + \left\|\xi^\intercal\right\|\|\xi\|\big)\\ &= \lambda\Big(2\|\xi\|_2 + \|\xi\|_2^2\Big)2\\ &\le \lambda\Big(2\|\xi\|_F + \|\xi\|_F^2\Big)2\\ &\le \lambda\big(2\varepsilon n + \varepsilon^2 n^2\big)\end{aligned}$$

as desired. ■

We have the following corollary:

COROLLARY 2.8.1. *Let $Q \in \mathrm{Orth}(n)$, and let $D \in M_{n\times n}(\mathbb{R})$ be diagonal. Suppose $M = Q + \xi$ where $\xi \in B_\varepsilon(0_{n\times n})$. Let $\lambda_1^k, \lambda_2^k, \ldots, \lambda_n^k$ be the eigenvalues of D^k, and let $\tau_1(k), \tau_2(k), \ldots, \tau_n(k)$ be the eigenvalues of $MD^kM^\intercal$ for $k \in \mathbb{N}$. If $\lambda = \max_i(|\lambda_i|)$, then*

$$\left|\lambda_i^k - \tau_i(k)\right| \le \lambda^k\big(2\varepsilon n + \varepsilon^2 n^2\big)$$

△

Proof. This follows immediately by replacing D in the previous theorem by D^k. ■

Using this result, we can tighten some of the bounds we presented earlier.

COROLLARY 2.8.2. *Let $Q \in \mathrm{Orth}(n)$ and let $D \in M_{n\times n}(\mathbb{R})$ be diagonal. Suppose $M = Q + \xi$ where $\xi \in B_\varepsilon(0_{n\times n})$. Then for sufficiently small ε,*

$$\delta(k) \le 4n(d(1 + 2\varepsilon n + \varepsilon^2 n^2))^{2k}$$

where $d = \max_i(|d_i|)$ △

Proof. In light of Theorem 2.8 we can assume for small enough ε that the largest singular value λ_l in $\sigma(QDQ^\intercal)$ corresponds to the largest singular value τ_l in $\sigma(MDM^\intercal)$. Therefore, we get that ν (the same ν in Theorem 2.5) is bounded by

$$d + d(2\varepsilon n + \varepsilon^2 n^2)$$

Applying Corollary 2.5.1 gives us the desired bound. ■

Note that, for all of the arguments above, by the equivalence of matrix norms, the results also hold (up to a constant multiple) under the Frobenius norm. Thus, we can apply them to our original bounds.

COROLLARY 2.8.3. *Let $Q \in \mathrm{Orth}(n)$, and let $\xi \in B_\varepsilon(0_{n\times n})$. As usual, let $M = Q + \xi$, and let $D \in M_{n\times n}(\mathbb{R})$ be diagonal, and let $\lambda = \max_i(|\lambda_i|)$. Then $\left|\sqrt{\Delta(k)} - \sqrt{\delta(k)}\right| \le \lambda^k(2 + \varepsilon n + \varepsilon^2 n^2)$.* △

Proof. Proving this result is now easy. Let $\lambda = \max_i \lambda_i(D)$. By the reverse triangle inequality, we have

$$\begin{aligned}\left|\sqrt{\Delta(k)} - \sqrt{\delta(k)}\right| &= |\|\mathbf{v}_\Delta\|_F - \|\mathbf{v}_\delta\||_F\\ &\le \|\mathbf{v}_\Delta - \mathbf{v}_\delta\|\\ &= \left\|MD^kM^\intercal - QD^kQ^\intercal\right\|\\ &\le \lambda^k(2\varepsilon n + \varepsilon^2 n^2)\end{aligned}$$

as desired. ■

We're interested in the bounds on $\Delta(k)$. We'll use a crude approach here to approximate it.

LEMMA 2.9. *Let $M = Q + \xi$, where $Q \in \mathrm{Orth}(n)$, and $\xi \in B_\varepsilon(0_{n\times n})$. Let $D \in M_{n\times n}(\mathbb{R})$ be diagonal. Then for $\varepsilon < 1$, $\left\|(MDM^\intercal)^k\right\|_2 \le \|D\|_2^k(1 + O(\varepsilon))$* △

Proof.

$$\begin{aligned}\left\|\prod_{i=1}^k MDM^\intercal\right\|_2 &= \left\|\prod_{i=1}^k (Q+\xi)D(Q+\xi)^\intercal\right\|_2\\ &\le \prod_{i=1}^k \left\|(Q+\xi)D(Q+\xi)^\intercal\right\|_2\\ &\le \prod_{i=1}^k \left\|QDQ^\intercal\right\|_2 + \left\|\xi DQ^\intercal\right\|_2\\ &\qquad + \left\|QD\xi^\intercal\right\|_2 + \left\|\xi D\xi^\intercal\right\|_2\\ &= \prod_{i=1}^k \|D\|_2\big(1 + 2\varepsilon + \varepsilon^2\big)\end{aligned}$$

$$= \|D\|_2^k \sum_{i=1}^{2k} \binom{2k}{i} \varepsilon^i$$
$$= \|D\|_2^k (1 + O(\varepsilon)),$$

As desired. ■

REMARK. This bound is exceptionally crude. See computational results section for performance analysis.

3. Computational Results & Randomness

We made considerable efforts to obtain theoretical answers to the following questions:

- If we were to draw ξ from $\mathcal{N}(0_{n\times n}, \varepsilon_{n\times n})$, how much tighter would our bounds on $\mathbb{E}[\delta(k)], \mathbb{E}[\Delta(k)]$ be than our current bounds on $\delta(k)$, $\Delta(k)$?
- On a related note, if we sample the entries of our random matrix ξ from another distribution, what
- How can we view the Johnson-Lindenstrauss Lemma from the perspective of almost orthogonal matrices?
- How can $n \times k$ (where $k \ll n$) almost-orthogonal matrices be used in
- Do we always have $\delta(k) > \Delta(k)$?
- Is $\delta(k) \sim \Delta(k)$?

However, after spending a lot of time trying to find literature on these topics and/or trying to prove the bounds ourselves from scratch, we realized we simply didn't have adequate machinery yet to try and approach these problems from a theoretical standpoint. As such, we decided to take a more computational approach, as is detailed in the following section.

3.1. Testing δ, Δ Bounds

To investigate the first question, we created some python code to test the $\mathbb{E}[\delta(k)]$ and $\mathbb{E}[\Delta(k)]$ when $\xi \sim \mathcal{N}(0_{n\times n}, \varepsilon)$ as a function of n, k, and ε. The ε dependence was not terribly interesting, hence we'll focus mainly on the k, n dependence here. Implementation details can be found in the appendix.

Proof. (For result 3.1; plots on next page)

The first big hurdle we have to work past is how we can test the *average*-case performance of the algorithm. Generating acceptably-distributed ξ is trivial, since we're given a particular distribution we want them to follow. However, we also want to make sure that our orthogonal matrices are also generated in an evenly-distributed manner, to ensure that we're not skewing the results by our particular choice of Q. We guarantee this as follows:

Data: Number of dimensions, n
Result: $Q \in \mathrm{Orth}(n)$
$M \leftarrow$ construct a random $n \times n$ matrix with uniformly distributed entries;
$Q, R \leftarrow$ apply QR decomposition to M;
return Q

Algorithm 1: Algorithm for generating our orthogonal matrix Q

Then, we perform the following computational scheme:

Data: Number of dimensions n, σ, k list, tests per k
Result: Plot of $\log(\delta(k)/\Delta(k))$
initialization of δ list, Δ list;
$Q, R \leftarrow$ apply QR decomposition to M;
return Q **for** $k \leftarrow\in$ *k list* **do**
  initialize list of $\delta(k), \Delta(k)$ results;
  for $0 \le i \le$ *tests per k* **do**
    $Q \leftarrow$ generate random orthogonal matrix Q;
    $\xi \leftarrow$ sample from $\mathcal{N}(0_{n\times n}, \sigma_{n\times n})$;
    $D \leftarrow$ generate a random diagonal matrix D;
    calculate $\delta(k)$, $\Delta(k)$ in the usual way;
    Push to list of $\delta(k)$, $\Delta(k)$ results;
  end
  take the average of the $\delta(k)$, $\Delta(k)$ lists and push to the δ, Δ lists;
end
perform linear regression on the $\log(\delta)$, $\log(\Delta)$ lists to verify the bound is tight;

Algorithm 2: Algorithm for testing the δ, Δ bounds

Interestingly, upon making plots of $\delta(k) - \Delta(k)$ and $\log\left(\frac{\delta(k)}{\Delta(k)}\right)$, we obtained the following empirical result:

RESULT 3.1. $\delta(k) - \Delta(k) = o(\delta(k)/\Delta(k))$ △

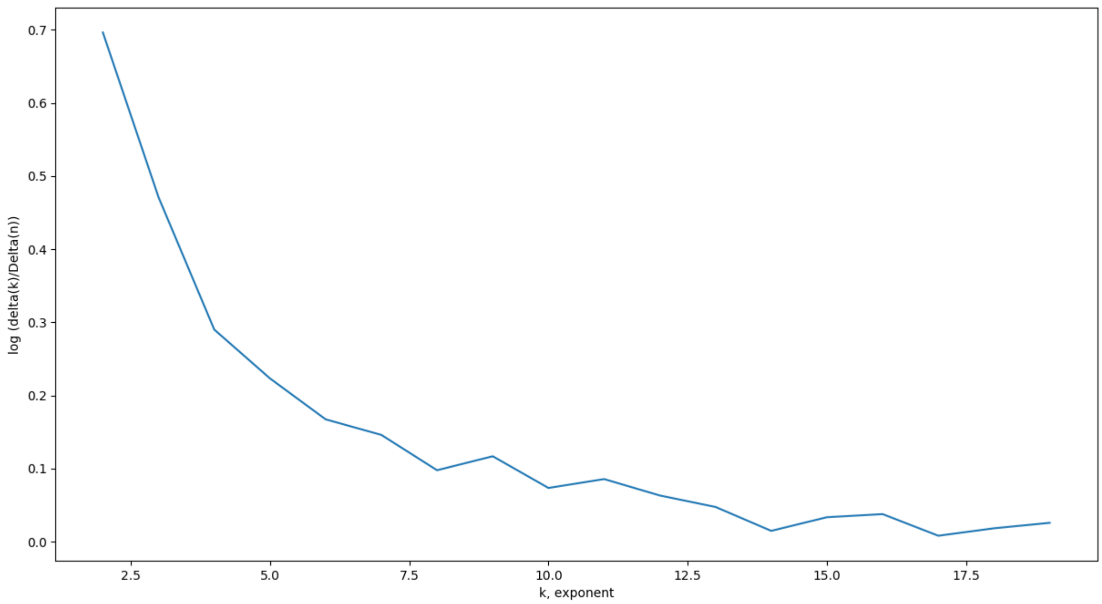

Figure 1: $\log\left(\frac{\delta(k)}{\Delta(k)}\right)$

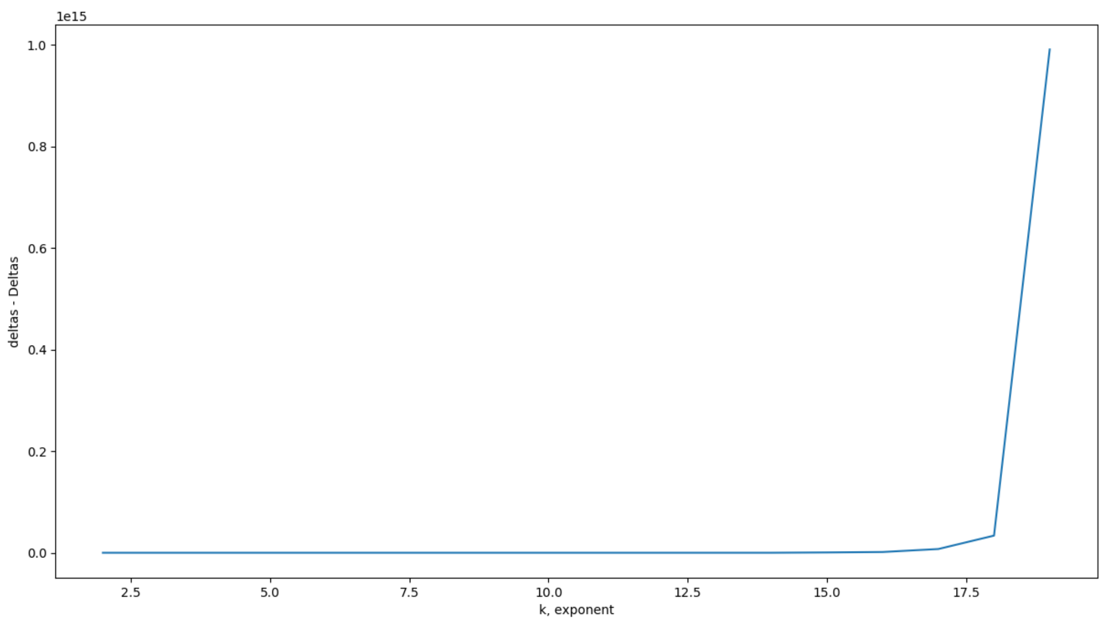

Figure 2: $\log\left(\frac{\delta(k)}{\Delta(k)}\right)$

hence we see $\log\left(\frac{\delta(k)}{\Delta(k)}\right) \to 1$, but $\delta(k) - \Delta(k)$ is unbounded, hence we have the desired result. $\blacksquare$

Similarly, observe the following empirical verification of the exponential nature of the bounds:

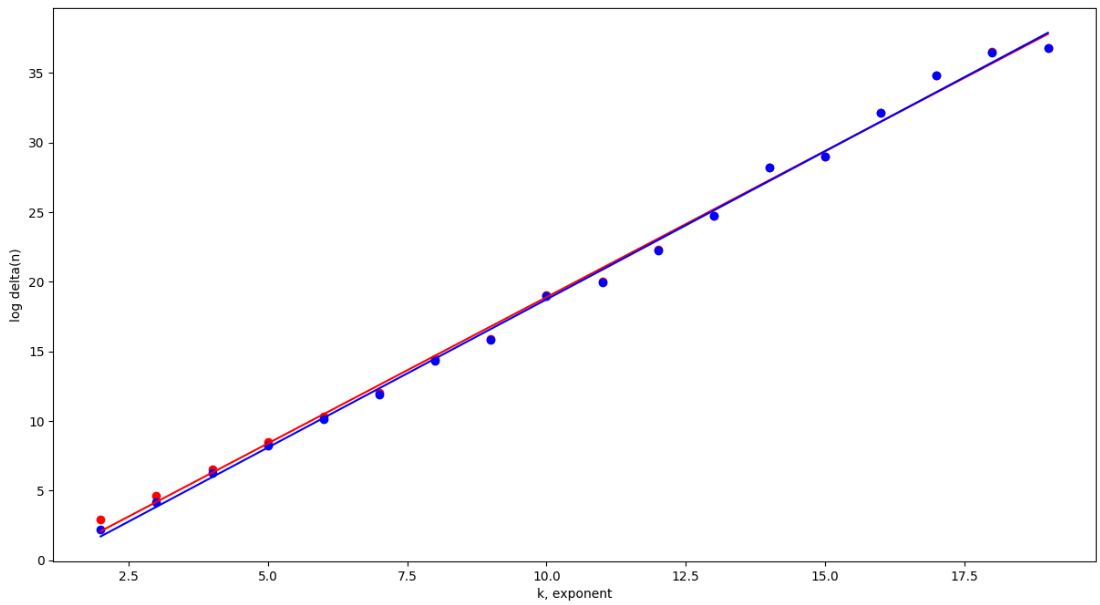

Figure 3: $\log(\delta(k))$

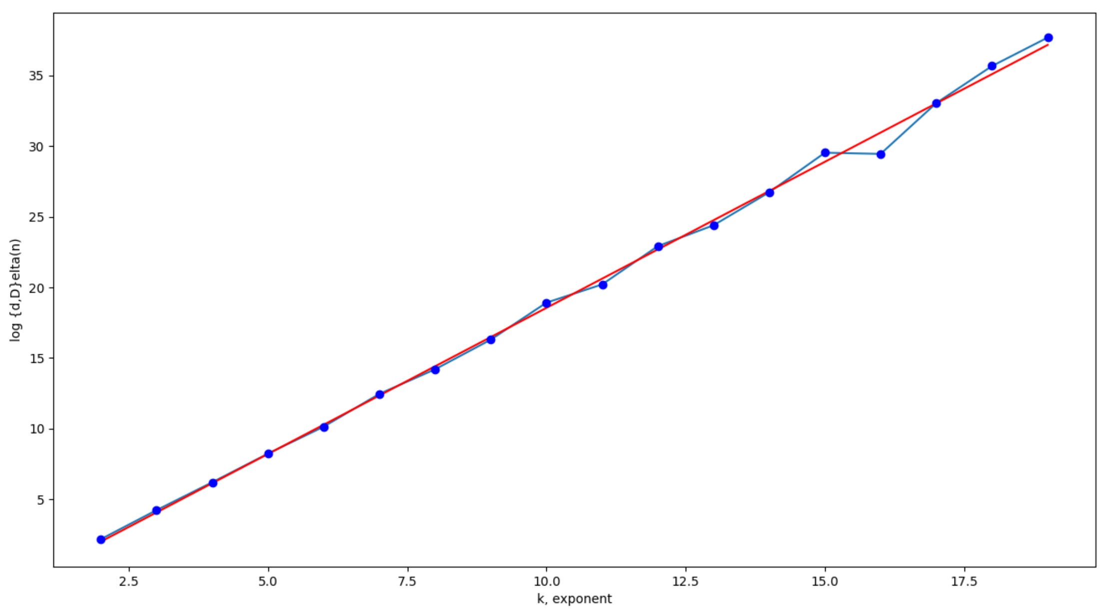

Figure 4: $\log(\Delta(k))$

hence we see $\log\left(\frac{\delta(k)}{\Delta(k)}\right) \to 1$, but $\delta(k) - \Delta(k)$ is unbounded, hence we have the desired result.

REFERENCES

[Axl15] Sheldon Jay Axler. *Linear Algebra Done Right*. Undergraduate Texts in Mathematics. Springer International Publishing, third edition, 2015.

A comprehensive text on intermediate Linear Algebra

[Bha13] R. Bhatia. *Matrix Analysis*. Springer New York, 2013.

[(ht] Jon (https://math.stackexchange.com/users/98140/jon). expected value calculation for squared normal distribution. Mathematics Stack Exchange. URL:https://math.stackexchange.com/q/511720 (version: 2013-10-02).

[Sch66] Peter H. Schönemann. A generalized solution of the orthogonal procrustes problem. *Psychometrika*, 31(1):1–10, Mar 1966.

[TB97] L. N. Trefethen and D. Bau. *Numerical Linear Algebra*. Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104), 1997.

4. Code

```
import scipy
import scipy.linalg
# import numpy as np
import numpy as np
import matplotlib.pyplot as plt

from tqdm import tqdm
from mpl_toolkits import mplot3d

class Distribution:
    """
    Just a simple object wrapper so that we can partially instantiate
    a distribution for ease of use in defining perturbations below.
    """

    def __init__(self, distribution, *args, **kwargs):
        """

        distribution should be a numpy.random.<distribution_name>

        *args should be whatever parameters the distribution needs to
        instantiate
        """
        self.distribution = distribution
        self.args = args
        self.kwargs = kwargs
        # self.kwargs = kwargs

    def __call__(self, **extra_kwargs):
        """
        samples the attached distribution given the args,
        """
        return self.distribution(*self.args, **self.kwargs, **extra_kwargs)


# Functions for applying distributions to matrices and stuff
def perturb(vec, distribution):
    """
    Create a correctly sized perturbation vector and add it to the
    input vector
    """
    return vec + distribution(size=vec.shape)

def perturb_mat(M, distribution):
    """
    Create a correctly sized perturbation matrix and add it to the
    input matrix
    """
    size = M.shape
    return M + distribution(size = size)


def construct_almost_basis(n, distribution):
    """
    Use a given distribution to perturb basis vectors yielding an
    (almost) orthonormal basis
    """
    I = np.eye(n)
    for i, row in enumerate(I):
        I[i] = perturb(row, distribution)
    return I

def orthog_error(A):
    """
    return error from the square root process as well as the
    calculation from the "closest" orthogonal matrix
    """
    sqrt = scipy.linalg.sqrtm(A.T @ A)
    invsqrt = np.linalg.inv(sqrt)
    R = A @ invsqrt
    return np.linalg.norm(A - R)

# Helper functions for plot generation
def f(x,y):
    """
```

```
    Example z_func.

    the transformation we'll apply to x and y in our surface plot below
    """
    try:
        # return 3*(x ** 2) + (y ** 2)
        return x * y / (x**2 + y**2)
    except ZeroDivisionError as e:
        return 0

def surface_plot(x, y, z_func=f, x_label="x", y_label="y", z_label="z"):
    """
    z_func should be a map from R^2 to R
    """
    # Clear any previous drawings
    plt.clf()

    # np.meshgrid basically gives an object such that we can apply f over the
    # object as seen below instead of looping and stuff
    X, Y = np.meshgrid(x, y)

    # Apply f over all of (X,Y)
    Z = z_func(X, Y)

    # Initialize figure object
    fig = plt.figure()

    # Inform matplotlib we'll have a 3d surface plot
    ax = plt.axes(projection='3d')

    # Plot the surface
    surface = ax.plot_surface(X, Y, Z, cmap='viridis')

    # Make a scale thing
    fig.colorbar(surface)

    # axis labels
    ax.set_xlabel(x_label)
    ax.set_ylabel(y_label)
    ax.set_zlabel(z_label)

    # Return object so that it can be used later. Also allows us to
    # decide between "show" and "save"
    return (fig, ax)


def plot_normal_3delbow(min_sigma=0.001, max_sigma=10,
                        num_tests=500, samps_per=20):

    # This is going to be the first axis thing to iterate over (in my
    # case, this is the stuff I'm gonna put on the x axis later)
    dims = np.array(range(2,20))

    #
    sample_sigma = np.linspace(min_sigma, max_sigma, num=num_tests)

    def z_func(dim, sigma):
        """
        Function to be applied to each entry
        """
        normal = Distribution(np.random.normal, loc=0.0, scale=sigma)
        sub_test = []
        for i in range(samps_per):
            almost = construct_almost_basis(dim, normal)
            sub_test += [orthog_error(almost)]
            # cond_num = np.linalg.cond(almost)
            # sub_test += [cond_num]
        sub_test = np.array(sub_test)
        # print(sub_test)
        return np.mean(sub_test)

    X, Y = np.meshgrid(dims, sample_sigma)

    z_func = np.vectorize(z_func)

    fig, ax = surface_plot(dims, sample_sigma, z_func=z_func,
                        x_label="dimension of matrix", y_label="std." +
                        "dev. of normal perturbation",
```

```
                         z_label="distance to closest orthogonal matrix")

    plt.show()

def exp_error(n, A, diag_matrix):
    """
    n: exponent to raise to

    Return the error as propogated through when taking the matrix
    exponential and treating the input
    """
    true_matrix = np.linalg.matrix_power(A @ diag_matrix @ A.T, n)
    est_matrix = A @ (diag_matrix ** n) @ A.T
    error = np.linalg.norm(true_matrix - est_matrix)
    return error/(np.linalg.norm(A)**n)


def plot_normal_exp_elbow(min_val=0.0001, max_val=0.5,
                          num_tests=500, samps_per=50):

    dim = 15
    sigmas = np.linspace(min_val, max_val, num=num_tests)
    Z = []

    exps = range(2,70)

    diag_matrix = np.diag(np.random.normal(loc=1.0, scale=.01, size=(dim,)))

    for exp in tqdm(exps):
        # Reverse the order to start with big errors then go to small

        y_vals = []
        # y_errs = []
        for sigma in sigmas:
            normal = Distribution(np.random.normal, loc=0.0, scale=sigma)
            sub_test = []
            for i in range(samps_per):
                # print(normal, dims)
                almost = construct_almost_basis(dim, normal)
                sub_test += [exp_error(exp, almost, diag_matrix)]

            # Convert to numpy array so we can use nice easy stats without
            # writing helper functions
            sub_test = np.array(sub_test)

            # Use the average value
            y_vals += [np.mean(sub_test)]

            # std_dev = np.std(sub_test)
            # y_errs += [std_dev/samps_per]

        Z += [y_vals]

    Z = np.array(Z).T
    X, Y = np.meshgrid(exps, sigmas)
    # print(X.shape, Y.shape, Z.shape)

    fig = plt.figure()
    ax = plt.axes(projection='3d')
    # ax.contour3D(X, Y, Z, 50, cmap='binary')
    ax.plot_surface(X, Y, np.log(Z), cmap='binary')
    ax.set_xlabel('exponent')
    ax.set_ylabel('stard deviation of perturbation')
    ax.set_zlabel('log normed exponent error')
    plt.show()



def plot_uniform_exp_elbow(min_val=0.0001, max_val=20,
                           num_tests=100, samps_per=100):

    dim = 15
    sample_range = np.linspace(min_val, max_val, num=num_tests)
    Z = []

    exps = range(2,20)

    diag_matrix = np.diag(np.random.normal(loc=1.0, scale=0.5, size=(dim,)))
```

```

    for exp in tqdm(exps):
        # Reverse the order to start with big errors then go to small

        y_vals = []
        # y_errs = []
        for val in sample_range:
            uniform = Distribution(np.random.uniform, low=-1*val, high=val)
            sub_test = []
            for i in range(samps_per):
                # print(normal, dims)
                almost = construct_almost_basis(dim, uniform)
                sub_test += [exp_error(exp, almost, diag_matrix)]

            # Convert to numpy array so we can use nice easy stats without
            # writing helper functions
            sub_test = np.array(sub_test)

            # Use the average value
            y_vals += [np.mean(sub_test)]

            # std_dev = np.std(sub_test)
            # y_errs += [std_dev/samps_per]

        Z += [y_vals]

    Z = np.array(Z).T
    X, Y = np.meshgrid(exps, sample_range)
    # print(X.shape, Y.shape, Z.shape)

    fig = plt.figure()
    ax = plt.axes(projection='3d')
    # ax.contour3D(X, Y, Z, 50, cmap='binary')
    ax.plot_surface(X, Y, np.log(Z+1), cmap='binary')
    ax.set_xlabel('exponent')
    ax.set_ylabel('range of uniform perturbation (-y to y)')
    ax.set_zlabel('log normed (exponent error +1)')
    plt.show()



def plot_uniform_3delbow(min_val=0.001, max_val=10,
                        num_tests=100, samps_per=5):

    dims = np.array(range(2,30))
    sample_range = np.linspace(min_val, max_val, num=num_tests)
    Z = []
    for dim in tqdm(dims):
        # Reverse the order to start with big errors then go to small

        y_vals = []
        # y_errs = []
        for val in sample_range:
            uniform = Distribution(np.random.uniform, low=-1*val, high=val)
            sub_test = []
            for i in range(samps_per):
                # print(normal, dims)
                almost = construct_almost_basis(dim, uniform)
                sub_test += [orthog_error(almost)]

            # Convert to numpy array so we can use nice easy stats without
            # writing helper functions
            sub_test = np.array(sub_test)

            # Use the average value
            y_vals += [np.mean(sub_test)]

            # std_dev = np.std(sub_test)
            # y_errs += [std_dev/samps_per]

        Z += [y_vals]

    Z = np.array(Z).T
    X, Y = np.meshgrid(dims, sample_range)
    # print(X.shape, Y.shape, Z.shape)

    fig = plt.figure()
    ax = plt.axes(projection='3d')
```

```
    ax.contour3D(X, Y, Z, 50, cmap='binary')
    ax.set_xlabel('dimension of matrix')
    ax.set_ylabel('range of normal perturbation (-y to y)')
    ax.set_zlabel('distance to closest orthogonal matrix')
    plt.show()

def plot_beta_3delbow(dim=10, min_val=0.001, max_val=10,
                      num_tests=100, samps_per=10):

    a_range = np.linspace(min_val, max_val, num=num_tests)
    b_range = a_range.copy()

    Z = []
    for a in tqdm(a_range):
        # Reverse the order to start with big errors then go to small

        y_vals = []
        # y_errs = []
        for b in b_range:
            beta = Distribution(np.random.beta, a, b)
            sub_test = []
            for i in range(samps_per):
                # print(normal, dims)
                almost = construct_almost_basis(dim, beta)
                sub_test += [orthog_error(almost)]

            # Convert to numpy array so we can use nice easy stats without
            # writing helper functions
            sub_test = np.array(sub_test)

            # Use the average value
            y_vals += [np.mean(sub_test)]

            # std_dev = np.std(sub_test)
            # y_errs += [std_dev/samps_per]

        Z += [y_vals]

    Z = np.array(Z).T
    X, Y = np.meshgrid(a_range, b_range)
    # print(X.shape, Y.shape, Z.shape)

    fig = plt.figure()
    ax = plt.axes(projection='3d')
    ax.plot_surface(X, Y, Z, cmap='viridis')
    ax.set_xlabel('alpha (beta distribution parameter)')
    ax.set_ylabel('beta (beta distribution parameter)')
    ax.set_zlabel('distance to closest orthogonal matrix')
    plt.show()

def plot_gamma_3delbow(dim=10, min_val=0.001, max_val=10,
                       num_tests=100, samps_per=10):

    k_range = np.linspace(min_val, max_val, num=num_tests)
    theta_range = k_range.copy()

    Z = []
    for k in tqdm(k_range):
        # Reverse the order to start with big errors then go to small

        y_vals = []
        # y_errs = []
        for t in theta_range:
            gamma = Distribution(np.random.gamma, k, scale=t)
            sub_test = []
            for i in range(samps_per):
                # print(normal, dims)
                almost = construct_almost_basis(dim, gamma)
                sub_test += [orthog_error(almost)]

            # Convert to numpy array so we can use nice easy stats without
            # writing helper functions
            sub_test = np.array(sub_test)

            # Use the average value
            y_vals += [np.mean(sub_test)]

            # std_dev = np.std(sub_test)
```

```python
            # y_errs += [std_dev/samps_per]

        Z += [y_vals]

    Z = np.array(Z).T
    X, Y = np.meshgrid(k_range, theta_range)
    # print(X.shape, Y.shape, Z.shape)

    fig = plt.figure()
    ax = plt.axes(projection='3d')
    ax.plot_surface(X, Y, Z, cmap='viridis')
    ax.set_xlabel('k (gamma distribution parameter)')
    ax.set_ylabel('theta (gamma distribution parameter)')
    ax.set_zlabel('distance to closest orthogonal matrix')
    plt.show()


def frobenius_tester():
    dimensions = range(2,100)
    sigma = .1
    normal = Distribution(np.random.normal, loc=0.0, scale=sigma)

    tests_per_dim = 10000

    norms = []

    for dim in tqdm(dimensions):
        this_dim_norms = []
        for test in range(tests_per_dim):
            this_dim_norms += [np.linalg.norm(normal(size=(dim,dim)))]

        norms += [np.mean(np.array(this_dim_norms)) - (dim*sigma)]

    print(np.mean(norms))
    plt.plot(dimensions, norms)
    plt.xlabel("Dimension of xi matrix")
    plt.ylabel("Error ||xi||_F relative to (dim * sigma)")
    plt.title("Error (||xi||_F - dim * sigma); 10000 tests per dim, dim in" +
    "[2..200]; sigma = 0.1")
    plt.show()


def frobenius_tester():
    dimensions = range(2,100)
    sigma = .1
    normal = Distribution(np.random.normal, loc=0.0, scale=sigma)

    tests_per_dim = 10000

    norms = []

    for dim in tqdm(dimensions):
        this_dim_norms = []
        for test in range(tests_per_dim):
            this_dim_norms += [np.linalg.norm(normal(size=(dim,dim)))]

        norms += [np.mean(np.array(this_dim_norms)) - (dim*sigma)]

    print(np.mean(norms))
    plt.plot(dimensions, norms)
    plt.xlabel("Dimension of xi matrix")
    plt.ylabel("Error ||xi||_F relative to (dim * sigma)")
    plt.title("Error (||xi||_F - dim * sigma); 10000 tests per dim, dim in" +
    "[2..200]; sigma = 0.1")
    plt.show()

def delta(n, A, xi, diag_matrix):
    """
    n: exponent to raise to

    Return the error as propogated through when taking the matrix
    exponential and treating the input
    """
    M = A + xi
    true_matrix = np.linalg.matrix_power(M @ diag_matrix @ M.T, n)
    est_matrix = A @ (diag_matrix ** n) @ A.T
    error = np.linalg.norm(true_matrix - est_matrix)
    return error
```

```python

def Delta(n, A, xi, diag_matrix):
    """
    n: exponent to raise to

    Return the error as propogated through when taking the matrix
    exponential and treating the input
    """
    M = A + xi
    true_matrix = np.linalg.matrix_power(M @ diag_matrix @ M.T, n)
    est_matrix = M @ (diag_matrix ** n) @ M.T
    error = np.linalg.norm(true_matrix - est_matrix)
    return error


def get_rms(W, X, y):
    """ gets the root mean square error """
    return np.linalg.norm(y - D@W, ord=2)/np.sqrt(D.shape[0])

def lin_reg(X, y):
    """ Takes two numpy arrays as inputs, and calculates the coefficients
    minimizing the least-square error between them """
    # Solve for optimal weight parameters minimizing regression
    X = np.column_stack((np.ones_like(X), X))
    W_opt = np.linalg.solve(X.T @ X, X.T @ y)
    return W_opt

def gen_rand_orthog(dim):
    rand_mat = np.random.rand(dim,dim)
    q, r = np.linalg.qr(rand_mat)
    return q

def test_theorem_4(sigma=.2, plot=True):

    dim = 20

    # dimensions = np.array(range(2,30))
    exps = range(2,50)

    normal = Distribution(np.random.normal, loc=0.0, scale=sigma)

    tests_per_exp = 100

    deltas = []
    Deltas = []

    A = gen_rand_orthog(dim)
    xi = normal(size=(dim,dim))

    for exp in exps:
    # for exp in tqdm(exps):
        this_dim_deltas = []
        this_dim_Deltas = []
        for test in range(tests_per_exp):
            test_mat = np.diag(np.random.normal(loc=1.0, scale=2,
                                                size=(dim,)))
            this_dim_deltas += [delta(exp, A, xi, test_mat)]
            this_dim_Deltas += [Delta(exp, A, xi, test_mat)]

        deltas += [np.mean(np.array(this_dim_deltas))]
        Deltas += [np.mean(np.array(this_dim_Deltas))]

    deltas = np.array(deltas)
    Deltas = np.array(Deltas)

    logdeltas = np.log(deltas)
    logDeltas = np.log(Deltas)

    exps = np.array(exps)

    bd, md = lin_reg(exps, logdeltas)
    bD, mD = lin_reg(exps, logDeltas)

    if plot:
        plt.plot(exps, logdeltas, 'ro', exps, logDeltas, 'bo', exps,
                 (md*exps) +bd, "r", exps, (mD * exps) + bD, "b")
        plt.xlabel("n, exponent")
        plt.ylabel("log delta(n)")
```

```
        plt.show()

        plt.plot(exps, np.log(np.array(Deltas)))
        plt.plot(exps, logDeltas, "bo")
        plt.xlabel("n, exponent")
        plt.ylabel("log {d,D}elta(n)")
        plt.show()

        plt.plot(exps, np.log(np.log(deltas/Deltas)))
        plt.xlabel("n, exponent")
        plt.ylabel("diff of log deltas (n)")
        plt.show()

    return (bd, md, bD, mD)

def test_ms():
    plt.clf()
    num_tests = 50
    min_sigma = 0.001
    max_sigma = 3
    sigmas = np.linspace(min_sigma, max_sigma, num=num_tests)

    bds = []
    mds = []
    bDs = []
    mDs = []

    for sigma in tqdm(sigmas):
        bd, md, bD, mD = test_theorem_4(sigma=sigma, plot=False)

        bds += [bd]
        mds += [md]
        bDs += [bD]
        mDs += [md]

    bds = np.array(bds)
    mds = np.array(mds)
    bDs = np.array(bDs)
    mDs = np.array(mDs)

    print(sigmas.shape, bds.shape, mds.shape, bDs.shape, mDs.shape)

    plt.plot(sigmas, bds, "r", sigmas, mds, "b", sigmas, bDs, "ro",
             sigmas, mDs, "bo")

    plt.ylabel("slope of best linear fit to log deltas")
    plt.xlabel("sigma")
    # plt.savefig("test.png")
    plt.show()

def closest_orthog(A):
    sqrt = scipy.linalg.sqrtm(A.T @ A)
    invsqrt = np.linalg.inv(sqrt)
    R = A @ invsqrt
    return A - R

def test_orthog_trace():

    numtests = range(100)
    dims = range(2,50)

    traces = []
    # trace_errs = []

    traces2 = []
    # trace_errs2 = []

    for dim in tqdm(dims):
        this_dim_traces = []
        this_dim_traces2 = []

        for test in numtests:
            rand_mat = 2*np.random.randn(dim,dim) + 1
            this_dim_traces += [np.trace(closest_orthog(rand_mat))]

            Q, _ = np.linalg.qr(rand_mat)
            this_dim_traces2 += [np.trace(Q)]

```

```
        this_dim_traces = np.array(this_dim_traces)
        this_dim_traces2 = np.array(this_dim_traces2)

        # sigma = np.std(this_dim_traces)
        mu = np.mean(this_dim_traces)
        traces += [mu]
        # trace_errs += [sigma]

        # sigma2 = np.std(this_dim_traces2)
        mu2 = np.mean(this_dim_traces2)
        traces2 += [mu2]
        # trace_errs2 += [sigma2]

    plt.plot(dims, traces, "b", dims, traces2, "r")

    plt.show()


def plot_beta_3delbow(dim=10, min_val=0.001, max_val=10,
                      num_tests=100, samps_per=10):

    a_range = np.linspace(min_val, max_val, num=num_tests)
    b_range = a_range.copy()

    Z = []
    for a in tqdm(a_range):
        # Reverse the order to start with big errors then go to small

        y_vals = []
        # y_errs = []
        for b in b_range:
            beta = Distribution(np.random.beta, a, b)
            sub_test = []
            for i in range(samps_per):
                # print(normal, dims)
                almost = construct_almost_basis(dim, beta)
                sub_test += [orthog_error(almost)]

            # Convert to numpy array so we can use nice easy stats without
            # writing helper functions
            sub_test = np.array(sub_test)

            # Use the average value
            y_vals += [np.mean(sub_test)]

            # std_dev = np.std(sub_test)
            # y_errs += [std_dev/samps_per]

        Z += [y_vals]

    Z = np.array(Z).T
    X, Y = np.meshgrid(a_range, b_range)
    # print(X.shape, Y.shape, Z.shape)

    fig = plt.figure()
    ax = plt.axes(projection='3d')
    ax.plot_surface(X, Y, Z, cmap='viridis')
    ax.set_xlabel('alpha (beta distribution parameter)')
    ax.set_ylabel('beta (beta distribution parameter)')
    ax.set_zlabel('distance to closest orthogonal matrix')
    plt.show()
```

almost.py