Learning with Noise, Sparse Errors, and Missing Data

by

Eric L. Perkerson

(Under the Direction of Alexander Petukhov)

Abstract

We introduce three algorithms for solving problems related to the matrix completion problem, with applications to machine learning problems. The first algorithm is the Affine Low-Rank Matrix Completion algorithm, which can handle noise, outliers, and missing data. It extends convex optimization algorithms for robust matrix completion to work for affine low-rank matrices, which have the form of a low-rank matrix plus an affine shift term in the form a matrix with identical columns.

The second algorithm is the Robust Orthogonal Rank-One Matrix Pursuit algorithm, which extends the Orthogonal Rank-One Matrix Pursuit algorithm by making it robust to outliers. The OR1MP algorithm is a greedy algorithm, which we pair with a greedy approach to identify outliers and then treat them as missing data, which any matrix completion algorithm is already capable of handling.

The third algorithm computes an approximate square distance function by training a neural network on a data set generated by adding noise vectors to data drawn from a smooth manifold. This function can be used to solve manifold learning problems that are a nonlinear extension of the matrix completion problem, and has numerous applications in machine learning, such as classification, projection, and style transfer.

Index words: Matrix Completion, Manifold Learning, Missing Data, Outliers

Learning with Noise, Sparse Errors, and Missing Data

by

Eric L. Perkerson

B.S., University of Georgia, 2013

M.A., University of Georgia, 2013

A Dissertation Submitted to the Graduate Faculty

of The University of Georgia in Partial Fulfillment

of the

Requirements for the Degree

Doctor of Philosophy

Athens, Georgia

2019

Learning with Noise, Sparse Errors, and Missing Data

by

Eric L. Perkerson

Approved:

Major Professor: Alexander Petukhov

Committee: Ming-Jun Lai
Edward Azoff
Qing Zhang

Electronic Version Approved:

Ron Walcott
Interim Dean of the Graduate School
The University of Georgia
December 2019

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Modern machine learning techniques often rely on large amounts of data, far more than can be processed by hand. This data often is corrupted in ways that can prevent the algorithms that rely on this data from working correctly, as most algorithms make some important assumptions about their input data which may not be satisfied by real-world data. Often, this problem is corrected by "cleaning" the data, often by hand in a time-consuming and expensive process, before being used as input by a particular algorithm. To avoid this, there is great interest in automatic cleaning of data or in algorithms which can be fed corrupted data directly without needing cleaning.

In this thesis, we present various machine learning algorithms that have been developed to handle corrupted data directly without preprocessing. We will assume that the input data consists of some number of data points, each of which is a vector of real-valued data. In particular, we focus on three types of corruption:

(a) Noise: a low-magnitude, dense corruption, i.e., one affecting every or nearly every coordinate of every data point. This type of corruption typically affects all real-valued data due to imprecision in measurements or approximation error in the model being used.

(b) Errors: a high-magnitude, sparse corruption with unknown location. "Sparse" here means that this corruption affects relatively few data points' coordinates. Potentially all data points could have some number of their coordinates affected by errors, but the sum of the number of coordinates corrupted over all data points is assumed to be sparse. Data that has had an error applied to it is usually referred to as an "outlier," though this term is sometimes reserved for the entire data point and not just some coordinates of a data point.

(c) Erasures: a high-magnitude corruption or deleted entry with a known location. Coordinates affected by erasures are typically called "missing" or "missing data." We choose to refer to "erasures" instead of "missing data" because an "erasure" can be applied to an existing data point much like an error so the terminology is similar.

The key observation for the presented algorithms is the following: Sparsity can be leveraged to identify the locations of errors. The high magnitude of errors means that the resulting data plus error often no longer conveys the information present in the data, essentially destroying the coordinates affected by this type of corruption. In this case, once the error is identified, the error can be treated as an erasure. If the algorithm is already made to handle erasures, then simply being able to identify errors means that the algorithm can also handle errors. Assuming that the algorithm is already capable of handling noise, we now have an algorithm capable of dealing with noise, errors, and erasures.

The first algorithm is the Affine Low-Rank Matrix Completion (ALRMC) algorithm, a numerical algorithm for the recovery of an affine low-rank matrix corrupted by noise, errors, and erasures, based on the Proximal Forward-Backward Splitting algorithm. This recovery problem arises when solving the rigid motion segmentation problem, i.e., the problem of identifying which feature tracks in a video sequence belong to a common rigid object. Under orthography, feature tracks that all belong

to a particular rigid object lie in a 4-dimensional affine subspace. Hence the interest in an effective algorithm to recover a low-dimensional affine subspace from the data points. With this application in mind, we allow for three sources of corruption in the data common to video: noise (low magnitude), errors (high magnitude but sparse), and erasures (missing data or high magnitude corruptions with known locations).

The second algorithm is the Robust Orthogonal Rank-One Matrix Pursuit algorithm (RORM1P), which extends the Orthogonal Rank-One Matrix Pursuit algorithm by making it robust to outliers. The OR1MP algorithm is an algorithm for solving the matrix completion problem, i.e., the problem of recovering a low rank matrix corrupted by erasures. It does so by building up a basis of rank one matrices by finding the rank one matrices most correlated with a running residual matrix, then by estimating the optimal coefficients on these basis matrices to approximate the given data matrix. The problem in this paper is to recover a low rank matrix that has been corrupted not simply by erasures, but errors as well. The ROR1MP algorithm, extends the OR1MP algorithm by greedily identifying the errors and then by treating them as missing entries.

The third algorithm computes an approximate square distance function by training a neural network on a data set generated by adding noise vectors to data drawn from a smooth manifold. This function can be used to solve manifold learning problems that are a nonlinear extension of the matrix completion problem, and has numerous applications in machine learning, such as classification, projection, and style transfer. By learning an approximate squared-distance function, we learn a fair amount about the structure of the manifold M in a way that lends itself to useful applications.

Chapter 2

Affine Low-Rank Matrix Completion Algorithm

2.1 Introduction

Low-rank matrices have enough structure to allow recovery after some amount of corruption. For example, in principal component analysis (PCA), one can take a low-rank matrix corrupted by additive noise and recover the original low-rank matrix and the added noise. Throughout this dissertation, noise is defined to be a low-magnitude, dense corruption, i.e. no assumption is made about the sparsity of the noise. In robust principal component analysis (RPCA, see [26]), a low-rank matrix can be corrupted by additive noise and errors and can still be recovered. Throughout, errors will be defined as high-magnitude, sparse corruptions. In the matrix completion (MC) problem, a low-rank matrix is corrupted by erasures, i.e. certain entries of the matrix are deleted entirely and no information is given about them. Still the low-rank matrix has enough structure to be recovered effectively if not too many entries are erased.

In this dissertation, we consider the recovery of *affine low-rank matrices*.

Definition 2.1.1. An affine low-rank matrix M is an $m \times n$ matrix Y that can be written as $M = A + c\mathbb{1}_{1\times n}$ for some shift vector $c \in \mathbb{R}^m$, where $\operatorname{rank}(A) < \min(m, n)$ and $\mathbb{1}_{1\times n}$ denotes the vector in $\mathbb{R}^{1\times n}$ with all entries equal to 1. In other words,

$$Y = A + \begin{bmatrix} | & | & & | \\ c & c & \dots & c \\ | & | & & | \end{bmatrix}$$

This definition is motivated by the problem of recovering an affine linear space from a set of corrupted measurements, where M denotes the matrix with each uncorrupted data vector as a column. The rank of A is then the dimension of the affine linear space, and the vector c is an affine shift. Clearly, a given affine subspace will not have a unique affine shift c, so throughout we will assume that c is the affine shift with minimal ℓ_2 norm.

The motivating application for this problem is that of rigid motion segmentation problem, viz. the problem of identifying which feature tracks in a video sequence belong to a common rigid object. Under the assumption of orthography, feature tracks belonging to a common rigid object form a 4-dimensional affine linear space. The goal is to recover a basis and affine shift for the affine linear space formed by the feature tracks all belonging to one rigid object.

This application motivates the three types of corruption considered:

(a) Noise: a low-magnitude, dense corruption. This can come from many sources, including small violations in the assumption of orthography, in the assumption that an object is rigid, or lighting or texture changes in the video.

(b) Errors: a high-magnitude, sparse corruption with unknown location. This can arise from feature tracks being mis-identified as belonging to the particular rigid

object being studied, or from feature tracks that have "jumped" to the wrong point in the video. Undetected occlusions also cause errors.

(c) Erasures: a high-magnitude corruption or deleted entry with a known location. This type of corruption arises with algorithms that can detect when a feature track is no longer reliable. Detected occlusions are also erasures.

To solve the recovery problem means finding a basis and affine shift term for the affine linear subspace that the uncorrupted data points belong to. In this dissertation we will consider three different problems with three different algorithms for their solutions: ALRMCvA is the model with an affine low-rank matrix corrupted only by noise, ALRMCvB is the model with an affine low-rank matrix corrupted by noise and errors, and ALRMC (the most general version) is the model with an affine low-rank matrix corrupted by noise, errors, and erasures. We can think about ALRMCvA as extending PCA to affine low-rank matrices; ALRMCvB extending RPCA to affine low-rank matrices; and ALRMC combines this extension of RPCA with the matrix completion problem.

In this dissertation we will denote the ℓ_p norm on vectors $x \in \mathbb{R}^n$ by $\|x\|_p = (\sum_{i=1}^n x_i^p)^{1/p}$. On matrices $A \in \mathbb{R}^{m\times n}$, the operator norm will be denoted $\|A\|_{p\to q}$, the Schatten-p norm will be denoted $\|A\|_{\mathrm{Sch}(p)}$, and the entrywise-p norm will be denoted $\|A\|_{\mathrm{vec}(p)}$, with definitions below:

$$\begin{aligned}
\|A\|_{p\to q} &= \sup_{\|x\|_p\leq 1} (\|Ax\|_q) \\
\|A\|_{\mathrm{Sch}(p)} &= \left(\sum_{i=1}^{\min(m,n)} \sigma_i(A)^p\right)^{1/p} \\
\|A\|_{\mathrm{vec}(p)} &= \left(\sum_{i,j} A_{i,j}^p\right)^{1/p}
\end{aligned}$$

As an extension of this notation, we will use the Hamming weight (also called the "ℓ_0 norm," despite not being absolutely homogenious), denoted by $\|x\|_0 = |\{i \in \{1, \ldots, n\} \colon x_i \neq 0\}|$ which counts the number of non-zero entries in a vector $x \in \mathbb{R}^n$. As a further extension, $\operatorname{rank}(A) = \|A\|_{\mathrm{Sch}(0)}$ is simply the number of non-zero singular values, and $\|A\|_{\mathrm{vec}(0)}$ is the number of non-zero entries of A.

Matrices will usually be denoted with capital letters, e.g. $A \in \mathbb{R}^{m \times n}$. The $(i,j)^{\text{th}}$ entry of the matrix A will be denoted by either $A_{i,j}$ or $\pi_{i,j}(A)$. The i^{th} column vector of a matrix A will be denoted $A_{:,i}$, and the i^{th} row vector of a matrix A will be denoted $A_{i,:}$.

In the PCA problem, the data matrix is modeled as $Y = A + Z$, where A is a low-rank matrix and Z is a matrix of noise. The solution to the problem is formulated as the solution to the optimization problem

$$\operatorname*{argmin}_A \left(\operatorname{rank}(A) + \frac{\lambda_z}{2} \|Y - A\|_{\mathrm{vec}(2)}^2 \right)$$

where $\lambda_z > 0$ weighs the importance of minimizing the $\operatorname{rank}(A)$ term versus the $\frac{\lambda_z}{2}\|Z\|_{\mathrm{vec}(2)}^2 = \frac{\lambda_z}{2}\|Y - A\|_{\mathrm{vec}(2)}^2$ term. However, this problem is in fact NP-hard, so instead we use the convex relaxation

$$\operatorname*{argmin}_A \left(\|A\|_{\mathrm{Sch}(1)} + \frac{\lambda_z}{2} \|Y - A\|_{\mathrm{vec}(2)}^2 \right),$$

where $\operatorname{rank}(A) = \|A\|_{\mathrm{Sch}(0)}$ is replaced by $\|A\|_{\mathrm{Sch}(1)}$. Note also that the norm $\|\cdot\|_{\mathrm{vec}(2)}^2$ is the Frobenius norm.

In the RPCA problem from [26], the data matrix is modeled as $Y = A + E + Z$, where the E term is a matrix of errors. The solution to the problem is formulated as the solution to the optimization problem

$$\operatorname*{argmin}_{A,E} \left(\operatorname{rank}(A) + \lambda_e \|E\|_{\mathrm{vec}(0)} + \frac{\lambda_z}{2} \|Y - A\|_{\mathrm{vec}(2)}^2 \right).$$

Here, λ_e, $\lambda_z > 0$ weigh the importance of minimizing the different terms. This is also NP-hard, but can be reformulated with the convex relaxation

$$\underset{A,E}{\operatorname{argmin}} \left(\|A\|_{\mathrm{Sch}(1)} + \lambda_e \|E\|_{\mathrm{vec}(1)} + \frac{\lambda_z}{2} \|Y - A\|^2_{\mathrm{vec}(2)} \right).$$

In the sections below, we will formulate similar convex optimization problems to solve the different ALRMC problems. Each of the convex optimization problems used involves both differentiable and non-differentiable terms, so we use either the proximal forward-backward splitting algorithm (in fact a slightly generalized version, see below) to numerically solve these convex optimization problem.

2.2 The Proximal Forward-Backward Splitting Algorithm

The proximal forward-backward splitting algorithm is an iterative algorithm for finding the minimum of a function $h(x) = f(x) + g(x)$, where both f and g are proper convex functions that are lower-semicontinuous, and where f has an L-Lipschitz continuous gradient. However, g is not assumed to be differentiable. The updates are of the form

$$x_{k+1} = [\mathrm{prox}_{\eta g} \circ (\mathrm{Id} - \gamma \nabla f)](x_k)$$

where $\mathrm{prox}_{\eta g}$ is the proximal operator (see definition below) associated with the function ηg for some parameter $\eta > 0$, and where $0 < \gamma < \frac{2}{L}$. Here, γ is the step size for the first part of the update step, and η is the step size for the second part of the update step. In this dissertation, the most general version of the numerical algorithm given is a slight generalization of the proximal forward-backward splitting algorithm, because we will allow the variable $x = (A, E, c)$ to have different step sizes η for the proximal operator over the subvariables A, E, and c.

Definition 2.2.1. The proximal operator associated with a function $g\colon \mathbb{R}^n \to \mathbb{R}$ is defined as

$$\text{prox}_g(x) = \underset{w\in\mathbb{R}^n}{\text{argmin}} \left(g(w) + \frac{1}{2}\|w - x\|_2^2 \right).$$

Often, the associated function is written with a multiplicative parameter $\gamma > 0$ that is often interpreted as a step size when the proximal operator is used in iterative algorithms, in which case we have

$$\begin{aligned} \text{prox}_{\gamma g}(x) &= \underset{w\in\mathbb{R}^n}{\text{argmin}} \left(\gamma g(w) + \frac{1}{2}\|w - x\|_2^2 \right) \\ &= \underset{w\in\mathbb{R}^n}{\text{argmin}} \left(g(w) + \frac{1}{2\gamma}\|w - x\|_2^2 \right) \end{aligned}$$

To formulate the different versions of the ALRMC algorithm to follow, we need the following lemmas:

Lemma 2.2.1. *An equivalent formulation for* $\text{prox}_{\gamma g}(x)$ *is given by*

$$\text{prox}_{\gamma g}(x) = (\text{Id} + \gamma \partial g)^{-1}(x)$$

where Id *is the identity operator and* ∂g *is the subgradient of* g*.*

Proof.

$$
\begin{aligned}
u = \operatorname{prox}_{\gamma g}(x) &= \operatorname*{argmin}_{w\in\mathbb{R}} \left(g(w) + \frac{1}{2\gamma}\|w-x\|^2 \right) \\
&\Rightarrow 0 \in \partial_w \left(g(w) + \frac{1}{2\gamma}\|w-x\|^2 \right) \Big|_u \\
&\Rightarrow 0 \in \partial g|_u + \frac{1}{\gamma}(w-x))|_u \\
&\Rightarrow x - u \in \gamma \partial g|_u \\
&\Rightarrow x \in (\mathrm{Id} + \gamma \partial g)(u) \\
&\Rightarrow u = (\mathrm{Id} + \gamma \partial g)^{-1}(x) \\
&\Rightarrow \operatorname{prox}_{\gamma g}(x) = (\mathrm{Id} + \gamma \partial g)^{-1}(x)
\end{aligned}
$$

□

Lemma 2.2.2. *The proximal operator associated to the absolute value function* $|\cdot| : \mathbb{R} \to \mathbb{R}$ *is* $\operatorname{prox}_{\gamma|\cdot|} = \mathcal{S}_\gamma$*, where*

$$
\mathcal{S}_\gamma(x) = \begin{cases} x+\gamma, & x < -\gamma \\ 0, & -\gamma \le x \le \gamma \\ x-\gamma, & x > \gamma \end{cases}
$$

is the soft-thresholding operator with parameter γ

Proof. First note that

$$
\partial \gamma|\cdot| = \begin{cases} -\gamma, & x < 0 \\ [-\gamma, \gamma], & x = 0 \\ \gamma, & x > 0 \end{cases},
$$

and thus

$$(I+\gamma\partial|\cdot|)(x) = \begin{cases} x-\gamma, & x<0 \\ [x-\gamma, x+\gamma], & x=0 \\ x+\gamma, & x>0 \end{cases}$$

$$\Rightarrow (I+\gamma\partial|\cdot|)^{-1}(u) = \begin{cases} u+\gamma, & u<-\gamma \\ 0, & -\gamma \leq u \leq \gamma \\ u-\gamma, & u>\gamma \end{cases}$$

Therefore, $\text{prox}_{\gamma|\cdot|}$ is the soft-thresholding operator with parameter γ, i.e. $\text{prox}_{\gamma|\cdot|} = \mathcal{S}_\gamma$. □

Lemma 2.2.3 (Splitting-sum Property)**.** *Suppose that the function $g(x,y)$ can be written as $g(x,y) = g_1(x) + g_2(y)$. Then $\text{prox}_{\gamma g}(x,y) = (\text{prox}_{\gamma g_1}(x), \text{prox}_{\gamma g_2}(y))$.*

Proof. The result follows immediately from the definition of the proximal operator. □

Lemma 2.2.4. *The proximal operator associated to the function $\gamma\|\cdot\|_{vec(1)} \colon \mathbb{R}^{m\times n} \to \mathbb{R}$ is given by the soft-thresholding operator with parameter γ applied entrywise, i.e.*

$$(\text{prox}_{\gamma\|\cdot\|_{vec(1)}})_{i,j}(M) = \begin{cases} M_{i,j}+\gamma, & M_{i,j}<-\gamma \\ 0, & -\gamma \leq M_{i,j} \leq \gamma \\ M_{i,j}-\gamma, & M_{i,j}>\gamma \end{cases}$$

Proof. This follows directly from 2.2.3 and 2.2.4 above. □

Lemma 2.2.5. *The proximal operator associated to the function $\gamma\|\cdot\|_{Sch(1)} \colon \mathbb{R}^{m\times n} \to \mathbb{R}$ is given by the soft-thresholding operator with parameter γ applied to the singular*

values of the argument, i.e.

$$\text{prox}_{\gamma\|\cdot\|_{Sch(1)}}(W) = US_\gamma(\sigma(W))V^T$$

where S_γ *is the soft-thresholding operator with parameter* γ *and* $\text{svd}(W) = U\sigma(W)V^T$ *is the singular value decomposition of* W*.*

Proof. See Theorem 6.4 in [23]. □

2.3 Pseudocontractive, Averaged, and Inverse Strongly Monotone Operators

In this section, we present the definitions and results needed to show that the ALRMC algorithm converges.

Definition 2.3.1. Suppose $\mathcal{H}$ is a Hilbert space. We say an operator N is nonexpansive if $\|N(x) - N(y)\| \leq \|x - y\|$ for all $x, y \in \mathcal{H}$.

Definition 2.3.2. Suppose $\mathcal{H}$ is a Hilbert space. We say that an operator T is α-averaged if T can be written as $T = (1 - \alpha)\,\mathrm{Id} + \alpha N$ where N is a nonexpansive operator and $\alpha \in (0, 1)$.

Definition 2.3.3. Suppose $\mathcal{H}$ is a Hilbert space. We say that an operator T is ν-pseudocontractive if T satisfies

$$\|T(x) - T(y)\|^2 \leq \|x - y\|^2 - \nu\|[\mathrm{Id} - T](x) - [\mathrm{Id} - T](y)\|^2$$

for all $x, y \in \mathcal{H}$.

Definition 2.3.4. Suppose $\mathcal{H}$ is a Hilbert space. We say that an operator T is σ-inverse strongly monotone if T satisfies

$$\langle T(x) - T(y), x - y\rangle \leq \sigma\|T(x) - T(y)\|^2$$

for all $x, y \in \mathcal{H}$.

Theorem 2.3.1. *(See Chapter 2 in [22] for (1) $\iff$ (2) and Lemma 2.1 in [2] for (2) $\iff$ (3)) The following are equivalent:*

(a) T is α-averaged.

(b) T is $\left(\frac{1-\alpha}{\alpha}\right)$-pseudocontractive.

(c) $\mathrm{Id} - T$ *is* $\left(\frac{1}{2\alpha}\right)$*-inverse strongly monotone.*

Proof. We begin by showing (1) $\Rightarrow$ (2). Suppose that T is α-averaged. Therefore we can write T as $T = (1-\alpha)\,\mathrm{Id} + \alpha N$ where N is a nonexpansive operator and $\alpha \in (0,1)$. We want to show that T is $\left(\frac{1-\alpha}{\alpha}\right)$-pseudocontractive, or equivalently, that

$$\|T(x) - T(y)\|^2 \leq \|x - y\|^2 - \left(\frac{1-\alpha}{\alpha}\right) \|[\mathrm{Id} - T](x) - [\mathrm{Id} - T](y)\|^2.$$

Start by analyzing the left-hand side of this inequality:

$$\begin{aligned}
\text{LHS} &= \|T(x) - T(y)\|^2 \\
&= \|[(1-\alpha)\,\mathrm{Id} + \alpha N](x) - [(1-\alpha)\,\mathrm{Id} + \alpha N](y)\|^2 \\
&= \|[(1-\alpha)\,\mathrm{Id} + \alpha N](x) - [(1-\alpha)\,\mathrm{Id} + \alpha N](y)\|^2 \\
&= \|(1-\alpha)(x-y) + \alpha(N(x) - N(y))\|^2 \\
&= \langle (1-\alpha)(x-y) + \alpha(N(x) - N(y)), (1-\alpha)(x-y) + \alpha(N(x) - N(y)) \rangle \\
&= (1-\alpha)^2 \|x-y\|^2 + \alpha(1-\alpha)\langle N(x) - N(y), x - y\rangle + \alpha \|N(x) - N(y))\|^2.
\end{aligned}$$

Now lets analyze the right-hand side:

$$\begin{aligned}
\text{RHS} &= \|x-y\|^2 - \left(\frac{1-\alpha}{\alpha}\right) \|[\mathrm{Id} - (1-\alpha)\,\mathrm{Id} - \alpha N](x) - [\mathrm{Id} - (1-\alpha)\,\mathrm{Id} - \alpha N](y)\|^2 \\
&= \|x-y\|^2 - \left(\frac{1-\alpha}{\alpha}\right) \|[\alpha\,\mathrm{Id} - \alpha N](x) - [\alpha\,\mathrm{Id} - \alpha N](y)\|^2 \\
&= \|x-y\|^2 - \alpha(1-\alpha)\|[\mathrm{Id} - N](x) - [\mathrm{Id} - N](y)\|^2 \\
&= \|x-y\|^2 - \alpha(1-\alpha)\Big\langle x - N(x) - y + N(y), x - N(x) - y + N(y) \Big\rangle \\
&= \|x-y\|^2 - \alpha(1-\alpha)\Big(\|x-y\|^2 - 2\langle x-y, N(x) - N(y)\rangle + \|N(x) - N(y)\|^2\Big) \\
&= (1 - \alpha(1-\alpha))\|x-y\|^2 + 2\alpha(1-\alpha)\langle x-y, N(x) - N(y)\rangle \\
&\qquad - \alpha(1-\alpha)\|N(x) - N(y)\|^2.
\end{aligned}$$

To prove the desired result, we now need only show that LHS $\leq$ RHS, or equivalently that $0 \leq$ RHS $-$ LHS:

$$\begin{aligned}
\text{RHS} - \text{LHS} &= \Big((1-\alpha(1-\alpha)) - (1-\alpha)^2\Big)\|x-y\|^2 \\
&\qquad + \Big(-\alpha(1-\alpha) - \alpha^2\Big)\|N(x)-N(y)\|^2 \\
&= \Big(1-\alpha+\alpha^2-1+2\alpha-\alpha^2\Big)\|x-y\|^2 \\
&\qquad + \Big(-\alpha+\alpha^2-\alpha^2\Big)\|N(x)-N(y)\|^2 \\
&= \alpha\|x-y\|^2 - \alpha\|N(x)-N(y)\|^2 \\
&= \alpha\Big(\|x-y\|^2 - \|N(x)-N(y)\|^2\Big) \\
&\geq 0
\end{aligned}$$

where the last inequality follows from the fact that $\alpha > 0$ and that N is nonexpansive. So we have proved the forward direction.

To prove (2) $\Rightarrow$ (1)., begin by assuming that T is $\left(\frac{1-\alpha}{\alpha}\right)$-pseudocontractive. We need to find a nonexpansive operator N such that $T = (1-\alpha)\,\mathrm{Id} - \alpha N$. We will see that this operator N is in fact

$$\begin{aligned}
N &= \left(1+\frac{1-\alpha}{\alpha}\right)T - \left(\frac{1-\alpha}{\alpha}\right)\mathrm{Id} \\
&= \left(\frac{1}{\alpha}\right)T - \left(\frac{1-\alpha}{\alpha}\right)\mathrm{Id}\,.
\end{aligned}$$

We only need to check that

$$\begin{aligned}
(1-\alpha)\,\mathrm{Id} + \alpha N &= (1-\alpha)\,\mathrm{Id} + \alpha\left(\left(\frac{1}{\alpha}\right)T - \left(\frac{1-\alpha}{\alpha}\right)\mathrm{Id}\right) \\
&= (1-\alpha)\,\mathrm{Id} + T - (1-\alpha)\,\mathrm{Id} \\
&= T
\end{aligned}$$

and that N is nonexpansive. To see this, let ν denote $\frac{1-\alpha}{\alpha}$ and note that because T is ν-pseudocontractive,

$$\|T(x) - T(y)\|^2 + \nu\|[\mathrm{Id} - T](x) - [\mathrm{Id} - T](y)\|^2 \leq \|x - y\|^2$$
$$(1+\nu)\|T(x) - T(y)\|^2 - 2\nu\langle x - y, T(x) - T(y)\rangle + \nu\|x - y\|^2 \leq \|x - y\|^2.$$

Denote the expression on left-hand side above by U. Now expand

$$\begin{aligned}
\|N(x) - N(y)\|^2 &= (1+\nu)^2\|T(x) - T(y)\|^2 - 2\nu(1+\nu)\langle T(x) - T(y), x - y\rangle \\
&\quad + \nu^2\|x - y\|^2 \\
&= (1+\nu)\Big((1+\nu)\|T(x) - T(y)\|^2 - 2\nu\langle T(x) - T(y), x - y\rangle \\
&\quad + \nu\|x - y\|^2 - \nu\|x - y\|^2\Big) + \nu^2\|x - y\|^2 \\
&= (1+\nu)U - \nu(1+\nu)\|x - y\|^2 + \nu^2\|x - y\|^2 \\
&= (1+\nu)U - \nu\|x - y\|^2 \\
&\leq (1+\nu)\|x - y\|^2 - \nu\|x - y\|^2, \text{using the above inequality} \\
&= \|x - y\|^2
\end{aligned}$$

so N is indeed nonexpansive, so we have proved (1) $\iff$ (2).

We now prove that (1) $\iff$ (3), starting by showing that (1) $\Rightarrow$ (3). Assume that T is α-averaged for some $\alpha \in (0, 1)$. Then there exists a non-expansive operator N such that $T = (1-\alpha)\,\mathrm{Id} + \alpha N$. Then the complement can be written as $G = \mathrm{Id} - T = \alpha(\mathrm{Id} - N)$. Note that the following equation holds between an operator T and its complement $G = \mathrm{Id} - T$:

$$\|x - y\|^2 - \|T(x) - T(y)\|^2 = 2\langle G(x) - G(y), x - y\rangle - \|G(x) - G(y)\|^2 \qquad \text{(op 1)}$$

because the right-hand side (RHS) of this expression can be written

$$\begin{aligned}
\text{RHS} &= 2\langle G(x) - G(y), x - y\rangle - \|G(x) - G(y)\|^2 \\
&= 2\langle G(x) - G(y), x - y\rangle - \langle G(x) - G(y), G(x) - G(y)\rangle \\
&= \langle G(x) - G(y), 2(x - y) - (G(x) - G(y))\rangle \\
&= \langle (x - T(x)) - (y - T(y)), 2(x - y) - (x - T(x)) + (y - T(y))\rangle \\
&= \langle (x - y) - (T(x) - T(y)), (x - y) + (T(x) - T(y))\rangle \\
&= \|x - y\|^2 - \|T(x) - T(y)\|^2.
\end{aligned}$$

Note that the left-hand side of (op 1) is non-negative if and only if T is non-expansive, therefore T is non-expansive if and only if

$$\begin{aligned}
0 &\leq 2\langle G(x) - G(y), x - y\rangle - \|G(x) - G(y)\|^2 \\
\frac{1}{2}\|G(x) - G(y)\|^2 &\leq \langle G(x) - G(y), x - y\rangle
\end{aligned}$$

i.e. the complement is $\frac{1}{2}$-inverse strongly monotone. Therefore, since we know we can write $G = \mathrm{Id} - T = \alpha(\mathrm{Id} - N)$, and $(\mathrm{Id} - N)$ is $\frac{1}{2}$-inverse strongly monotone, we know that $G = \alpha(\mathrm{Id} - N)$ is $\frac{1}{2\alpha}$-inverse strongly monotone.

To show (3) $\iff$ (1), assume $G = \mathrm{Id} - T$ is σ-inverse strongly monotone for $\sigma \in (1/2, \infty)$. Then let $\alpha = \frac{1}{2\sigma}$ and let $N = \mathrm{Id} - \frac{1}{\alpha}G$. Then $T = (1 - \alpha)\,\mathrm{Id} + \alpha N$, and because $\mathrm{Id} - N = \frac{1}{\alpha}G$, it is $\alpha\sigma$-inverse strongly monotone. Therefore N is non-expansive, and therefore T is α-averaged. □

Corollary 2.3.1. *The following are equivalent:*

(a) T is ν-pseudocontractive.

(b) T is $\left(\frac{1}{1+\nu}\right)$-averaged.

Proof. Using Theorem 2.3.1, set $\nu = \frac{1-\alpha}{\alpha}$ and the result follows from simple algebra.

□

Theorem 2.3.2. *If T_1 is α_1-averaged and T_2 is α_2-averaged with $\alpha_1, \alpha_2 \in [0, 1)$, then the composition $T_1 \circ T_2$ is*

$$\left(\frac{\alpha_1 + \alpha_2 - 2\alpha_1\alpha_2}{1 - \alpha_1\alpha_2}\right)\text{-}averaged.$$

Proof. See proof of Theorem 3 in [17]. □

Corollary 2.3.2. *If T_1 is ν_1-pseudocontractive and T_2 is ν_2-pseudocontractive, then the composition $T_1 \circ T_2$ is*

$$\frac{\nu_1\nu_2}{\nu_1 + \nu_2}\text{-}pseudocontractive$$

Proof. Using Corollary 2.3.1, we know that T_1 is $1/(1+\nu_1)$-averaged and T_2 is $1/(1+\nu_2)$-averaged. By Theorem 2.3.2, we have that $T_1 \circ T_2$ is averaged with parameter

$$\begin{aligned}\frac{\frac{1}{1+\nu_1} + \frac{1}{1+\nu_2} - \frac{2}{(1+\nu_1)(1+\nu_2)}}{1 - \frac{1}{(1+\nu_1)(1+\nu_2)}} &= \frac{\frac{1+\nu_2+1+\nu_1-2}{(1+\nu_1)(1+\nu_2)}}{\frac{(1+\nu_1)(1+\nu_2)-1}{(1+\nu_1)(1+\nu_2)}} \\ &= \frac{\nu_1 + \nu_2}{\nu_1\nu_2 + \nu_1 + \nu_2}.\end{aligned}$$

Now using Theorem 2.3.1, we know that $T_1 \circ T_2$ is pseudocontractive with parameter

$$\begin{aligned}\frac{1 - \frac{\nu_1+\nu_2}{\nu_1\nu_2+\nu_1+\nu_2}}{\frac{\nu_1+\nu_2}{\nu_1\nu_2+\nu_1+\nu_2}} &= \frac{\nu_1\nu_2 + \nu_1 + \nu_2 - \nu_1 - \nu_2}{\nu_1 + \nu_2} \\ &= \frac{\nu_1\nu_2}{\nu_1 + \nu_2},\end{aligned}$$

as was desired. □

Lemma 2.3.1. *("Concatenation" of pseudocontractive operators) Let $\mathcal{H} = \mathcal{H}_1 \times \mathcal{H}_2$ and suppose that $T\colon \mathcal{H} \to \mathcal{H}$ is given by $T(x) = (T_1(x_1), T_2(x_2))$, where we use the notation $T_1 = T|_{\mathcal{H}_1}$ and $T_2 = T|_{\mathcal{H}_2}$, and $x = (x_1, x_2)$ with $x_1 \in \mathcal{H}_1$ and $x_2 \in \mathcal{H}_2$. If T_1*

is ν_1-pseudocontractive and T_2 is ν_2-pseudocontractive, then T is $\min(\nu_1, \nu_2)$-pseudocontractive.

Proof. Assume without loss of generality that $\nu_1 \leq \nu_2$, so that $\min(\nu_1, \nu_2) = \nu_1$. Note that $\|x\|_{\mathcal{H}}^2 = \|x_1\|_{\mathcal{H}_1}^2 + \|x_2\|_{\mathcal{H}_2}^2$, so

$$\begin{aligned}
\|T(x) - T(y)\|_{\mathcal{H}}^2 &= \|T_1(x_1) - T_1(y_1)\|_{\mathcal{H}_1}^2 + \|T_2(x_2) - T_2(y_2)\|_{\mathcal{H}_2}^2 \\
&\leq \|x_1 - y_1\|_{\mathcal{H}_1}^2 - \nu_1\|(\mathrm{Id} - T_1)(x_1) - (\mathrm{Id} - T_1)(x_2)\|_{\mathcal{H}_1}^2 \\
&\qquad + \|x_2 - y_2\|_{\mathcal{H}_2}^2 - \nu_2\|(\mathrm{Id} - T_2)(x_2) - (\mathrm{Id} - T_2)(x_2)\|_{\mathcal{H}_2}^2 \\
&\leq \|x_1 - y_1\|_{\mathcal{H}_1}^2 - \nu_1\|(\mathrm{Id} - T_1)(x_1) - (\mathrm{Id} - T_1)(x_2)\|_{\mathcal{H}_1}^2 \\
&\qquad + \|x_2 - y_2\|_{\mathcal{H}_2}^2 - \nu_1\|(\mathrm{Id} - T_2)(x_2) - (\mathrm{Id} - T_2)(x_2)\|_{\mathcal{H}_2}^2 \\
&= \|x - y\|_{\mathcal{H}}^2 - \nu_1\|(\mathrm{Id} - T)(x) - (\mathrm{Id} - T)(x)\|_{\mathcal{H}}^2
\end{aligned}$$

and we are done. □

Theorem 2.3.3. *(See Theorem 3 in [5]) Suppose T is ν-pseudocontractive. An iterative update algorithm using T as its update operator converges, i.e. the sequence defined by a given starting point x_0 and updates given by $x_{k+1} = T(x_k)$ converges.*

Proof. Assume $\mathcal{H}$ is a finite dimensional Hilbert space, and that $z \in \mathcal{H}$ is a fixed point of a nonexpansive operator N. Let $\alpha \in (0,1)$, and define $T = (1-\alpha)\,\mathrm{Id} + \alpha N$ so that T is α-averaged, and

$$\begin{aligned} T(z) &= (1-\alpha)z + \alpha N(z) \\ &= (1-\alpha)z + \alpha z \\ &= z \end{aligned}$$

so z is also a fixed point of T. We need the identity (op 1) from the proof of Theorem 2.3.1

$$\|z - x^k\|^2 - \|T(z) - T(x^k)\|^2 = 2\langle G(z) - G(x^k), z - x^k\rangle - \|G(z) - G(x^k)\|^2$$

where $G = \mathrm{Id} - T$ as before. Because T is α-averaged, from Theorem 2.3.1 we know G is $\frac{1}{2\alpha}$-inverse strongly monotone. This now implies

$$\begin{aligned} \|z - x^k\|^2 - \|T(z) - T(x^k)\|^2 &= 2\langle G(z) - G(x^k), z - x^k\rangle - \|G(z) - G(x^k)\|^2 \\ \|z - x^k\|^2 - \|z - x^{k+1}\|^2 &= 2\langle G(z) - G(x^k), z - x^k\rangle - \|G(z) - G(x^k)\|^2 \\ \|z - x^k\|^2 - \|z - x^{k+1}\|^2 &\geq 2\left(\frac{1}{2\alpha}\right)\|G(z) - G(x^k)\|^2 - \|G(z) - G(x^k)\|^2 \\ \|z - x^k\|^2 - \|z - x^{k+1}\|^2 &\geq \left(\frac{1}{\alpha} - 1\right)\|G(z) - G(x^k)\|^2 \\ \|z - x^k\|^2 - \|z - x^{k+1}\|^2 &\geq \left(\frac{1}{\alpha} - 1\right)\|(\mathrm{Id} - T)(z) - (\mathrm{Id} - T)(x^k)\|^2 \\ \|z - x^k\|^2 - \|z - x^{k+1}\|^2 &\geq \left(\frac{1}{\alpha} - 1\right)\|x^k - x^{k+1}\|^2 \qquad \text{(op 2)} \end{aligned}$$

Note that $\left(\frac{1}{\alpha} - 1\right) > 0$, and hence

$$\|z - x^k\| \geq \|z - x^{k+1}\|$$

so the sequence $\{\|z - x^k\|\}$ is decreasing, and hence the sequence $\{x^k\}$ is bounded by $\|z\| + \|x^0\|$. Therefore $\{\|z - x^k\|\}$ convergences, and hence $\|z - x^k\|^2 - \|z - x^{k+1}\|^2 \to 0$, so by (op 2)

$$\|x^k - x^{k+1}\| \to 0$$

□

2.4 ALRMC Version A: Data Corrupted by Noise

The model we use is given by

$$Y = A + c\mathbb{1}_{1\times n} + Z$$

where $A \in \mathbb{R}^{m\times n}$ is a low-rank matrix, $c \in \mathbb{R}^m$ is the minimal affine shift, $\mathbb{1}_{1\times n} \in \mathbb{R}^{1\times n}$ is a row vector of all ones, and $Z \in \mathbb{R}^{m\times n}$ is the matrix of noise (low-magnitude, dense corruptions).

Given Y, our goal is to recover A and c. An optimization problem for this recovery can be formulated as

$$\underset{A,c}{\operatorname{argmin}} \left(\|A\|_{\mathrm{Sch}(1)} + \frac{\lambda_c}{2}\|c\|_2^2 + \frac{\lambda_z}{2}\|Y - A - c\mathbb{1}_{1\times n}\|_{\mathrm{vec}(2)}^2 \right) \tag{2.4.1}$$

for weights λ_c, $\lambda_z > 0$.

To solve this optimization problem, first denote the objective function as

$$h(A,c) = \|A\|_{\mathrm{Sch}(1)} + \frac{\lambda_c}{2}\|c\|_2^2 + \frac{\lambda_z}{2}\|Y - A - c\mathbb{1}_{1\times n}\|_{\mathrm{vec}(2)}^2,$$

and note that we can split the objective function into the two pieces

$$\begin{aligned} f(A,c) &= \frac{\lambda_c}{2}\|c\|_2^2 + \frac{\lambda_z}{2}\|Y - A - c\mathbb{1}_{1\times n}\|_{\mathrm{vec}(2)}^2 \\ g(A,c) &= \|A\|_{\mathrm{Sch}(1)} \end{aligned}$$

where $h = g + f$.

Lemma 2.4.1. *The functions $f, g\colon \mathcal{H} \to \mathbb{R}$ defined above are convex, proper, and continuous, where $\mathcal{H} = \mathbb{R}^{m\times n} \times \mathbb{R}^m$ is a Hilbert space.*

Proof. Note that all norms are convex, because if $\theta \in [0,1]$, then

$$\begin{aligned}\|\theta x + (1-\theta)y\| &\leq \|\theta x\| + \|(1-\theta)y\| \\ &= |\theta|\|x\| + |1-\theta|\|y\| \\ &= \theta\|x\| + (1-\theta)\|y\|\end{aligned}$$

Moreover, all norms are continous as well. Note now that each of the functions f and g is a weighted sum of norms with nonnegative weights. Both convexity and continuity are preserved under weighted sums with non-negative weights, so both f and g are convex and continuous. And f and g are obviously proper, as they take values in $\mathbb{R}$ as opposed to the extended real numbers. □

Lemma 2.4.2. *The function $f\colon \mathcal{H} \to \mathbb{R}$ define above is differentiable with a L-Lipschitz continuous gradient.*

Proof. Start by calculating

$$\nabla f = (\nabla_A f, \nabla_c f).$$

For notational and (later) computational efficiency, let us first define $J = -\lambda_z(Y - A - c\mathbb{1}_{1\times n})$. With this definition, note that

$$\begin{aligned}\nabla_A f &= -\lambda_z(Y - A - c\mathbb{1}_{1\times n}) \\ &= J \\ \nabla_c f &= \lambda_c c - \lambda_z \sum_{j=1}^{n} \begin{bmatrix} Y_{1,j} - A_{1,j} - c_1 \\ \vdots \\ Y_{m,j} - A_{m,j} - c_m \end{bmatrix} \\ &= \lambda_c c + J\mathbb{1}_{n\times 1}\end{aligned}$$

where $J\mathbb{1}_{n\times 1}$ denotes the vector formed by summing over the rows of J.

To calculate the Hessian, first we define how to represent elements in $\mathcal{H}$ as vectors. Define $\mathrm{vec}\colon \mathbb{R}^{m\times n} \to \mathbb{R}^{mn}$ by

$$\mathrm{vec}(M) = (M_{1,1}, \ldots, M_{1,n}, M_{2,1}, \ldots, M_{2,n}, \ldots, M_{m,1}, \ldots, M_{m,n}).$$

That is to say, $\mathrm{vec}(\cdot)$ takes a matrix in $\mathbb{R}^{m\times n}$ and gives the row-major representation in $\mathbb{R}^{mn}$. Now consider the objective function $h(x)$ instead defined for $x \in \mathbb{R}^{mn} \times \mathbb{R}^m$, where $x = (\mathrm{vec}(A), c)$. With this representation, the gradient is now a vector, and the Hessian can be represented as a matrix H, which is given below

$$H = \begin{bmatrix} & & \\ \lambda_z I_{mn} & & \lambda_z \begin{pmatrix} \mathbb{1}_{n\times 1} & & \\ & \ddots & \\ & & \mathbb{1}_{n\times 1} \end{pmatrix} \\ & & \\ \lambda_z \begin{pmatrix} \mathbb{1}_{1\times n} & & \\ & \ddots & \\ & & \mathbb{1}_{1\times n} \end{pmatrix} & & (\lambda_c + \lambda_z n) I_m \end{bmatrix}$$

Note that H is constant with respect to $x = (\mathrm{vec}(A), c)$, so $\|H\|_{2\to 2}$ is constant with respect to x, and hence $\|\nabla f(x) - \nabla f(y)\| \leq \|H\|_{2\to 2}\|x - y\|$ where $\|H\|_{2\to 2}$ is the spectral norm of H. Hence f has a L-Lipschitz continuous gradient where $L = \|H\|_{2\to 2}$. A closed form expression for $\|H\|_{2\to 2}$ is given in Appendix A. □

Psuedocode for the algorithm is given in Figure 2.1 and for proof of convergence, see Appendix D.

Lemma 2.4.3. *The objective function h is coercive.*

Proof. Note that if $\|(A, c)\| \to \infty$, then either $\|A\| \to \infty$ or $\|c\| \to \infty$. Then it is clear that $\|A\|_{\mathrm{Sch}(1)} + \frac{\lambda_c}{2}\|c\|_2^2 + \frac{\lambda_z}{2}\|Y - A - c\mathbb{1}_{1\times n}\|_{\mathrm{vec}(2)}^2 \to \infty$ by the equivalence of norms. □

Algorithm 2.1: ALRMCvA

Input: Data matrix Y, weights λ_c, $\lambda_z > 0$, proximal step size γ_2

Output: A and c

$[m, n] = \texttt{size}(Y)$
Initialize $A = 0$, $c = 0$
Compute L according to the formula in Appendix A, and initialize $\gamma_1 = 1/L$
while stopping criterion not met **do**
    $J = -\lambda_z \left(Y - A - c\mathbb{1}_{1\times n}\right)$
    $A_{\texttt{inter}} = A - \gamma_1 J$
    $c_{\texttt{inter}} = c - \gamma_1(\lambda_c c - J\mathbb{1}_{n\times 1}))$

    $[U, S, V] = \texttt{svd}(A_{\texttt{inter}})$
    $S_{\texttt{new}} = \mathcal{S}_{\gamma_2}(S)$
    $A = US_{\texttt{new}}V^{\mathrm{T}}$
    $c = c_{\texttt{inter}}$
end

Figure 2.1: The ALRMCvA Algorithm

2.5 ALRMC Version B: Data Corrupted by Noise and Errors

The model we use is given by

$$Y = A + E + c\mathbb{1}_{1\times n} + Z$$

where $A \in \mathbb{R}^{m\times n}$ is a low-rank matrix, $E \in \mathbb{R}^{m\times n}$ is the matrix of errors (high-magnitude, sparse corruptions at unknown locations), $c \in \mathbb{R}^m$ is the minimal affine shift, $\mathbb{1}_{1\times n} \in \mathbb{R}^{1\times n}$ is a row vector of all ones, and $Z \in \mathbb{R}^{m\times n}$ is the matrix of noise (low-magnitude, dense corruptions).

Given Y, our goal is to recover A, E, and c. An optimization problem for this recovery can be formulated as

$$\underset{A,E,c}{\operatorname{argmin}} \left(\|A\|_{\mathrm{Sch}(1)} + \lambda_e \|E\|_{\mathrm{vec}(1)} + \frac{\lambda_c}{2}\|c\|_2^2 + \frac{\lambda_z}{2}\|Y - A - E - c\mathbb{1}_{1\times n}\|_{\mathrm{vec}(2)}^2 \right) \quad (2.5.1)$$

for weights λ_c, λ_e, and $\lambda_z > 0$.

To solve this optimization problem, first denote the objective function as

$$h(A, E, c) = \|A\|_{\mathrm{Sch}(1)} + \lambda_e \|E\|_{\mathrm{vec}(1)} + \frac{\lambda_c}{2}\|c\|_2^2 + \frac{\lambda_z}{2}\|Y - A - E - c\mathbb{1}_{1\times n}\|_{\mathrm{vec}(2)}^2,$$

and note that we can split the objective function into the two pieces

$$\begin{aligned} f(A, E, c) &= \frac{\lambda_c}{2}\|c\|_2^2 + \frac{\lambda_z}{2}\|Y - A - E - c\mathbb{1}_{1\times n}\|_{\mathrm{vec}(2)}^2 \\ g(A, E, c) &= \|A\|_{\mathrm{Sch}(1)} + \lambda_e \|E\|_{\mathrm{vec}(1)} \end{aligned}$$

where $h = g + f$.

With this decomposition, note that our problem is an instance of Problem 1.1 in [3] with the Hilbert space $\mathcal{H} = \mathbb{R}^{m\times n} \times \mathbb{R}^{m\times n} \times \mathbb{R}^m$, as f and g are proper (because they are both real-valued), lower semi-continuous (because they are both continuous) convex functions (because all norms are convex functions), where f is differentiable on $\mathcal{H}$ with a L-Lipschitz continuous gradient.

To see why f has a L-Lipschitz continuous gradient, we start by calculating

$$\nabla f = (\nabla_A f, \nabla_E f, \nabla_c f).$$

Define

$$J = -\lambda_z(Y - A - E - c\mathbb{1}_{1\times n}).$$

With this definition, note that

$$\begin{aligned}
\nabla_A f &= -\lambda_z(Y - A - E - c\mathbb{1}_{1\times n}) \\
&= J \\
\nabla_E f &= -\lambda_z(Y - A - E - c\mathbb{1}_{1\times n}) \\
&= J \\
\nabla_c f &= \lambda_c c - \lambda_z \sum_{j=1}^{n} \begin{bmatrix} Y_{1,j} - A_{1,j} - E_{1,j} - c_1 \\ \vdots \\ Y_{m,j} - A_{m,j} - E_{m,j} - c_m \end{bmatrix} \\
&= \lambda_c c + J\mathbb{1}_{n\times 1}
\end{aligned}$$

where $J\mathbb{1}_{n\times 1}$ denotes the vector formed by summing over the second dimension (the rows) of J.

The Hessian is calculated as

$$H = \begin{bmatrix} & & & \\ \lambda_z I_{mn} & \lambda_z I_{mn} & \lambda_z \begin{bmatrix} \mathbb{1}_{n\times 1} & & \\ & \ddots & \\ & & \mathbb{1}_{n\times 1} \end{bmatrix} \\ & & & \\ \lambda_z I_{mn} & \lambda_z I_{mn} & \lambda_z \begin{bmatrix} \mathbb{1}_{n\times 1} & & \\ & \ddots & \\ & & \mathbb{1}_{n\times 1} \end{bmatrix} \\ & & & \\ \lambda_z \begin{bmatrix} \mathbb{1}_{1\times n} & & \\ & \ddots & \\ & & \mathbb{1}_{1\times n} \end{bmatrix} & \lambda_z \begin{bmatrix} \mathbb{1}_{1\times n} & & \\ & \ddots & \\ & & \mathbb{1}_{1\times n} \end{bmatrix} & (\lambda_c + n\lambda_z) I_m \end{bmatrix}$$

Note that H is constant with respect to $x = (\mathrm{vec}(A), \mathrm{vec}(E), c)$, so $\|H\|_{2\to 2}$ is

Algorithm 2.2: ALRMCvB

Input: Data matrix Y, weights λ_c, λ_e, $\lambda_z > 0$, proximal step sizes γ_2, γ_3
Output: A, E, and c

$[m, n] = \texttt{size}(Y)$
Initialize $A = 0$, $E = 0$, $c = 0$
Compute L according to the formula in Appendix A
$\gamma_1 = 1/L$
while stopping criterion not met **do**
    $J = -\lambda_z \left(Y - A - E - c\mathbb{1}_{1\times n}\right)$
    $A_{\texttt{inter}} = A - \gamma_1 J$
    $E_{\texttt{inter}} = E - \gamma_1 J$
    $c_{\texttt{inter}} = c - \gamma_1(\lambda_c c - J\mathbb{1}_{n\times 1})$

    $[U, S, V] = \texttt{svd}(A_{\texttt{inter}})$
    $S_{\texttt{new}} = \mathcal{S}_{\gamma_2}(S)$
    $A = US_{\texttt{new}}V^{\mathrm{T}}$
    $E = \mathcal{S}_{\gamma_3}(E_{\texttt{inter}})$
    $c = c_{\texttt{inter}}$
end

Figure 2.2: The ALRMCvB Algorithm

constant with respect to x, and hence $\|\nabla f(x) - \nabla f(y)\| \leq \|H\|_{2\to 2}\|x - y\|$ where $\|H\|_{2\to 2}$ is the spectral norm of H. Hence f has a L-Lipschitz continuous gradient where $L = \|H\|_{2\to 2}$. This is calculated in Appendix A.

Psuedocode for the algorithm is given in Figure 2.2, and proof of convergence is in Appendix D.

2.6 ALRMC (General Version): Data Corrupted by Noise, Errors, and Erasures

The model we use is given by

$$Y = A|_\Omega + E + (c\mathbb{1}_{1\times n})|_\Omega + Z|_\Omega$$

where $\Omega \subset \mathbb{R}^{m\times n}$ is the support of the matrix $Y \in \Omega$, i.e. Ω^{C} is the support of the erasures (high-magnitude, sparse corruptions or missing entries with known locations),

$A \in \mathbb{R}^{m\times n}$ is a low-rank matrix, $E \in \Omega$ is the matrix of errors (high-magnitude, sparse corruptions at unknown locations), $c \in \mathbb{R}^m$ is the minimal affine shift, $\mathbb{1}_{1\times n} \in \mathbb{R}^{1\times n}$ is a row vector of all ones, and $Z \in \mathbb{R}^{m\times n}$ is the matrix of noise (low-magnitude, dense corruptions). $X|_\Omega$ denotes the restriction of a matrix $X \in \mathbb{R}^{m\times n}$ to the space Ω. With slight abuse of notation, we will say that a pair of indices $(i,j) \in \Omega$ if the $(i,j)^{\text{th}}$ coordinate of the data matrix Y is not an erasure. Note that Ω is a Hilbert space with the inner product

$$\langle X, Y\rangle = \sum_{(i,j)\in\Omega} X_{i,j}Y_{i,j}.$$

Given Y and Ω, our goal is to recover A, E, and c. An optimization problem for this recovery can be formulated as

$$\underset{A,E,c}{\operatorname{argmin}} \left(\|A\|_{\text{Sch}(1)} + \lambda_e \|E\|_{\text{vec}(1)} + \frac{\lambda_z}{2}\|Y - A|_\Omega - E - (c\mathbb{1}_{1\times n})|_\Omega\|^2_{\text{vec}(2)} + \frac{\lambda_c}{2}\|c\|_2^2 \right) \tag{2.6.1}$$

for weights λ_e, λ_z, $\lambda_c > 0$. We denote this objective function by $h\colon \mathcal{H} \to \mathbb{R}$ where $\mathcal{H} = \mathbb{R}^{m\times n} \times \Omega \times \mathbb{R}^m$ is a Hilbert space. Here, $\|X\|_{\text{Sch}(1)}$ denotes the Schatten-1 norm (also called the nuclear norm), and $\|X\|_{\text{vec}(p)}$ denotes the entrywise-p norm over the space Ω, given by

$$\|X\|_{\text{vec}(p)} = \left(\sum_{(i,j)\in\Omega} |X_{i,j}|^p \right)^{1/p}.$$

To use the proximal forward-backward splitting algorithm to solve this optimization problem we first split the objective function h into the two pieces

$$\begin{aligned} f(A,E,c) &= \frac{\lambda_z}{2}\|Y - A|_\Omega - E - (c\mathbb{1}_{1\times n})|_\Omega\|^2_{\text{vec}(2)} + \frac{\lambda_c}{2}\|c\|_2^2 \\ g(A,E,c) &= \|A\|_{\text{Sch}(1)} + \lambda_e\|E\|_{\text{vec}(1)} \end{aligned}$$

where $h = g + f$.

With this decomposition, note that f and g are proper (because they are both real-valued), lower semi-continuous (because they are both continuous), and convex functions (because all norms are convex functions). Moreover, f is differentiable on $\mathcal{H}$ with a L-Lipschitz continuous gradient. To see why f has a L-Lipschitz continuous gradient, it is first useful to reformulate the problem in a "vectorized" manner, which we can do because f does not depend on the matrix structure on $\mathbb{R}^{m\times n}$. Define $\operatorname{vec}\colon \mathbb{R}^{m\times n} \to \mathbb{R}^{mn}$ as before. Similarly, we can define $\operatorname{vec}\colon \Omega \to \mathbb{R}^{mn-s}$ by ordering the non-erased entries in lexicographical order, where $s = |\Omega^{\mathrm{C}}|$ is the number of erased entries of the data matrix Y.

Define

$$J = -\lambda_z \left(Y - A|_\Omega - E - (c\mathbb{1}_{1\times n})|_\Omega\right) \in \Omega.$$

With this definition, note that the gradient of f can be computed as

$$\nabla f = (\nabla_A f, \nabla_E f, \nabla_c f)$$

where

$$\begin{aligned}
\nabla_A f &= -\lambda_z \left(\operatorname{ext}(Y) - A - \operatorname{ext}(E) - c\mathbb{1}_{1\times n}\right) \\
&= \operatorname{ext}(J) \\
\nabla_E f &= -\lambda_z \left(Y - A|_\Omega - E - (c\mathbb{1}_{1\times n})|_\Omega\right) \\
&= J \\
\nabla_c f &= \lambda_c c - \lambda_z \begin{bmatrix} \sum_{\{j\colon (1,j)\in\Omega\}} Y_{1,j} - A_{1,j} - E_{1,j} - c_1 \\ \vdots \\ \sum_{\{j\colon (m,j)\in\Omega\}} Y_{m,j} - A_{m,j} - E_{m,j} - c_m \end{bmatrix} \\
&= \lambda_c c + \operatorname{ext}(J)\mathbb{1}_{n\times 1}.
\end{aligned}$$

where ext: $\Omega \to \mathbb{R}^{m\times n}$ extends a matrix with partially erased entries in Ω to a matrix in $\mathbb{R}^{m\times n}$ by filling the erased entries with 0. And of course, in the vector formulation we have $\nabla \mathrm{vec}(f) = (\mathrm{vec}(\nabla_A f), \mathrm{vec}(\nabla_E f), \mathrm{vec}(\nabla_c f))$.

Using the vector formulation, the Hessian can be represented as a matrix H. First we need some notation: Let $\chi_\Omega = [\chi_\Omega(i,j)]_{i,j} \in \{0,1\}^{m\times n}$ where

$$\chi_\Omega(i,j) = \begin{cases} 1, & (i,j) \in \Omega \\ 0, & (i,j) \notin \Omega \end{cases}$$

And we need to define $\omega = \mathrm{vec}(\chi_\Omega)$ and

$$S = \begin{bmatrix} \mathbb{1}_{1\times n} & & \\ & \ddots & \\ & & \mathbb{1}_{1\times n} \end{bmatrix}.$$

Then we compute

$$H = \lambda_z \begin{bmatrix} \mathrm{Diag}(\omega) & I_{mn}[:,\omega] & \omega \mathbb{1}_{1\times m} \bigodot S^{\mathrm{T}} \\ I_{mn}[\omega,:] & I_{mn-s} & I_{mn}[\omega,:]S^{\mathrm{T}} \\ (\omega \mathbb{1}_{1\times m})^{\mathrm{T}} \bigodot S & S I_{mn}[:,\omega] & \frac{\lambda_c}{\lambda_z} I_m - \mathrm{Diag}(\chi_\Omega \mathbb{1}_{n\times 1}) \end{bmatrix}$$

Here, Diag: $\mathbb{R}^k \to \mathbb{R}^{k\times k}$ takes a vector and returns the diagonal matrix with that vector along the diagonal, and $\bigodot$: $\mathbb{R}^{k\times k} \times \mathbb{R}^{k\times k} \to \mathbb{R}^{k\times k}$ is the entrywise (Hadamard) product. $M[:,\omega]$ denotes logical indexing of M using the logical vector $\omega \in \{0,1\}^{mn}$, i.e. $M[:,\omega]$ is the matrix M with all columns corresponding to zero elements of ω omitted, and $M[\omega,:]$ is the matrix M with all rows corresponding to the zero elements of ω omitted.

Note that H is constant with respect to $x = (A, E, c)$, and hence $\|\nabla f(x) - \nabla f(y)\| \le \|H\|_{2\to 2}\|x - y\|$ where $\|H\|_{2\to 2}$ is the spectral norm of H. Hence f has a

L-Lipschitz continuous gradient where $L = \|H\|_{2\to 2}$. In fact, the singular values of H have a closed-form solution:

Lemma: Define $q_i = n - \pi_i(\chi_\Omega \mathbb{1}_{n\times 1})$, i.e. q_i is the difference between the number of data points and the number erasures among all data points in the i^{th} coordinate. Now let $q_{\max} = n - \min_{i=1,\dots m}(\pi_i(\chi_\Omega \mathbb{1}_{n\times 1}))$, hence $q_{\max} \geq q_i$ for all i. The singular values of H are

$$\sigma_{2i-1}, \sigma_{2i} = \sqrt{\frac{\lambda_z^2(q_i+2)^2 + 2q_i\lambda_c\lambda_z + \lambda_c^2 \pm ((q_i+2)\lambda_z + \lambda_c)R_i}{2}},$$

$$\text{for } i = 1, \dots, m$$

$$\sigma_i = 2\lambda_z \quad \text{for } m(n-1) - s \text{ of the } (mn + (mn-s) + m) \text{ total singular values}$$

$$\sigma_i = 0 \quad \text{for } mn \text{ of the } (mn + (mn-s) + m) \text{ total singular values}$$

where $R_i = \sqrt{\lambda_z^2(q_i+2)^2 + 2(q_i-2)\lambda_c\lambda_z + \lambda_c^2}$. Moreover, define

$$L_{\max} = \sqrt{\frac{\lambda_z^2(q_{\max}+2)^2 + 2q_{\max}\lambda_c\lambda_z + \lambda_c^2 \pm ((q_{\max}+2)\lambda_z + \lambda_c)Q}{2}}$$

where

$$Q = \sqrt{\lambda_z^2(q_{\max}+2)^2 + 2(q_{\max}-2)\lambda_c\lambda_z + \lambda_c^2}$$

and then we have that $\|H\|_{2\to 2} \leq L_{\max}$.

So the above lemma establishes that $L_{\max}$ is *a* Lipschitz constant for f (but perhaps not *the smallest* Lipschitz constant).

Lemma 2.6.1. *The ALRMC update operator $T \circ (\mathrm{Id} - \gamma_1 \nabla f)$ is $(1 - \frac{\gamma_1 L}{2})$-pseudocontractive, where $T(A, E, c) = (\mathrm{prox}_{\gamma_2 g_A}(A), \mathrm{prox}_{\gamma_3 g_E}(E), c)$ and where $g_A = g|_{\mathbb{R}^{m\times n}}$ and $g_E = g|_\Omega$.*

Proof. From Proposition 3 in [5], we know that $\mathrm{Id} - \gamma_1 \nabla f$ is $(\frac{2}{\gamma_1 L} - 1)$-pseudocontractive, where ∇f is L-Lipschitz continuous. Proposition 4 in [5] tells us that $\mathrm{prox}_{\gamma g}$ is

Algorithm 2.3: ALRMC

Input: Data matrix Y, indicator of support of the data matrix χ_Ω, weights λ_c, λ_e, $\lambda_z > 0$, proximal step sizes γ_2, γ_3

Output: A, E, and c

$[m, n] = \texttt{size}(Y)$
Initialize $A = 0$, $E = 0$, $c = 0$
Compute L according to the formula in Appendix A, and initialize $\gamma_1 = 1/L$
while stopping criterion not met **do**
    $J_{\texttt{ext}} = -\lambda_z \chi_\Omega \bigodot (Y - A - E - c\mathbb{1}_{1\times n})$
    $A_{\texttt{inter}} = A - \gamma_1 J_{\texttt{ext}}$
    $E_{\texttt{inter}} = E - \gamma_1 J_{\texttt{ext}}$
    $c_{\texttt{inter}} = c - \gamma_1(\lambda_c c - J_{\texttt{ext}}\mathbb{1}_{n\times 1})$

    $[U, S, V] = \texttt{svd}(A_{\texttt{inter}})$
    $S_{\texttt{new}} = \mathcal{S}_{\gamma_2}(S)$
    $A = U S_{\texttt{new}} V^{\mathrm{T}}$
    $E = \mathcal{S}_{\gamma_3}(E_{\texttt{inter}})$
    $c = c_{\texttt{inter}}$
end

Figure 2.3: The ALRMCvC Algorithm

1-pseudocontractive. Using the fact that the identity operator is clearly 1-pseudocontractive, applying the above lemma twice yields that T is 1-pseudocontractive as well. Converting this to the notion of α-averaged, note that the following are equivalent:

$$\left(\frac{2}{\gamma_1 L} - 1\right)\text{-pseudocontractive}$$
$$\left(\frac{1 - \gamma_1 L/2}{\gamma_1 L/2}\right)\text{-pseudocontractive}$$
$$\left(\frac{\gamma_1 L}{2}\right)\text{-averaged}$$

and that

$$\text{1-pseudocontractive}$$
$$\left(\frac{1 - 1/2}{1/2}\right)\text{-pseudocontractive}$$
$$(1/2)\text{-averaged.}$$

Therefore, the composition $T \circ (\mathrm{Id} - \gamma_1 \nabla f)$ is

$$\left(\frac{\gamma_1 L/2 + 1/2 - 2(\gamma_1 L/2)(1/2)}{1 - (\gamma_1 L/2)(1/2)}\right)\text{-averaged}$$
$$\left(\frac{\gamma_1 L + 1 - \gamma_1 L}{2 - \gamma_1 L/2}\right)\text{-averaged}$$
$$\left(\frac{1}{2 - \gamma_1 L/2}\right)\text{-averaged}$$

and hence the composition $T \circ (\mathrm{Id} - \gamma_1 \nabla f)$ is

$$\left(\frac{1 - \frac{1}{2-\gamma_1 L/2}}{\frac{1}{2-\gamma_1 L/2}}\right)\text{-pseudocontractive}$$
$$(2 - \gamma_1 L/2 - 1)\text{-pseudocontractive}$$
$$(1 - \gamma_1 L/2)\text{-pseudocontractive.}$$

□

Theorem 2.6.1. *The ALRMC algorithm converges to a solution* (A^*, E^*, c^*) *to the problem* $\min_{(A,E,c)}(h(A, E, c))$.

Proof. It follows that ALRMC converges because the above lemma establishes that its update operator is ν-pseudocontractive, and the Krasnoselskii-Mann Theorem [12] establishes weak convergence of iterative algorithms with averaged updates, and hence for iterative algorithms with pseudocontractive updates as well. And because $\mathcal{H}$ is finite dimensional, this establishes strong convergence as well. □

2.7 An Extension: Simple Greedy Affine Low-Rank Matrix Completion (SGALRMC)

This section describes a simple greedy extention of the ALRMC algorithm. The ALRMC algorithm is used iteratively, and at the end of each iteration the indicator

Algorithm 2.4: SGALRMC

Input: Data matrix Y, indicator of support of the data matrix χ_Ω, weights λ_c, λ_e, $\lambda_z > 0$, proximal step sizes γ_2, γ_3

Output: A, E, and c

$[m, n] = \texttt{size}(Y)$
Initialize $A = 0$, $E = 0$, $c = 0$
Compute L according to the formula in Appendix A, and initialize $\gamma_1 = 1/L$
while Greedy stopping criterion not met **do**
    while ALRMC stopping criterion not met **do**
        $J_{\texttt{ext}} = -\lambda_z \chi_\Omega \odot (Y - A - E - c\mathbb{1}_{1\times n})$
        $A_{\texttt{inter}} = A - \gamma_1 J_{\texttt{ext}}$
        $E_{\texttt{inter}} = E - \gamma_1 J_{\texttt{ext}}$
        $c_{\texttt{inter}} = c - \gamma_1(\lambda_c c - J_{\texttt{ext}}\mathbb{1}_{n\times 1})$

        $[U, S, V] = \texttt{svd}(A_{\texttt{inter}})$
        $S_{\texttt{new}} = \mathcal{S}_{\gamma_2}(S)$
        $A = U S_{\texttt{new}} V^{\mathrm{T}}$
        $E = \mathcal{S}_{\gamma_3}(E_{\texttt{inter}})$
        $c = c_{\texttt{inter}}$
    end
    $\texttt{threshold} = \texttt{percentile}(E, \alpha)$
    $\chi_\Omega = \chi_\Omega - (E \geq \texttt{threshold})$
end

Figure 2.4: The SGALRMC Algorithm

matrix χ_Ω is updated to now exclude the highest α^{th}-percentile of the errors E from the previous iteration. That is to say, the largest errors from each previous iteration are treated as erasures in the next iteration. The hope is that the algorithm can better identify the matrices A, E, and the vector c if it is allowed this confidence that the most egrigious errors and be assumed to be 100% certain to be errors for future iterations.

Algorithm 2.5: FGALRMC

Input: Data matrix Y, initial indicator of support of the data matrix χ_Ω, weights λ_c, λ_e, $\lambda_z > 0$, proximal step sizes γ_2, γ_3, the burn-in period b, the number of iterations between erasure updates w

Output: A, E, c, and χ_Ω

$[m, n] = \texttt{size}(Y)$
Initialize $A = 0$, $E = 0$, $c = 0$
Compute L according to the formula in Appendix A, and initialize $\gamma_1 = 1/L$
while Greedy stopping criterion not met **do**
  while ALRMC stopping criterion not met **do**
    $J_{\texttt{ext}} = -\lambda_z \chi_\Omega \odot (Y - A - E - c\mathbb{1}_{1\times n})$
    $A_{\texttt{inter}} = A - \gamma_1 J_{\texttt{ext}}$
    $E_{\texttt{inter}} = E - \gamma_1 J_{\texttt{ext}}$
    $c_{\texttt{inter}} = c - \gamma_1(\lambda_c c - J_{\texttt{ext}}\mathbb{1}_{n\times 1})$

    $[U, S, V] = \texttt{svd}(A_{\texttt{inter}})$
    $S_{\texttt{new}} = \mathcal{S}_{\gamma_2}(S)$
    $A = U S_{\texttt{new}} V^{\mathrm{T}}$
    $E = \mathcal{S}_{\gamma_3}(E_{\texttt{inter}})$
    $c = c_{\texttt{inter}}$
  end
  $\texttt{threshold} = \texttt{percentile}(E, \alpha)$
  $\chi_\Omega = \chi_\Omega - (E \geq \texttt{threshold})$
end

Figure 2.5: The FGALRMC Algorithm

2.8 Fast Greedy Affine Low-Rank Matrix Completion (FGALRMC)

This section describes a fast greedy extention of the ALRMC algorithm. In this version, the indicator matrix χ_Ω is periodically updated to exclude the highest α^{th}-percentile of the errors E. That is to say, the largest errors from previous iterations are treated as erasures after this periodic update. The hope is that the algorithm can better identify the matrices A, E, and the vector c if it is allowed to assume the most egregious errors are 100% certain to be errors, at which point the algorithm can more effectively deal with them by treating them as erasures.

To analyze the FGALRMC algorithm, we start by defining $\mathcal{H} = \mathbb{R}^{m\times n} \times \mathbb{R}^{m\times n} \times \mathbb{R}^m \times \mathbb{R}^{m\times n}$. We will use $x = (A, E, c, \chi_\Omega)$ to denote a point in $\mathcal{H}$. We will denote the gradient descent operator by D, the proximal update operator by P, and the greedy

update operator by G, all defined below:

$$D(A, E, c, \chi_\Omega) = ([\mathrm{Id} - \gamma_1 \nabla f](A, E, c), \chi_\Omega)$$
$$P(A, E, c, \chi_\Omega) = (\mathrm{prox}_{\gamma_2 g_A}(A), \mathrm{prox}_{\gamma_3 g_E}(E), c, \chi_\Omega)$$
$$G(A, E, c, \chi_\Omega) = (A, E, c, \chi_\Omega \bigodot (E < t)),$$

where t is some thershold,

$$(E < t) = \left[\begin{cases} 1, & \text{if } E_{ij} < 1 \\ 0, & \text{if } E_{ij} \geq 1 \end{cases} \right]$$

and $g_A = g|_{\mathbb{R}^{m \times n}}$ and $g_E = g|_{\mathbb{R}^{m \times n}}$ denote restrictions of the function g to the respective pieces over the variables A and E. If we let T denote $P \circ D$, then the FGALRMC update operator U can be written as $G \circ T \circ \cdots \circ T$, where the operator T is applied w times. Note that initially T is applied b times, but this does not change the analysis.

Lemma 2.8.1. *The operator D defined above is $(\frac{2}{\gamma_1 L} - 1)$-pseudocontractive, assuming that f is convex with L-Lipschitz gradient and $0 < \gamma_1 < \frac{2}{L}$.*

Proof. By Proposition 3 in [5], we know $[\mathrm{Id} - \gamma_1 \nabla f]$ is $(\frac{2}{\gamma_1 L} - 1)$-pseudocontractive, and it is obvious that Id is 1-pseudocontractive. Therefore, by Lemma 2.3.1, because $1 - \gamma_1 L/2 < 1$ we are done. □

Lemma 2.8.2. *The operator P defined above is 1-pseudocontractive.*

Proof. By Proposition 4 in [5], we know all proximal operators are 1-pseudocontractive, and it is obvious that Id is 1-pseudocontractive. Therefore, by Lemma 2.3.1 we are done. □

Lemma 2.8.3. *The operator G defined above is 1-pseudocontractive.*

Proof. Note that G is a projection operator onto a convex set. Therefore, by Proposition 2 in [5], we are done. □

Theorem 2.8.1. *The FGALRMC update operator is pseudocontractive, and hence the algorithm converges.*

Proof. Follows from Corollary 2.3.2 and Theorem 2.3.3. □

Algorithm 2.6: ALRMC with Nesterov Acceleration and Continuation

Input: Data matrix Y, indicator of support of the data matrix χ_Ω, weights λ_c, λ_e, $\lambda_z > 0$, proximal step sizes γ_2, $\gamma_3 > 0$, continuation parameters $\kappa_1 > 1$, $\kappa_2 \gg 1$, $\kappa_3 > 1$

Output: A, E, and c

$[m, n] = \texttt{size}(Y)$
Initialize $A = A_{\texttt{hold}} = 0$, $E = E_{\texttt{hold}} = 0$, $c = c_{\texttt{hold}} = 0$
Initialize $\lambda_z = \kappa_1 / \|Y \odot \chi_\Omega\|_{2\to 2}$ and $\overline{\lambda_z} = \kappa_2 \lambda_z$
Compute L according to the formula in Appendix A, and initialize $\gamma_1 = 1/L$
Initialize $t = 1$, $t_{\texttt{hold}} = 1$, and $\eta = (t_{\texttt{hold}} - 1)/t$
while stopping criterion not met **do**
    $\widetilde{A} = A + \eta(A - A_{\texttt{hold}})$
    $\widetilde{E} = E + \eta(E - E_{\texttt{hold}})$
    $\widetilde{c} = c + \eta(c - c_{\texttt{hold}})$

    $J_{\texttt{ext}} = -\lambda_z \chi_\Omega \odot \left(Y - \widetilde{A} - \widetilde{E} - \widetilde{c}\mathbb{1}_{1\times n}\right)$
    $A_{\texttt{inter}} = \widetilde{A} - \gamma_1 J_{\texttt{ext}}$
    $E_{\texttt{inter}} = \widetilde{E} - \gamma_1 J_{\texttt{ext}}$
    $c_{\texttt{inter}} = \widetilde{c} - \gamma_1(\lambda_c \widetilde{c} - J_{\texttt{ext}}\mathbb{1}_{n\times 1})$

    $[U, S, V] = \texttt{svd}(A_{\texttt{inter}})$
    $S_{\texttt{new}} = \mathcal{S}_{\gamma_2}(S)$
    $A_{\texttt{hold}} = A$
    $A = U S_{\texttt{new}} V^{\mathrm{T}}$
    $E_{\texttt{hold}} = E$
    $E = \mathcal{S}_{\gamma_3}(E_{\texttt{inter}})$
    $c_{\texttt{hold}} = c$
    $c = c_{\texttt{inter}}$

    $\lambda_z = \min(\kappa_3 \lambda_z, \overline{\lambda_z})$
    $t_{\texttt{hold}} = t$
    $t = (1 + \sqrt{1 + 4t^2})/2$
    $\eta = (t_{\texttt{hold}} - 1)/t$
end

Figure 2.6: ALRMC with Nesterov Acceleration and Continuation

2.9 Implementation Notes

2.9.1 Pseudocode

Pseudocode for the ALRMC algorithm with Nesterov acceleration and continuation is given in Figure 2.6.

2.10 Numerical Results

This section gives numerical results on synthetic data. The experiments are all testing Algorithm 2.6 (The general version of the ALRMC algorithm using both Nesterov acceleration and continuation). For all experiments, $m = n = 100$ is chosen, so the low-rank matrix $A \in \mathbb{R}^{100\times 100}$ is a square matrix.

2.10.1 Experiment 1

This experiment tests the performance of the algorithm over different values probabilities of errors in the data and different dimensions of the affine subspaces. The parameter p_{err} denotes the probability of error in the data, and we will use r to denote the rank of the low-rank matrix A. A is generated by first generating matrices M_1, M_2 with elements taken from the normal distribution $\mathcal{N}(m, r)$, and then $A = M_1 M_2^{\mathrm{T}}$. E is generated by first randomly generated the support of the errors χ_E with each entry being 1 with probability p_{err} and 0 with probability $1 - p_{err}$. Then for each non-zero entry in the support, the value of that entry of E is generated accordint to the normal distribtion $\mathcal{N}(0, \sigma_e)$. No erasures are generated for this experiment, i.e. $\chi_\Omega = \mathbb{1}_{m\times n}$.

Success of the experiments is measured by calculating the relative recovery error for each of the variables A, E, and c, where the relative recovery error of a variable x is defined by

$$\text{error}(x) = \frac{\|x - \widehat{x}\|_{\text{vec}(2)}}{\|x\|_{\text{vec}(2)}}$$

where x is the true value of the variable generated by the model described above, and $\widehat{x}$ is the recovered value using the ALRMC algorithm.

The results were averaged over 10 trials, with parameters shown in Table 2.1. Results of the tests are shown in Figure 2.7.

Table 2.1: Parameter Values for Experiments 1 and 2

τ	1×10^{-6}
`max_iter`	10000
λ_c	0.04
λ_e	1
γ_2	0.05
γ_3	0.01
κ_1	1.01
κ_2	3×10^5
κ_3	1.01
σ_c	500
σ_e	1000
σ_z	0

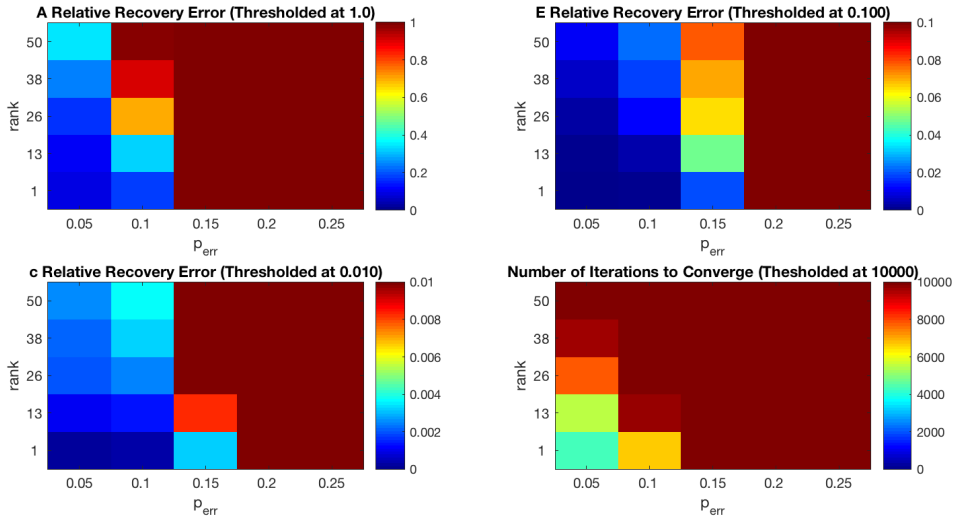

Figure 2.7: Experiment 1

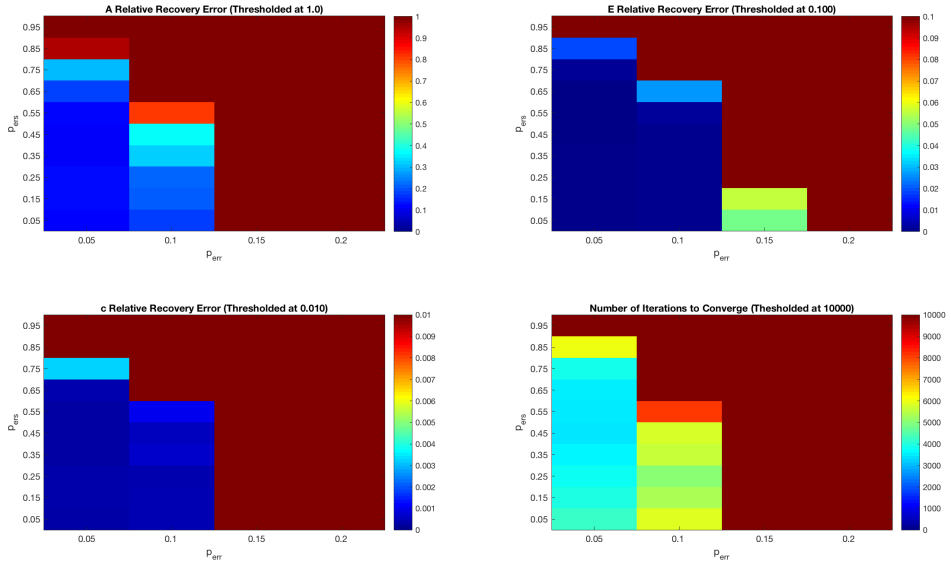

Figure 2.8: Experiment 2

2.10.2 Experiment 2

This experiment tests the performance of the algorithm over different probabilities of erasures and different probabilities errors in the data. The parameter p_{ers} denotes the probability of erasures in the data, and just like above, p_{err} denotes the probability of errors in the data. A and E are generated identically to experiment 1, except that erasures are now introduces according to the uniform distribution with probability p_{ers} and $r = \text{rank}(A) = 1$.

The results were averaged over 10 trials, with parameters shown in Table 2.1. Results of the tests are shown in Figure 2.8.

2.11 Appendix A: Singular Values of Hessian Matrices

We will need the following lemmas for some of the following computations:

Lemma 2.11.1. *Suppose M is a symmetric, positive semi-definite matrix. Then every eigenvalue of M is also a singular value of M.*

Proof. Suppose that λ is an eigenvalue of M with associated eigenvector v. Then

$$\begin{aligned} Mv &= \lambda v \\ M^{\mathrm{T}}Mv &= \lambda M^{\mathrm{T}}v \\ M^{\mathrm{T}}Mv &= \lambda Mv \\ M^{\mathrm{T}}Mv &= \lambda^2 v, \end{aligned}$$

so λ^2 is an eigenvalue of $M^{\mathrm{T}}M$. Then by definition, $\sqrt{\lambda^2} = |\lambda|$ is a singular value of M. However, because M is positive semi-definite, $\lambda \geq 0$. Thus $|\lambda| = \lambda$ is a singular value of M. □

Lemma 2.11.2. *Given a 2×2 block matrix*

$$M = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right],$$

and supposing that the matrix D is invertible, the determinant of M can be calculated by

$$\det(M) = \det(A - BD^{-1}C)\det(D)$$

Proof. This follows easily using properties of determinants from the factorization

$$\left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right] \left[\begin{array}{c|c} I & 0 \\ \hline -D^{-1}C & I \end{array}\right] = \left[\begin{array}{c|c} A - BD^{-1}C & B \\ \hline 0 & D \end{array}\right]$$

□

Lemma 2.11.3. *Suppose M is a 2×2 block matrix*

$$M = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right],$$

and suppose that $CD = DC$ and D is invertible. Then the determinant of M can be calculated by

$$\det(M) = \det(AD - BC)$$

Proof. This follows easily using properties of determinants and Lemma 2.11.2

$$\begin{aligned} \det(M) &= \det(A - BD^{-1}C)\det(D) \\ &= \det((A - BD^{-1}C)D) \\ &= \det(AD - BD^{-1}CD) \\ &= \det(AD - BD^{-1}DC) \\ &= \det(AD - BC) \end{aligned}$$

□

Lemma 2.11.4. *Denote $X_n = aI_n - b\mathbb{1}_{n\times n}$. Then*

$$X_n^{-1} = \left(\frac{1}{a^n - na^{n-1}b}\right)\Big((a^{n-1} - na^{n-2}b)I_n + a^{n-2}b\mathbb{1}_{n\times n}\Big).$$

Moreover, $\det(X_n) = a^n - na^{n-1}b$.

Proof. Check:

$$\begin{aligned}
X_n^{-1}X_n &= \left(\frac{1}{a^n - na^{n-1}b}\right)\Big((a^{n-1} - na^{n-2}b)I_n + a^{n-2}b\mathbb{1}_{n\times n}\Big)\,(aI_n - b\mathbb{1}_{n\times n}) \\
&= \left(\frac{1}{a^n - na^{n-1}b}\right)\Big(a(a^{n-1} - na^{n-2}b)I_n + b(a^{n-1} - na^{n-2}b)\mathbb{1}_{n\times n} \\
&\qquad + a^{n-1}b\mathbb{1}_{n\times n} + a^{n-2}b^2\mathbb{1}_{n\times n}\mathbb{1}_{n\times n}\Big) \\
&= \left(\frac{1}{a^n - na^{n-1}b}\right)\Big((a^n - na^{n-1}b)I_n + (a^{n-1}b - na^{n-2}b^2)\mathbb{1}_{n\times n} \\
&\qquad + a^{n-1}b\mathbb{1}_{n\times n} + na^{n-2}b^2\mathbb{1}_{n\times n}\Big) \\
&= \left(\frac{1}{a^n - na^{n-1}b}\right)\Big((a^n - na^{n-1}b)I_n + \\
&\qquad (a^{n-1}b - na^{n-2}b^2 + a^{n-1}b + na^{n-2}b^2)\mathbb{1}_{n\times n}\Big) \\
&= I_n.
\end{aligned}$$

We will prove the formula for the determinant by induction. Note that in the case where $n = 1$, we have a trivially true statement. Now supposing that the formula holds for $n \in \mathbb{N}$, write X_{n+1} as a block matrix as follows

$$X_{n+1} = \left[\begin{array}{c|c} (a-b) & -b\mathbb{1}_{1\times n} \\ \hline -b\mathbb{1}_{n\times 1} & X_n \end{array}\right]$$

By Lemma 2.11.2,

$$\begin{aligned}
\det(X_{n+1}) &= \left((a-b) - \left(\frac{b^2}{a^n - na^{n-1}b}\right)\Big(n(a^{n-1} - na^{n-2}b) + n^2a^{n-2}b\Big)\right) \\
&\qquad \times (a^n - na^{n-1}b) \\
&= (a-b)(a^n - na^{n-1}b) - b^2\Big(n(a^{n-1} - na^{n-2}b) + n^2a^{n-2}b\Big) \\
&= a^{n+1} - na^nb - a^nb + na^{n-1}b^2 - na^{n-1}b^2 \\
&= a^{n+1} - (n+1)a^nb.
\end{aligned}$$

□

2.11.1 ALRMC Version A

Recall that the Hessian of f in ALRMCvA is given by:

$$H = \begin{bmatrix} & \lambda_z I_{mn} & & \lambda_z \begin{pmatrix} \mathbb{1}_{n\times 1} & & \\ & \ddots & \\ & & \mathbb{1}_{n\times 1} \end{pmatrix} \\ \lambda_z \begin{pmatrix} \mathbb{1}_{1\times n} & & \\ & \ddots & \\ & & \mathbb{1}_{1\times n} \end{pmatrix} & & (\lambda_c + \lambda_z n) I_m \end{bmatrix}.$$

For compactness of notation, we will denote

$$S = \begin{bmatrix} \mathbb{1}_{n\times 1} & & \\ & \ddots & \\ & & \mathbb{1}_{n\times 1} \end{bmatrix}, \qquad R = \begin{bmatrix} \mathbb{1}_{n\times n} & & \\ & \ddots & \\ & & \mathbb{1}_{n\times n} \end{bmatrix}.$$

To calculate the eigenvalues of H, start by writing in block form the matrix

$$H - \lambda I_{2n(m+1)} = \left[\begin{array}{c|c} (\lambda_z - \lambda) I_{mn} & \lambda_z S \\ \hline \lambda_z S^{\mathrm{T}} & (\lambda_c + \lambda_z n - \lambda) I_m \end{array}\right] = \left[\begin{array}{c|c} A & B \\ \hline B^{\mathrm{T}} & D \end{array}\right]$$

The idea is to now use the block matrix determinant formula from Lemma 2.11.2. To this end, simplify piece by piece

$$\begin{aligned} A - BD^{-1}B^{\mathrm{T}} &= (\lambda_z - \lambda)I_{mn} - \lambda_z S(\lambda_c + \lambda_z n - \lambda)^{-1} I_m \lambda_z S^{\mathrm{T}} \\ &= (\lambda_z - \lambda)I_{mn} - \frac{\lambda_z^2}{\lambda_c + \lambda_z n - \lambda} R \\ &= \begin{bmatrix} (\lambda_z - \lambda)I_n - \frac{\lambda_z^2}{\lambda_c + \lambda_z n - \lambda}\mathbb{1}_{n\times n} & & \\ & \ddots & \\ & & (\lambda_z - \lambda)I_n - \frac{\lambda_z^2}{\lambda_c + \lambda_z n - \lambda}\mathbb{1}_{n\times n} \end{bmatrix} \end{aligned}$$

from whence we have

$$\begin{aligned} \det(A - BD^{-1}B^{\mathrm{T}}) &= \left(\det\left((\lambda_z - \lambda)I_n - \frac{\lambda_z^2}{\lambda_c + \lambda_z n - \lambda}\mathbb{1}_{n\times n}\right)\right)^m \\ &= \left((\lambda_z - \lambda)^n - n(\lambda_z - \lambda)^{n-1}\frac{\lambda_z^2}{\lambda_c + n\lambda_z - \lambda}\right)^m \\ &= \left((\lambda_z - \lambda)^{n-1}\left((\lambda_z - \lambda) - \frac{n\lambda_z^2}{\lambda_c + n\lambda_z - \lambda}\right)\right)^m \end{aligned}$$

and it can easily be computed that

$$\det(D) = \det((\lambda_c + n\lambda_z - \lambda)I_m) = (\lambda_c + n\lambda_z - \lambda)^m$$

Therefore,

$$\begin{aligned} \det(H - \lambda I) &= \left((\lambda_z - \lambda)^{n-1}\left((\lambda_z - \lambda) - \frac{n\lambda_z^2}{\lambda_c + n\lambda_z - \lambda}\right)\right)^m (\lambda_c + n\lambda_z - \lambda)^m \\ &= \left((\lambda_z - \lambda)^{n-1}\left((\lambda_z - \lambda)(\lambda_c + n\lambda_z - \lambda) - n\lambda_z^2\right)\right)^m \\ &= \left((\lambda_z - \lambda)^{n-1}\left(\lambda^2 - (\lambda_c + (n+1)\lambda_z)\lambda + \lambda_c\lambda_z\right)\right)^m \end{aligned}$$

From this, it follows that the eigenvalues of H are

$$\begin{array}{ll} \lambda_z & \text{with multiplicity } m(n-1) \\ \frac{\lambda_c+(n+1)\lambda_z \pm \sqrt{(\lambda_c+(n+1)\lambda_z)^2-4\lambda_c\lambda_z}}{2} & \text{each with multiplicity } m. \end{array}$$

Lemma 2.11.5. *In the formulation of the problem for ALRMC Version A, H is positive definite and*

$$\|H\|_{2\to 2} = \frac{\lambda_c + (n+1)\lambda_z + \sqrt{\Big(\lambda_c + (n+1)\lambda_z\Big)^2 - 4\lambda_c\lambda_z}}{2}.$$

Proof. From the above, we know the three unique eigenvalues of H are λ_z and

$$\frac{\lambda_c + (n+1)\lambda_z \pm \sqrt{\Big(\lambda_c + (n+1)\lambda_z\Big)^2 - 4\lambda_c\lambda_z}}{2}.$$

Denote

$$\begin{aligned} e^+ &= \frac{\lambda_c + (n+1)\lambda_z + \sqrt{\Big(\lambda_c + (n+1)\lambda_z\Big)^2 - 4\lambda_c\lambda_z}}{2} \\ e^- &= \frac{\lambda_c + (n+1)\lambda_z - \sqrt{\Big(\lambda_c + (n+1)\lambda_z\Big)^2 - 4\lambda_c\lambda_z}}{2}, \end{aligned}$$

and recall that $n \geq 1$ and λ_c, $\lambda_z > 0$. It is clear that $e^+ > 0$ and $\lambda_z > 0$. To prove

the first part of the Lemma, it only remains to prove that $e^- > 0$.

$$\begin{aligned}
-4\lambda_c\lambda_z &< 0 \\
(\lambda_c + (n+1)\lambda_z)^2 - 4\lambda_c\lambda_z &< (\lambda_c + (n+1)\lambda_z)^2 \\
\sqrt{(\lambda_c + (n+1)\lambda_z)^2 - 4\lambda_c\lambda_z} &< \lambda_c + (n+1)\lambda_z \\
0 &< \lambda_c + (n+1)\lambda_z - \sqrt{(\lambda_c + (n+1)\lambda_z)^2 - 4\lambda_c\lambda_z} \\
0 &< \frac{\lambda_c + (n+1)\lambda_z - \sqrt{(\lambda_c + (n+1)\lambda_z)^2 - 4\lambda_c\lambda_z}}{2}
\end{aligned}$$

thus all eigenvalues are positive, and hence H is positive definite.

Note now that $e^+ > e^-$ and $e^+ > \lambda_z$. Because H is also symmetric, by Lemma 2.11.1 we know that the eigenvalues are equal to the singular values, and hence the largest eigenvalue is also the largest singular value. Therefore,

$$\|H\|_{2\to 2} = \frac{\lambda_c + (n+1)\lambda_z + \sqrt{\Big(\lambda_c + (n+1)\lambda_z\Big)^2 - 4\lambda_c\lambda_z}}{2}$$

□

2.11.2 ALRMC Version B

From the above, we have that

$$H = \begin{bmatrix} & \lambda_z I_{mn} & & \lambda_z I_{mn} & & \lambda_z \begin{bmatrix} \mathbb{1}_{n\times 1} & & \\ & \ddots & \\ & & \mathbb{1}_{n\times 1} \end{bmatrix} \\ & \lambda_z I_{mn} & & \lambda_z I_{mn} & & \lambda_z \begin{bmatrix} \mathbb{1}_{n\times 1} & & \\ & \ddots & \\ & & \mathbb{1}_{n\times 1} \end{bmatrix} \\ \lambda_z \begin{bmatrix} \mathbb{1}_{1\times n} & & \\ & \ddots & \\ & & \mathbb{1}_{1\times n} \end{bmatrix} & & \lambda_z \begin{bmatrix} \mathbb{1}_{1\times n} & & \\ & \ddots & \\ & & \mathbb{1}_{1\times n} \end{bmatrix} & & (\lambda_c + n\lambda_z) I_m & \end{bmatrix}$$

For compactness of notation, we will denote

$$S = \begin{bmatrix} \mathbb{1}_{n\times 1} & & \\ & \ddots & \\ & & \mathbb{1}_{n\times 1} \end{bmatrix}, \qquad R = \begin{bmatrix} \mathbb{1}_{n\times n} & & \\ & \ddots & \\ & & \mathbb{1}_{n\times n} \end{bmatrix}.$$

To calculate the eigenvalues of this matrix, start by writing $H - \lambda I_{m(2n+1)}$ in block form as

$$\begin{aligned} H - \lambda I_{m(2n+1)} &= \left[\begin{array}{ccc} (\lambda_z - \lambda) I_{mn} & \lambda_z I_{mn} & \lambda_z S \\ \lambda_z I_{mn} & (\lambda_z - \lambda) I_{mn} & \lambda_z S \\ \hline \lambda_z S^{\mathrm{T}} & \lambda_z S^{\mathrm{T}} & (\lambda_c + n\lambda_z - \lambda) I_m \end{array}\right] \\ &= \left[\begin{array}{c|c} A & B \\ \hline B^{\mathrm{T}} & D \end{array}\right] \end{aligned}$$

By the formula in Lemma 2.11.2 we now have

$$\det(H - \lambda I_{m(2n+1)}) = \det(A - BD^{-1}B^{\mathrm{T}}) \det(D)$$

Simplify this by analyzing

$$\begin{aligned} A - BD^{-1}B^{\mathrm{T}} &= \begin{bmatrix} (\lambda_z - \lambda)I_{mn} & \lambda_z I_{mn} \\ \lambda_z I_{mn} & (\lambda_z - \lambda)I_{mn} \end{bmatrix} \\ &\qquad - \begin{bmatrix} \lambda_z S \\ \lambda_z S \end{bmatrix} \frac{1}{\lambda_c + n\lambda_z - \lambda} I_m \begin{bmatrix} \lambda_z S^{\mathrm{T}} & \lambda_z S^{\mathrm{T}} \end{bmatrix} \\ &= \begin{bmatrix} (\lambda_z - \lambda)I_{mn} & \lambda_z I_{mn} \\ \lambda_z I_{mn} & (\lambda_z - \lambda)I_{mn} \end{bmatrix} - \frac{\lambda_z^2}{\lambda_c + n\lambda_z - \lambda} \left[\begin{array}{c|c} R & R \\ \hline R & R \end{array}\right] \\ &= \left[\begin{array}{ccc|ccc} \Lambda & & & \Phi & & \\ & \ddots & & & \ddots & \\ & & \Lambda & & & \Phi \\ \hline \Phi & & & \Lambda & & \\ & \ddots & & & \ddots & \\ & & \Phi & & & \Lambda \end{array}\right] \end{aligned}$$

where

$$r = \frac{-\lambda_z^2}{\lambda_c + n\lambda_z - \lambda}, \quad \Lambda = (\lambda_z - \lambda)I_n + r\mathbb{1}_{n\times n}, \quad \Phi = \lambda_z I_n + r\mathbb{1}_{n\times n}.$$

Note now that I_n and $\mathbb{1}_{n\times n}$ commute. This means that $\Lambda\Phi = \Phi\Lambda$, and thus

$$\begin{bmatrix} \Lambda & & \\ & \ddots & \\ & & \Lambda \end{bmatrix} \begin{bmatrix} \Phi & & \\ & \ddots & \\ & & \Phi \end{bmatrix} = \begin{bmatrix} \Phi & & \\ & \ddots & \\ & & \Phi \end{bmatrix} \begin{bmatrix} \Lambda & & \\ & \ddots & \\ & & \Lambda \end{bmatrix}$$

as well. So by Lemma 2.11.3 we have

$$\begin{aligned}
\det\left(A - BD^{-1}C\right) &= \det\left(\begin{bmatrix} \Lambda^2 - \Phi^2 & & \\ & \ddots & \\ & & \Lambda^2 - \Phi^2 \end{bmatrix}\right) \\
&= \left(\det(\Lambda^2 - \Phi^2)\right)^m \\
&= \left(\det\left([(\lambda_z - \lambda)I_n + r\mathbb{1}_{n\times n}]^2 - [\lambda_z I_n + r\mathbb{1}_{n\times n}]^2\right)\right)^m \\
&= \Big(\det\Big((\lambda_z - \lambda)^2 I_n + (2(\lambda_z - \lambda)r + nr^2)\mathbb{1}_{n\times n} \\
&\qquad - [\lambda_z^2 I_n + (2\lambda_z r + nr^2)\mathbb{1}_{n\times n}]\Big)\Big)^m \\
&= \left(\det\left((\lambda^2 - 2\lambda_z\lambda)\right) I_n - 2\lambda r\mathbb{1}_{n\times n}\right)^m \\
&= \left((\lambda^2 - 2\lambda_z\lambda)^n - n(\lambda^2 - 2\lambda_z\lambda)^{n-1}2\lambda r\right)^m, \quad \text{by Lemma 2.11.4} \\
&= \left((\lambda^2 - 2\lambda_z\lambda)^n + (\lambda^2 - 2\lambda_z\lambda)^{n-1}\frac{2n\lambda\lambda_z^2}{\lambda_c + n\lambda_z - \lambda}\right)^m.
\end{aligned}$$

Therefore, we have

$$\begin{aligned}
\det(H - \lambda I_{m(2n+1)}) &= \det(A - BD^{-1}C)\det(D) \\
&= \left((\lambda^2 - 2\lambda_z\lambda)^n + (\lambda^2 - 2\lambda_z\lambda)^{n-1}\frac{2n\lambda_z^2\lambda}{\lambda_c + n\lambda_z - \lambda}\right)^m \\
&\qquad \times (\lambda_c + n\lambda_z - \lambda)^m \\
&= \left((\lambda^2 - 2\lambda_z\lambda)^n(\lambda_c + n\lambda_z - \lambda) + (\lambda^2 - 2\lambda_z\lambda)^{n-1}(2n\lambda_z^2\lambda)\right)^m \\
&= \left((\lambda^2 - 2\lambda_z\lambda)^{n-1}\left((\lambda^2 - 2\lambda_z\lambda)(\lambda_c + n\lambda_z - \lambda) + (2n\lambda_z^2\lambda)\right)\right)^m \\
&= \lambda^{m(n-1)}(\lambda - 2\lambda_z)^{m(n-1)}\left(-\lambda^3 + (\lambda_c + n\lambda_z + 2\lambda_z)\lambda^2 - 2\lambda_c\lambda_z\lambda\right)^m \\
&= \lambda^{mn}(\lambda - 2\lambda_z)^{m(n-1)}\left(-\lambda^2 + (\lambda_c + (n+2)\lambda_z)\lambda - 2\lambda_c\lambda_z\right)^m,
\end{aligned}$$

and the eigenvalues of H are thus

$$\begin{array}{cc} 0 & \text{with multiplicity } mn \\ 2\lambda_z & \text{with multiplicity } m(n-1) \\ \frac{\lambda_c+(n+2)\lambda_z \pm \sqrt{(\lambda_c+(n+2)\lambda_z)^2-8\lambda_c\lambda_z}}{2} & \text{each with multiplicity } m. \end{array}$$

Lemma 2.11.6. *In the formulation of the problem for ALRMC Version B, H is positive semi-definite and*

$$\|H\|_{2\to 2} = \frac{\lambda_c + (n+2)\lambda_z + \sqrt{(\lambda_c + (n+2)\lambda_z)^2 - 8\lambda_c\lambda_z}}{2}.$$

Proof. Denote

$$\begin{aligned} e^+ &= \frac{\lambda_c + (n+2)\lambda_z + \sqrt{(\lambda_c + (n+2)\lambda_z)^2 - 8\lambda_c\lambda_z}}{2} \\ e^- &= \frac{\lambda_c + (n+2)\lambda_z - \sqrt{(\lambda_c + (n+2)\lambda_z)^2 - 8\lambda_c\lambda_z}}{2} \end{aligned}$$

It is clear that $e^+ > e^-$ and $e^+ > 2\lambda_z > 0$. Now we need to show that $e^- > 0$:

$$\begin{aligned} 0 &> -8\lambda_z\lambda_c \\ (\lambda_c + (n+2)\lambda_z)^2 &> (\lambda_c + (n+2)\lambda_z)^2 - 8\lambda_z\lambda_c \\ \lambda_c + (n+2)\lambda_z - \sqrt{(\lambda_c + (n+2)\lambda_z)^2 - 8\lambda_z\lambda_c} &> 0. \end{aligned}$$

Therefore, H is positive semi-definite. Because H is also symmetric, by Lemma 2.11.1, we know the eigenvalues of H are the same as the singular values. Since we have already prove that e^+ is the largest eigenvalue, it is now the largest singular value, and we are done. □

2.11.3 ALRMC Version C

$$H = \begin{bmatrix} \lambda_z \mathrm{Diag}(\omega) & \lambda_z \mathrm{Diag}(\omega) & \lambda_z S_C^{\mathrm{T}} \\ \lambda_z \mathrm{Diag}(\omega) & \lambda_z \mathrm{Diag}(\omega) & \lambda_z S_C^{\mathrm{T}} \\ \lambda_z S_C & \lambda_z S_C & \lambda_c I_m + \lambda_z \mathrm{Diag}(\chi_\Omega \mathbb{1}_{n\times 1}) \end{bmatrix}$$

where

$$S_C = \begin{bmatrix} \chi_\Omega[1,:] & & \\ & \ddots & \\ & & \chi_\Omega[m,:] \end{bmatrix} = \begin{bmatrix} x_1 & & \\ & \ddots & \\ & & x_m \end{bmatrix}$$

To calculate the eigenvalues of this matrix, let $M = H - \lambda I_{m(2n+1)}$ and write M in block form as

$$M = \left[\begin{array}{cc|c} \lambda_z \mathrm{Diag}(\omega) - \lambda I_{mn} & \lambda_z \mathrm{Diag}(\omega) & \lambda_z S_C^{\mathrm{T}} \\ \lambda_z \mathrm{Diag}(\omega) & \lambda_z \mathrm{Diag}(\omega) - \lambda I_{mn} & \lambda_z S_C^{\mathrm{T}} \\ \hline \lambda_z S_C & \lambda_z S_C & (\lambda_c - \lambda) I_m + \lambda_z \mathrm{Diag}(\chi_\Omega \mathbb{1}_{n\times 1}) \end{array}\right]$$
$$= \left[\begin{array}{c|c} A & B \\ \hline B^{\mathrm{T}} & D \end{array}\right]$$

By the formula in Lemma 2.11.2 we now have

$$\det(H - \lambda I_{m(2n+1)}) = \det(A - BD^{-1}B^{\mathrm{T}}) \det(D)$$

For notational simplicity, let $x_i = \chi_\Omega[i,:]$, and let $q_i = x_i \mathbb{1}_{n\times 1}$, i.e. q_i is the number of non-erased entries on row i. First let us calculate $\det(D)$:

$$\det(D) = \det\left((\lambda_c - \lambda) I_m + \lambda_z \mathrm{Diag}(\chi_\Omega \mathbb{1}_{n\times 1})\right)$$
$$\det(D) = \det\left((\lambda_c - \lambda) I_m + \lambda_z \mathrm{Diag}(q)\right)$$
$$\det(D) = \prod_{i=1}^{m} (\lambda_c - \lambda + \lambda_z q_i)$$

where $q = [q_i] \in \mathbb{N}^m$ is the vector of the q_i. Now we calculate $\det(A - BD^{-1}B^{\mathrm{T}})$:

$$\begin{aligned} A - BD^{-1}B^{\mathrm{T}} &= A - \left[\begin{array}{c} S_C^{\mathrm{T}} \\ \hline S_C^{\mathrm{T}} \end{array}\right] \begin{bmatrix} \frac{-\lambda_z^2}{\lambda_c - q_1\lambda_z - \lambda} & & \\ & \ddots & \\ & & \frac{-\lambda_z^2}{\lambda_c - q_m\lambda_z - \lambda} \end{bmatrix} \left[\begin{array}{c|c} S_C & S_C \end{array}\right] \\ &= \begin{bmatrix} \lambda_z\mathrm{Diag}(\omega) - \lambda I_{mn} & \lambda_z\mathrm{Diag}(\omega) \\ \lambda_z\mathrm{Diag}(\omega) & \lambda_z\mathrm{Diag}(\omega) - \lambda I_{mn} \end{bmatrix} + \left[\begin{array}{c|c} R_C & R_C \\ \hline R_C & R_C \end{array}\right] \\ &= \left[\begin{array}{ccc|ccc} \Lambda_1 & & & \Phi_1 & & \\ & \ddots & & & \ddots & \\ & & \Lambda_m & & & \Phi_m \\ \hline \Phi_1 & & & \Lambda_1 & & \\ & \ddots & & & \ddots & \\ & & \Phi_m & & & \Lambda_m \end{array}\right] \end{aligned}$$

where we have used the simplifying notation

$$\begin{aligned} r_i &= \frac{-\lambda_z^2}{\lambda_c + q_i\lambda_z - \lambda} \\ \Lambda_i &= \lambda_z\mathrm{Diag}(x_i^{\mathrm{T}}) - \lambda I_n + r_i x_i^{\mathrm{T}} x_i \\ \Phi_i &= \lambda_z\mathrm{Diag}(x_i^{\mathrm{T}}) + r_i x_i^{\mathrm{T}} x_i \\ R_C &= \begin{bmatrix} r_1 x_1^{\mathrm{T}} x_1 & & \\ & \ddots & \\ & & r_m x_m^{\mathrm{T}} x_m \end{bmatrix}. \end{aligned}$$

Similarly to above, the matrices Λ_i and Φ_i commute, i.e.

$$\Lambda_i\Phi_i = \Phi_i\Lambda_i$$

and so we have

$$\begin{bmatrix} \Lambda_1 & & \\ & \ddots & \\ & & \Lambda_m \end{bmatrix} \begin{bmatrix} \Phi_1 & & \\ & \ddots & \\ & & \Phi_m \end{bmatrix} = \begin{bmatrix} \Phi_1 & & \\ & \ddots & \\ & & \Phi_m \end{bmatrix} \begin{bmatrix} \Lambda_1 & & \\ & \ddots & \\ & & \Lambda_m \end{bmatrix}$$

as well. So by Lemma 2.11.3 we have

$$\begin{aligned}
\det\left(A - BD^{-1}B^{\mathrm{T}}\right) &= \det\left(\begin{bmatrix} \Lambda_1^2 - \Phi_1^2 & & \\ & \ddots & \\ & & \Lambda_m^2 - \Phi_m^2 \end{bmatrix}\right) \\
&= \prod_{i=1}^{m} \det(\Lambda_i^2 - \Phi_i^2) \\
&= \prod_{i=1}^{m} \det([\lambda_z \mathrm{Diag}(x_i^{\mathrm{T}}) - \lambda I_n + r_i x_i^{\mathrm{T}} x_i]^2 \\
&\qquad - [\lambda_z \mathrm{Diag}(x_i^{\mathrm{T}}) + r_i x_i^{\mathrm{T}} x_i]^2) \\
&= \prod_{i=1}^{m} \det(\lambda^2 I_n - 2\lambda\lambda_z \mathrm{Diag}(x_i^{\mathrm{T}}) - 2\lambda r_i x_i^{\mathrm{T}} x_i) \\
&= \prod_{i=1}^{m} \det\left(\begin{bmatrix} \lambda^2 I_{n-q_i} & \\ & (\lambda^2 - 2\lambda\lambda_z) I_{q_i} - 2\lambda r_i \mathbb{1}_{q_i \times q_i} \end{bmatrix}\right), \quad \text{(note 1)} \\
&= \prod_{i=1}^{m} \lambda^{2(n-q_i)} \left((\lambda^2 - 2\lambda\lambda_z)^{q_i} - q_i(\lambda^2 - 2\lambda\lambda_z)^{q_i - 1} 2\lambda r_i\right), \quad \text{(note 2)} \\
&= \prod_{i=1}^{m} \lambda^{2(n-q_i)} (\lambda^2 - 2\lambda\lambda_z)^{q_i - 1} \left(\lambda^2 - 2\lambda\lambda_z - 2q_i\lambda r_i\right) \\
&= \prod_{i=1}^{m} \lambda^{2n-q_i} (\lambda - 2\lambda_z)^{q_i - 1} \left(\lambda - 2\lambda_z - 2q_i r_i\right),
\end{aligned}$$

where (note 1) following by switching rows and (note 2) follows by Lemma 2.11.4.

Therefore,

$$\begin{aligned}
\det(H-\lambda I_{m(2n+1)}) &= \det(A-BD^{-1}C)\det(D)\\
&= \left(\prod_{i=1}^{m}\lambda^{2n-q_i}(\lambda-2\lambda_z)^{q_i-1}\left(\lambda-2\lambda_z-2q_ir_i\right)\right)\\
&\qquad\times\left(\prod_{i=1}^{m}\left(\lambda_c-\lambda+\lambda_z q_i\right)\right)\\
&= \prod_{i=1}^{m}\lambda^{2n-q_i}(\lambda-2\lambda_z)^{q_i-1}\left((\lambda-2\lambda_z)-2q_ir_i(\lambda_c+\lambda_z q_i-\lambda)\right)\\
&= \prod_{i=1}^{m}\lambda^{2n-q_i}(\lambda-2\lambda_z)^{q_i-1}\Bigg((\lambda-2\lambda_z)(\lambda_c+\lambda_z q_i-\lambda)\\
&\qquad-2q_i\left(\frac{-\lambda_z^2}{\lambda_c+q_i\lambda_z-\lambda}\right)(\lambda_c+\lambda_z q_i-\lambda)\Bigg)\\
&= \prod_{i=1}^{m}\lambda^{2n-q_i}(\lambda-2\lambda_z)^{q_i-1}\Big((\lambda\lambda_c+\lambda\lambda_z q_i-\lambda^2)\\
&\qquad+(-2\lambda_z\lambda_c-2\lambda_z^2q_i+2\lambda_z\lambda)+2q_i\lambda_z^2\Big)\\
&= \prod_{i=1}^{m}\lambda^{2n-q_i}(\lambda-2\lambda_z)^{q_i-1}\left(\lambda\lambda_c+\lambda\lambda_z q_i-\lambda^2-2\lambda_z\lambda_c+2\lambda_z\lambda)\right)\\
&= \prod_{i=1}^{m}\lambda^{2n-q_i}(\lambda-2\lambda_z)^{q_i-1}\left(-\lambda^2+(\lambda_c+(2+q_i)\lambda_z)\lambda-2\lambda_z\lambda_c)\right)
\end{aligned}$$

so the eigenvalues of H are

0	with multiplicity $\sum_{i=1}^{m}(2n-q_i)=2nm-\sum_{i=1}^{m}q_i$
$2\lambda_z$	with multiplicity $\sum_{i=1}^{m}(q_i-1)=-m+\sum_{i=1}^{m}q_i$
$\frac{\lambda_c+(2+q_i)\lambda_z\pm\sqrt{(\lambda_c+(2+q_i)\lambda_z)^2-8\lambda_c\lambda_z}}{2}$	for $i=1,2,\ldots m$.

Lemma 2.11.7. *Let $q_{max}=\max_i(q_i)$. In the formulation of the problem for ALRMC Version C, H is positive semi-definite and*

$$\|H\|_{2\to2}=\frac{\lambda_c+(2+q_{max})\lambda_z+\sqrt{(\lambda_c+(2+q_{max})\lambda_z)^2-8\lambda_c\lambda_z}}{2}.$$

Proof. Denote

$$e_i^+ = \frac{\lambda_c + (2+q_i)\lambda_z + \sqrt{(\lambda_c + (2+q_i)\lambda_z)^2 - 8\lambda_c\lambda_z}}{2}$$
$$e_i^- = \frac{\lambda_c + (2+q_i)\lambda_z - \sqrt{(\lambda_c + (2+q_i)\lambda_z)^2 - 8\lambda_c\lambda_z}}{2}$$

It is clear that $e_i^+ > e_i^-$ and $e_i^+ > 2\lambda_z > 0$ for all i. Now we need to show that $e_i^- > 0$ for all i:

$$0 > -8\lambda_z\lambda_c$$
$$(\lambda_c + (2+q_i)\lambda_z)^2 > (\lambda_c + (2+q_i)\lambda_z)^2 - 8\lambda_z\lambda_c$$
$$\lambda_c + (2+q_i)\lambda_z - \sqrt{(\lambda_c + (2+q_i)\lambda_z)^2 - 8\lambda_z\lambda_c} > 0.$$

So H is positive semi-definite. Because H is also symmetric, by Lemma 2.11.1 we know that the eigenvalues of H are the same as the singular values. Now note that e_i^+ is an increasing function of q_i, so we are done. □

2.12 Appendix B: Analysis of Gradient Descent

Lemma 2.12.1. *(See Theorem 2.9 in [6]) Suppose that $f\colon \mathcal{H} \to \mathbb{R}$ is convex and has an L-Lipschitz gradient. Then ∇f is $(1/L)$-inverse strongly monotone.*

Proof. Let $x, y \in \mathcal{H}$ and $s(t) = (1-t)x + ty$, then by the Fundamental Theorem of

Calculus for Line Integrals

$$\begin{aligned}
f(y) &= f(x) + \int_0^1 \langle \nabla f(s(t)), y - x \rangle dt \\
&= f(x) + \langle \nabla f(x), y - x \rangle - \langle \nabla f(x), y - x \rangle + \int_0^1 \langle \nabla f(s(t)), y - x \rangle dt \\
&= f(x) + \langle \nabla f(x), y - x \rangle + \int_0^1 -\langle \nabla f(x), y - x \rangle dt + \int_0^1 \langle \nabla f(s(t)), y - x \rangle dt \\
&= f(x) + \langle \nabla f(x), y - x \rangle + \int_0^1 \langle \nabla f(s(t)) - \nabla f(x), y - x \rangle dt.
\end{aligned}$$

Now by Cauchy-Schwartz, we have

$$\begin{aligned}
\langle \nabla f(s(t)) - \nabla f(x), y - x \rangle &\leq \|\nabla f(s(t)) - \nabla f(x)\| \|y - x\| \\
&\leq L\|s(t) - x\| \|y - x\|, \text{ because } \nabla f \text{ is } L\text{-Lipschihtz} \\
&\leq L\|(1-t)x + ty - x\| \|y - x\| \\
&\leq Lt\|y - x\| \|y - x\| \\
&\leq Lt\|y - x\|^2.
\end{aligned}$$

So the above expression can now be bounded by

$$\begin{aligned}
f(y) &\leq f(x) + \langle \nabla f(x), y - x \rangle + \int_0^1 Lt\|y - x\|^2 dt \\
&= f(x) + \langle \nabla f(x), y - x \rangle + \frac{L}{2}\|y - x\|^2.
\end{aligned}$$

Now let us choose $y = x - \frac{1}{L}\nabla f(x)$ to substitute into the above to get

$$\begin{aligned}
f\left(x - \frac{1}{L}\nabla f(x)\right) &\leq f(x) + \left\langle \nabla f(x), -\frac{1}{L}\nabla f(x) \right\rangle + \frac{L}{2}\|\frac{1}{L}\nabla f(x)\|^2 \\
&\leq f(x) + \left(-\frac{1}{L} + \frac{1}{2L}\right)\|\nabla f(x)\|^2 \\
&\leq f(x) - \frac{1}{2L}\|\nabla f(x)\|^2.
\end{aligned}$$

Using the fact that $f\left(x - \frac{1}{L}\nabla f(x)\right) \geq \inf_x(f(x))$ we have

$$\inf_x(f(x)) \leq f(x) - \frac{1}{2L}\|\nabla f(x)\|^2$$
$$\inf_x(f(x)) + \frac{1}{2L}\|\nabla f(x)\|^2 \leq f(x). \qquad \text{(grad 1)}$$

Now we introduce an auxiliary function $\tilde{f}_y$ by subtracting out the tangent plane at y:

$$\tilde{f}_y(x) = f(x) - f(y) - \langle \nabla f(y), x - y \rangle$$

Note that because f is convex and the tangent plane is as well (because it is linear), $\tilde{f}$ is convex as well. Moreover, $\nabla \tilde{f}_y(x) = \nabla f(x) - \nabla f(y)$. Because f has an L-Lipschitz gradient,

$$\begin{aligned}\|\nabla \tilde{f}_y(x_1) - \nabla \tilde{f}_y(x_2)\| &= \|\nabla f(x_1) - \nabla f(y) - (\nabla f(x_2) - \nabla f(y))\| \\ &= \|\nabla f(x_1) - \nabla f(x_2)\| \\ &\leq L\|x_1 - x_2\|.\end{aligned}$$

so $\tilde{f}_y(x)$ has an L-Lipschitz gradient as well. Because $\tilde{f}_y(x)$ satisfies all of the properties we assumed for f, we can substitute $\tilde{f}_y(x)$ for $f(x)$ in (grad 1)

$$\begin{aligned}\inf_x(\tilde{f}_y(x)) + \frac{1}{2L}\|\nabla \tilde{f}_y(x)\|^2 &\leq \tilde{f}_y(x) \\ \tilde{f}_y(y) + \frac{1}{2L}\|\nabla \tilde{f}_y(x)\|^2 &\leq \tilde{f}_y(x) \\ \frac{1}{2L}\|\nabla \tilde{f}_y(x)\|^2 &\leq \tilde{f}_y(x) \\ \frac{1}{2L}\|\nabla f(x) - \nabla f(y)\|^2 &\leq f(x) - f(y) - \langle \nabla f(y), x - y \rangle \\ f(x) &\geq f(y) + \langle \nabla f(y), x - y \rangle + \frac{1}{2L}\|\nabla f(x) - \nabla f(y)\|^2.\end{aligned}$$

Having established this inequality, add it to itself with the roles of x and y reversed

to get

$$0 \geq \langle \nabla f(y), x - y \rangle + \langle \nabla f(y), x - y \rangle + \frac{1}{L} \|\nabla f(x) - \nabla f(y)\|^2$$
$$\langle \nabla f(x) - \nabla f(y), x - y \rangle \geq \frac{1}{L} \|\nabla f(x) - \nabla f(y)\|^2$$

which proves that $\nabla f(x)$ is $\frac{1}{L}$-inverse strongly monotone. □

Theorem 2.12.1. *Suppose that f is a convex function with an L-Lipschitz gradient, and suppose that $\gamma \in (0, \frac{2}{L})$. Then the gradient descent update operator $\mathrm{Id} - \gamma \nabla f$ is $(\frac{2}{\gamma L} - 1)$-pseudocontractive.*

Proof. From Lemma 2.12.1 we know that ∇f is $\frac{1}{L}$-inverse strongly monotone. Note from the definition that if T is α-inverse strongly monotone, then βT is $\frac{\alpha}{\beta}$-inverse strongly monotone. Therefore $\gamma \nabla f$ is $\frac{1}{\gamma L}$-inverse strongly monotone. From Theorem 2.3.1, we now have that the complement $\mathrm{Id} - \gamma \nabla f$ is $(\gamma L/2)$-averaged, or equivalently

$$\frac{1 - \gamma L/2}{\gamma L/2} = \left(\frac{2}{\gamma L} - 1 \right) \text{-pseudocontractive.}$$

Note that the pseudocontractive parameter is required to be strictly positive, hence the requirement that $\gamma \in (0, \frac{2}{L})$. □

Chapter 3

The Robust Orthogonal Rank-One Matrix Pursuit Algorithm

3.1 Introduction

We extend the Orthogonal Rank-One Matrix Pursuit Algorithm (OR1MP) to handle sparse errors. The OR1MP algorithm is an algorithm for solving the matrix completion problem, i.e. the problem of recovering a low rank matrix corrupted by erasures, developed in [24]. It does so by building up a basis of rank one matrices by finding the rank one matrices most correlated with a running residual matrix, then by estimating the optimal coefficients on these basis matrices to approximate the given data matrix. The problem in this dissertation is to recover a low rank matrix that has been corrupted not simply by erasures, but errors as well. The Robust Orthogonal Rank-One Matrix Pursuit (ROR1MP) algorithm, extends the OR1MP algorithm by greedily identifying the errors and then by treating them as missing entries. First, the working of the original algorithm to be extended is explained and the relevant results are collected here. Next we explain the new algorithm and test its performance on both synthetic and real-world data.

3.2 Similar Problems Solved in the Literature

Generally speaking, the problem is to reconstruct a matrix $Y \in \mathbb{R}^{m \times n}$ that has been corrupted by erasures, viz. a matrix where some of the entries are missing. Notationally, we will normally speak of the *support* of the matrix Ω, defined as the set of (i, j) where the entry $y_{i,j}$ is not erased. Of course, in general such reconstruction is impossible, as any missing entry could have held an arbitrary value. The only way in which there is hope for reconstructing a matrix corrupted by erasures is with some extra structure with determines the missing values. This extra structure is given by the assumption that the matrix is a low-rank matrix, i.e. one whose rank $r < \min(m, n)$. Depending on the application, the rank r might be known or unknown.

The matrix completion problem was initially studied in connection to the Netflix problem [9]. Netflix offered prize money for a solution to the problem so it could be applied to its recommendation system. Data was provided on the ratings that Netflix users gave to movies, but naturally most users have not rated all movies available on Netflix, so that data was missing. The hope was to infer the users ratings for movies that they had not seen yet by completing the recommendation matrix with columns corresponding to users. If users of Netflix fall into different clusters that all rate movies similarly, the matrix would be low rank as columns corresponding to users from one cluster would be correlated with one another.

There are a wide variety of techniques for solving the matrix completion problem. A good survey of the literature on matrix completion is found in [16]. However, the problem of recovering a low-rank matrix corrupted by both erasures and sparse errors was first studied in [26]. In this paper, the low-rank matrix completion problem is solved by replacing the rank minimization with a convex surrogate (a common technique for solving matrix completion problems), but in addition there is another term in the convex objective function designed to recover the sparse errors as well. Refer to the previous chapter for an detailed description of this algorithm, on which

the ALRMC algorithm is based.

In this paper, we adapt the OR1MP algorithm, which uses a greedy approach instead of a convex optimization approach, for solving the matrix completion problem with added sparse errors. The OR1MP algorithm is described in the following section.

3.3 The Orthogonal Rank-One Matrix Pursuit (OR1MP) Algorithm

First, some definitions:

Definition 3.3.1. The *incomplete matrix space supported on* Ω *in* $\mathbb{R}^{m\times n}$ is a vector space of incomplete matrices of size $m \times n$ where the matrix is only known on the support set $\Omega \subset \{(i,j) \in [m]\times[n]\}$ where $[n] = \{1, 2, \ldots, n\}$. Note that this space can be identified with the space $\mathbb{R}^{|\Omega|}$ via vectorization (stacking columns) of the known entries. We will denote this space by $\mathbb{R}^{\Omega}$. It will always be clear from context what m and n are, so this notation should not be ambiguous.

Definition 3.3.2. The projection operator onto the incomplete matrix space supported on Ω in $\mathbb{R}^{m\times n}$ is denoted by $\pi_\Omega \colon \mathbb{R}^{m\times n} \to \mathbb{R}^{\Omega}$. This drops all entries of a matrix which do not belong to the support set Ω.

Definition 3.3.3. The zero-impuatation operator fills in any entries of a matrix not supported on the set Ω with the value zero. This operator can be defined on $\mathbb{R}^{m\times n}$ or on the incomplete matrix space supported on Ω in $\mathbb{R}^{m\times n}$. We will not distinguish notationally between the two, and the operator is denoted by $\mathcal{P}_\Omega$.

The OR1MP algorithm completes an incomplete matrix Y supported on a set Ω. The main idea of the OR1MP algorithm is to pursue a basis of the space of m by n matrices consisting of rank-one matrices of unit Frobenius norm, each of which is chosen in a greedy fashion by maximizing the correlation with the residual R^k

Algorithm 3.1: OR1MP

Input: Data matrix $Y \in \mathbb{R}^{m \times n}$ known on support Ω.
Output: Learned matrix $\widehat{Y} = \sum_{i=1}^{k} \theta_i^k (B_i)_\Omega$.

Initialize: Set $X^0 = 0$, $\theta^0 = 0$ and $k = 0$.
while (stopping criterion not met) **do**
  $k = k + 1$
  $R_k = Y_\Omega - X^{k-1}$
  Step 1: Find the k^{th} element of the basis of rank-one matrices B^k by solving

$$B^k = \underset{M \in \mathbb{R}^{m \times n}}{\operatorname{argmax}} \left(\langle M, R^k \rangle : \ \operatorname{rank}(M) = 1, \|M\|_F = 1 \right)$$

  Step 2: Compute the weights θ^k by solving

$$\underset{\theta \in \mathbb{R}^k}{\operatorname{argmin}} \left\| \sum_{i=1}^{k} \theta^i M^i - Y \right\|_\Omega^2$$

  $X^k = \sum_{i=1}^{k} \theta^i B^i$.
end

Figure 3.1: The OR1MP Algorithm

(initialized to be Y). The corresponding weights to this basis are then computed to determine a rank k approximation of the given matrix at iteration k. At each step k of the algorithm, we (1) construct an additional rank-one basis element B_k for the space of matrices, (2) estimate the weights on each of basis elements constructed thus far that best approximate the input matrix Y, and (3) compute the residual R^k using this approximation. Formally, the algorithm can be described as in Figure 3.1.

3.3.1 Implementation Notes

Step 1 Implementation

To implement Step 1 in the algorithm, we start by observing the following lemma:

Lemma 3.3.1. *Any matrix $X \in \mathbb{R}^{n \times m}$ can be written as a linear combination of rank-one matrices, i.e.*

$$X = \sum_{i=1}^{r} \theta_i M_i$$

where $\text{rank}(M_i) = 1$ *for all* i.

Proof. Every matrix X can be factored into a singular value decomposition

$$X = USV^{\mathrm{T}}$$

using the theorem from the Appendix where $S = \text{diag}(s)$ and $\text{rank}(X) = \|s\|_0$. Let m_i denote the i^{th} column of the matrix M. Then we can rewrite X as

$$\begin{aligned}
X &= USV^{\mathrm{T}} \\
&= U \begin{bmatrix} \sigma_1 & & & & & \\ & \ddots & & & & \\ & & \sigma_r & & & \\ & & & 0 & & \\ & & & & \ddots & \\ & & & & & 0 \end{bmatrix} V^{\mathrm{T}} \\
&= U \begin{bmatrix} \text{—}\sigma_1 v_1\text{—} \\ \vdots \\ \text{—}\sigma_r v_r\text{—} \\ \text{—}0\text{—} \\ \vdots \\ \text{—}0\text{—} \end{bmatrix} \\
&= \left[\textstyle\sum_{k=1}^{r} U[i,k]\sigma_k V[k,j] \right]_{i\in[m],j\in[n]} \\
&= \sum_{k=1}^{r} \sigma_k \left[U[i,k]V[k,j] \right]_{i\in[m],j\in[n]} \\
&= \sum_{k=1}^{r} \sigma_k u_k v_k^{\mathrm{T}}
\end{aligned}$$

giving us the desired representation because $\operatorname{rank}(uv^{\mathrm{T}}) = 1$, where $r = \operatorname{rank}(X)$ is the number of nonzero singular values σ_k. □

This result gives us the following corollary:

Corollary 3.3.1. *Any rank-one matrix M with $\|M\|_F = 1$ can be written as*

$$M = uv^T$$

where $\|u\|_2 = 1$ and $\|v\|_2 = 1$.

Proof. We only need note that $\|M\|_F = \|M\|_{\mathrm{Sch}(2)} = \sigma_1 = 1$ and by the representation in the previous lemma we are done. □

Using this corollary, we can recast the computation of the basis element as follows

$$\begin{aligned}
\operatorname{argmax}(\langle M, R\rangle \colon \|M\|_F = 1, \operatorname{rank}(M) = 1) &= \operatorname{argmax}(\langle uv^{\mathrm{T}}, R\rangle \colon \|u\|_2, \|v\|_2 = 1) \\
&= \operatorname{argmax}(\operatorname{tr}((uv^{\mathrm{T}})^{\mathrm{T}} R) \colon \|u\|_2, \|v\|_2 = 1) \\
&= \operatorname{argmax}(\operatorname{tr}(vu^{\mathrm{T}} R) \colon \|u\|_2, \|v\|_2 = 1) \\
&= \operatorname{argmax}(\operatorname{tr}(u^{\mathrm{T}} Rv) \colon \|u\|_2, \|v\|_2 = 1) \\
&= \operatorname{argmax}(u^{\mathrm{T}} Rv \colon \|u\|_2, \|v\|_2 = 1)
\end{aligned}$$

Lemma 3.3.2. *The solution to* $\operatorname{argmax}(u^T R v \colon \|u\|_2, \|v\|_2 = 1)$ *is given by a pair of top left and top right singular vectors of* R*.*

Proof. Let $R = USV^{\mathrm{T}}$ be the singular value decomposition of R. Note that if we let u and v be a pair of top left and top right singular vectors of R then

$$\begin{aligned} u^{\mathrm{T}} R v &= u^{\mathrm{T}} U S V^{\mathrm{T}} v \\ &= [10 \dots 0] S \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \\ &= \sigma_1 \\ &= \|R\|_{2 \to 2} \end{aligned}$$

and the lemma follows from the fact that the spectral norm $\|\cdot\|_{2\to 2}$ is submultiplicative. □

To compute the top left and top right singular vectors of R, we can use the familiar power method for computing the largest eigenvalue.

Step 2 Implementation

To solve for the optimal coefficients on the basis matrices, we can vectorize each basis matrix and solve a system of linear equations for the coefficients.

Stopping Criterion

Two choices of stopping criterion are:

(a) Stop when the residual satisfies $\|R_k\| \leq \epsilon$.

(b) Stop when the rank of X reaches a given value r.

3.4 The Robust Orthogonal Rank-One Matrix Pursuit (ROR1MP) Algorithm

This modification of the OR1MP algorithm works by adding a step in each iteration where we select a subset P^k of possible errors. Elements of P^k are chosen in a greedy fashion, identified by looking for entries where the residual matrix is largest in absolute value. We will consider two different methods for determining the set P^k:

(a) Set the cardinality of P^k, so that a fixed number of possible errors is chosen at each iteration, i.e. pick a parameter $s = |P^k|$ and choose the elements of P^k to be the s elements of the residual that are largest in absolute value.

(b) Set a parameter α so that a fraction α of entries of the matrix are identified as possible errors in each iteration, i.e. so that $\tau = \text{quantile}(\text{vec}(|R^k|), \alpha)$, where $\text{vec}(|R^k|)$ is a vector of all entries of the matrix $|R^k|$ and the function $\text{quantile}(v, \alpha)$ returns the value t such that at least a fraction α of the entries of v are larger than t.

Formally, at each iteration k we set the indices of possible errors $\mathcal{P}_k$ to be

$$\mathcal{P}^k = \{(i,j) \in \Omega \colon |\pi_{i,j}(R_k)| > \tau\}$$

where Ω is once again the set of indices of non-erased entries. Corresponding to this set of indices, we define the matrix $\chi^k_{\mathcal{P}} \in \mathbb{R}^{m \times n}$ where the $(i,j)^{\text{th}}$ entry is 1 if $(i,j) \in \mathcal{P}^k$ and 0 otherwise.

Table 3.1: ROR1MP Experiment 1a: Recovery Errors for Different Portions of Erasures with $p_{err} = 0.01$

p_{ers}	0.01	0.05	0.1	0.5
OR1MP	6.55782	6.4937	6.46902	6.54775
ROR1MP	6.14748	6.11667	6.11403	6.11775

Table 3.2: ROR1MP Experiment 1b: Recovery Errors for Different Portions of Erasures with $p_{err} = 0.05$

p_{ers}	0.01	0.05	0.1	0.5
OR1MP	15.4046	15.4145	15.3738	15.396
ROR1MP	14.6698	14.6282	14.5703	14.6621

3.5 Experimental Results

3.5.1 Experiment 1: Results on Synthetic Data

In this experiment, we generate examples of the form $Y = M + E$ where $M \in \mathbb{R}^{100\times 100}$ is a rank $r < 100$ matrix and test the recovery of the matrix M from this corrupted version.

For the first set of parameters for this experiment, we choose: p_{err}, $r = 5$, $\sigma_{err} = 100$, and the results are listed in table 3.1.

For the second set of parameters for this experiment, we choose: $p_{err} = .05$, $r = 5$, $\sigma_{err} = 100$, and the results are listed in table 3.2.

Table 3.3: ROR1MP Experiment 1c: Recovery Errors for Different Portions of Erasures with $p_{err} = 0.1$

p_{ers}	0.01	0.05	0.1	0.5
OR1MP	22.4682	22.4797	22.4569	22.6421
ROR1MP	21.821	21.6203	21.5965	21.8364

Table 3.4: ROR1MP Experiment 1d: Recovery Errors for Different Portions of Erasures with $p_{err} = 0.5$

p_{ers}	0.01	0.05	0.1	0.5
OR1MP	51.5436	51.5907	51.5343	51.5883
ROR1MP	51.1796	51.2967	51.2811	51.2738

3.5.2 Experiment 2: Results on Images

In this experiment, the performance of the algorithm is tested on an image. This makes sense to test because images when thought of as matrices are not generally of full rank, or at least are very well approximated by low rank matrices. The image used for the test is the standard 512×512 Lena image.

The initial image is then corrupted in two stages: In the first stage, erasures are applied by fixing the fraction of erasures

$$\texttt{p\_ers} = \frac{\text{number of erasures}}{\text{number of pixels in image}}.$$

Then that number of pixels to erase are chosen uniformly randomly from all pixels in the image. In the second stage, errors are applied by fixing the fraction of errors

$$\texttt{p\_err} = \frac{\text{number of errors}}{\text{number of pixels in image}}.$$

Then that number of pixels to corrupt with errors are chosen uniformly randomly from all pixels in the image, and the error applied is to set the pixel value to 1 (white pixel). The result of this two-stage corruption is shown in Figure 3.2.

The results of running the ROR1MP algorithm to correct the image corrupted by erasures and errors is shown in Figure 3.3

(a) Original (b) With erasures (c) With erasures and errors

Figure 3.2: Images Corrupted by Errors and Erasures

(a) OR1MP (b) ROR1MP Algorithm

Figure 3.3: Image Results for OR1MP and ROR1MP Algorithms

3.6 Appendix: Matrix Norms

There are three important classes of matrix norms used in this dissertation:

(a) Operator norms: $\|A\|_{p\to q} = \max_{\|x\|_p=1} (\|Ax\|_q)$

(b) Entrywise norms: $\|A\|_{\text{vec}(p)} = \left(\sum_{i=1}^{m} \sum_{j=1}^{n} A_{i,j}^p \right)^{1/p}$

(c) Schatten norms: $\|A\|_{\text{Sch}(p)} = \left(\sum_{i=1}^{\min(m,n)} (\sigma_i(A))^p \right)^{1/p}$

where $\sigma_i(A)$ is a function that returns the i^{th} singular value of A.

Many of these norms for specific values of p and q have special names:

Definition 3.6.1. The *Frobenius norm* of a matrix A is defined as $\|A\|_{\text{vec}(2)}$, and is often denoted by $\|A\|_F$.

Definition 3.6.2. The *nuclear norm* of a matrix A is defined as $\|A\|_{\text{Sch}(1)}$, and is often denoted by $\|A\|_*$.

Definition 3.6.3. The *spectral norm* of a matrix A is defined as $\|A\|_{2\to 2}$, and is often denoted by $\|A\|_2$, though so are some other matrix norms, so we will avoid this notation.

Lemma 3.6.1. *The spectral norm of a matrix A is equal to the largest singular value of A, i.e. $\|A\|_{2\to 2} = \sigma_1(A) = \|A\|_{Sch(\infty)}$.*

Chapter 4

Learning Low-Dimensional Manifolds by Learning Distance

4.1 Introduction

High-dimensional data in machine learning applications often has an intrinsic low-dimensional structure to it. Classic examples include video and images, which exist in spaces which can have millions of dimensions, and yet most real world images cluster close to low-dimensional manifolds. This property of real world data is often called the *manifold hypothesis*. Because working with high-dimensional data is often quite difficult in terms of algorithmic complexity, there is great interest in methods for dimensionality reduction for working with data satisfying the manifold hypothesis. In the case of linear manifolds, principal component analysis (PCA) is one example of a very successful and well-known algorithm for dimensionality reduction. Unfortunately, real-world data quite often has a low-dimensional but nonlinear structure, and so there is a need for nonlinear methods for dimensionality reduction in order to exploit the structure of the data. *Manifold learning* is a branch of machine learning for working with data which satisfies the manifold hypothesis.

We begin by describing the current state of the field of manifold learning. In this chapter, we introduce an algorithm for constructing an approximate squared distance function by generating a data set of noisy data points from a given set of noiseless data and training a neural network to approximate the squared distance function. Knowing the squared distance function for a manifold provides a wealth of knowledge about the structure of the manifold and is often good enough to accomplish the goals of many applications of manifold learning.

The problem setup is as follows: suppose that M is a k-dimensional manifold embedded in $\mathbb{R}^N$, and that $\{y^i\}_{i=1}^n$ is a given set of (noiseless) data points with $y^i \in M$. We want to learn the structure of M from the given data set $\{y^i\}_{i=1}^n$. This chapter describes an approach for solving this problem where we construct a data set by adding many different noise vectors to each of the given data points, and then train a neural network to learn a function $\Phi\colon \mathbb{R}^N \to \mathbb{R}$ that approximates the squared distance function $\text{dist}^2(\cdot, M)$ to the manifold M. By learning an approximate squared-distance function, we learn a fair amount about the structure of the manifold M in a way that lends itself to useful applications.

There are two main theoretical results, the first of which is as follows: if a k-dimensional manifold M can be accurately approximated by its k-dimensional tangent space T_y at some point $y \in M$ in a ball of radius η, then the distribution of the distance squared between a point uniformly distributed on the sphere $u \sim \text{Unif}(S^{N-1})$ and T_y follows a $\text{Beta}((N-k)/2, k/2)$ distribution. As a corollary, we show that for manifolds of low-dimension embedded in high dimensional space, i.e. $k \ll N$ and $N \gg 0$, then in some sense

$$P(\text{dist}(y + \eta u, M) \approx \eta) \approx 1 \text{ if } u \sim \text{Unif}(S^{N-1}).$$

Leveraging this theoretical result, given a set of points $\{y^i\}_{i=1}^n$ with $y^i \in M$, we can construct a training set consisting of points of the form $\{z_j^i\} = \{y^i + \eta_j^i u_j^i\}$ with $j = 1, 2, \ldots, m$ where the target variable $\text{dist}(y^i + \eta_j^i u_j^i, M) \approx \eta_j^i$ is known approximately with high probability.

The second main theoretical result states that the squared distance function to the manifold M can be written as

$$\text{dist}^2(z, M) = E(\eta^2|z) - k\sigma^2,$$

when $\eta u \sim \mathcal{N}(0, \sigma^2 I_N)$, which tells us the bias term for this approach depends only on the intrinsic dimension k of the manifold and the chosen standard deviation parameter σ of the added noise. We can then train a function Φ from some function class V by minimizing a chosen loss function over the given training data so that $\Phi \approx \text{dist}^2(\cdot, M)$.

Applications of this approximation of the squared distance function to the manifold M include classification and projection onto the manifold M, and hence any applications of those as well including: binary classification, multi-class classification, denoising, sparse error correction, erasure completion, and "style transfer" via projection. Numerical demonstration of these applications is included in the last section.

4.2 Overview of Manifold Learning

In this section, we review the state of the field of manifold learning. A general overview of the field is given by Ma and Fu in [11], who present many of the popular techniques.

Principal components analysis (PCA) is an algorithm for learning the low- dimensional structure primarily of linear manifolds and manifolds that are very close to being linear. See Chapter 2 for details on PCA. It is an important precursor algorithm for many of the manifold learning algorithms used for nonlinear manifolds. For

example, a simple extension of PCA to nonlinear manifolds using kernel methods is the kernel PCA algorithm of Schölkopf *et al.* in [20].

Classical multidimensional scaling (MDS) [14], also known as principal coordinates analysis (PCoA), Torgerson Scaling, or Torgerson–Gower scaling, takes a set of pairwise, Euclidean distances between the given data points and maps the data into a low-dimensional Euclidean space with coordinates that best preserve the given pairwise distances. It is an important precursor algorithm to Isomap, explained below.

Isomap [21], an algorithm of Tenenbaum, Silva, and Langford, is a very popular nonlinear extension of the MDS algorithm. The MDS algorithm is designed for Euclidean distances, which do not work well for nonlinear manifolds which can bend back on themselves, often so that the Euclidean distance between two point $x, y \in M$ can be very small when the geodesic distance between x and y is very large. Isomap preprocesses data from nonlinear manifolds and then feeds that data into the MDS algorithm. Isomap works by implementing a three step algorithm. In the first step, each point in the given data set is given a neighborhood, either by taking balls of radius ϵ or by taking a K-nearest neighbors approach. Second, Isomap estimates the geodesic distance between all pairs of points in the data set by using short hops limited to each of the neighborhoods. Finally, the MDS algorithm is applied to the graph of geodesic distances which constructs an embedding in k-dimensional space that best preserves the intrinsic geometry of the manifold.

The approach in this chapter for manifold learning is similar to that used by Fefferman, Mitter, and Narayanan in [4] and Mohammed and Narayanan in [15] in that we construct an approximate squared distance function. In [4], the approximate squared distance function is constructed by covering the data with a finite collection of cylinders. On each of these cylinders, a function f_i is defined as the squared distance to the k-dimensional central cross section of the cylinder, and then the f_i are combined into an approximate square distance function f to the manifold M using

a partition of unity. Our approach for manifold learning is similar only in the fact that both use an approximate square distance function, but the approximate squared distance function constructed in this chapter is built in an entirely different manner from [4]. However, once the approximate squared distance function is constructed, it can be used in the same way as [4] and [15] to construct an estimate of the manifold. Their approach is to let the estimated manifold M_{put} consist of the points where ∇f is orthogonal to the the span of the largest $(N-k)$ eigenvectors of the Hessian $\nabla^2 f$.

Another well-known method is locally linear embedding (LLE) by Roweis and Saul [19]. This method fits the data locally using a linear model. Each data point is written as a linear combination of the other data points, with the constraints that its neighboring data points should be the only ones with non-zero contribution, and that the total contribution of the weights for each data point should sum to one. This problem is setup as a constrained least squares problem that turns out to have a closed form solution. The method does not require finding shortest paths in large graphs like Isomap does. Another advantage is that LLE only has one free parameter, the number K of nearest neighbors used to compute the neighborhoods of each point in the data set.

4.3 Uniformly Random Points on High-Dimensional Spheres are Approximately Distance One from Low-Dimensional Linear Manifolds

In this section, we make precise and prove the first main theoretical result: that in the model given above, if $k \ll N$, i.e. the dimension of the manifold M is "low" compared to the dimension of the ambient space $\mathbb{R}^N$, then

$$P(\text{dist}(y + \eta u, M) \approx \eta) \approx 1.$$

This provides some intuition about the second main theoretical result, which will be proved later, and which will be directly applicable to manifold learning.

Suppose that M is a k-dimensional smooth manifold embedded in $\mathbb{R}^N$. We are interested in the distribution of $\text{dist}(z, M)$ where z is the random variable given by $z = y + \eta u$. Here, η is a positive-valued random variable with an unspecified distribution, y is a random variable with $y \in M$, and $u \sim \text{Unif}(S^{N-1})$, where $S^{N-1} = \{x \in \mathbb{R}^N \colon \|x\|_2 = 1\}$ is the $(N-1)$-dimensional unit sphere.

To study this distribution, we first approximate the manifold M at a point $y \in M$ using the tangent space T, which is a k-dimensional affine subspace. We will then analyze the distribution of the square of the distance instead of the distance itself. Let $\Delta^2 = \text{dist}^2(z, T)$ denote the random variable of interest.

If we draw points $x_i \overset{\text{iid}}{\sim} \mathcal{N}(0, 1)$, $i = 1, 2, \ldots, N$, where $\mathcal{N}(0, 1)$ is the standard normal distribution with mean 0 and variance 1. Concatenated to form the vector $x = (x_1, x_2, \ldots, x_N)$ and then normalized, we have a vector $u = x/\|x\|_2 \sim \text{Unif}(S^{N-1})$. Now, if $y \in M$ is given, by translation we can assume that $y = 0$, and by rotational symmetry of S^{N-1}, we can assume that the k-dimensional affine subspace T that approximates M at the point y is $\text{Span}(e_1, e_2, \ldots, e_k)$ where e_i is the i^{th} standard basis vector.

The distance from u to T is then given by $\text{dist}(u, T) = \|u - \text{proj}_T(u)\|_2$, but $\text{proj}_T(u)$ is easy to compute in this case as $\text{proj}_T(u) = (0, \ldots, 0, u_{k+1}, \ldots, u_N)$. Now

let us analyze the square of the distance:

$$\begin{aligned}
\|u - \text{proj}_T(u)\|_2^2 &= \sum_{i=k+1}^{N} u_i^2 \\
&= \sum_{i=k+1}^{N} \frac{x_i^2}{\|x\|_2^2} \\
&= \frac{1}{\|x\|_2^2} \sum_{i=k+1}^{N} x_i^2 \\
&= \frac{\sum_{i=k+1}^{N} x_i^2}{\sum_{i=1}^{N} x_i^2} \\
&= \frac{\sum_{i=k+1}^{N} x_i^2}{\sum_{i=1}^{k} x_i^2 + \sum_{i=k+1}^{N} x_i^2} \\
&= \frac{a}{b+a}
\end{aligned}$$

where $a = \sum_{i=k+1}^{N} x_i^2$ and $b = \sum_{i=1}^{k} x_i^2$. The distribution of random variables formed by the sum of ν squared standard normal random variables is well known, and defines the *chi-squared* distribution. So $a \sim \chi^2_{N-k}$ and $b \sim \chi^2_k$, and because a and b do not have any terms in common and the X_i are independent, a and b are also independent. The following theorem from statistics then tells us the distribution of $\text{dist}^2(u, T)$:

Lemma 4.3.1. *Suppose that a and b are independent random variables with $a \sim \chi^2_{N-k}$ and $b \sim \chi^2_k$. Then*

$$\frac{a}{a+b} \sim \text{Beta}\left(\frac{N-k}{2}, \frac{k}{2}\right)$$

Proof. See [1]. □

Theorem 4.3.1. *The square of the distance from u to the k-dimensional affine subspace T approximating the smooth k-dimensional manifold M is distributed*

$$\|u - \text{proj}_T(u)\|_2^2 \sim \text{Beta}\left(\frac{N-k}{2}, \frac{k}{2}\right)$$

Proof. The above analysis and lemma give proof of this result. □

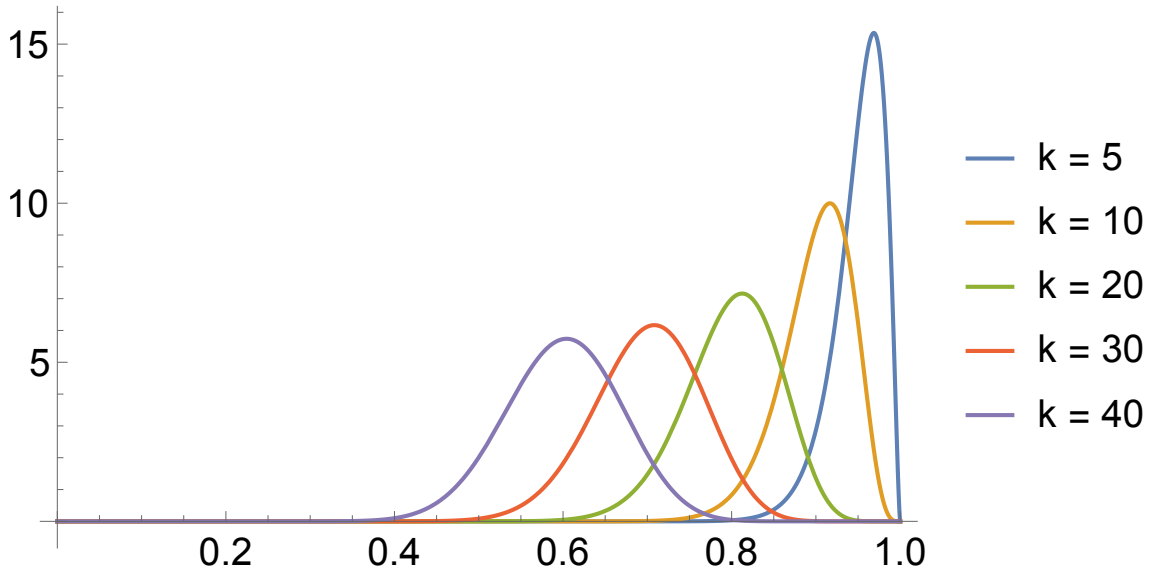

Figure 4.1: Plot of the probability density functions of the Beta $\left(\frac{N-k}{2}, \frac{k}{2}\right)$ distribution with $N = 100$ and the k values $k = 5, 10, 20, 30, 40$, showing how the distribution is concentrated near one with decreasing manifold dimensions k in relation to the fixed ambient dimension N.

Corollary 4.3.1. *The expectation and variance of* Δ^2 *are given by*

$$E(\Delta^2) = 1 - \frac{k}{N} \qquad \text{and} \qquad V(\Delta^2) = \frac{2}{N+2}\left[\frac{k}{N}\left(1 - \frac{k}{N}\right)\right]$$

Corollary 4.3.2. *If* $k \ll N$*, i.e. if the* k*-dimensional manifold* M *is of low dimension compared to the dimension of the ambient space* $\mathbb{R}^N$*, then the expectation* $E(\Delta^2) \approx 1$*. If the ratio* k/N *is fixed and* $N \to \infty$*, then the variance* $V(\Delta^2) \to 0$*.*

Knowing the distribution of Δ^2, and knowing in fact that it is a well-studied distribution, is quite fortuitous. Now, given any pair of dimensions k and N, we can easily compute the probability that Δ^2 is within a given tolerance of 1. The *regularized Beta function*, which we will denote by I, can be computed numerically by built-in functions in many popular programming languages (`betainc` in Matlab

[13], and `BetaRegularized` in Mathematica [25]), and

$$P(\Delta^2 < x) = I(x, (N-k)/2, k/2).$$

It is also useful to have an idea of how this quantity changes asymptotically as $N \to \infty$, which is described in the following theorem:

Theorem 4.3.2. *Let $s = k/N$, then*

$$P\left(\Delta^2 \leq 1 - 2s\right) \leq \frac{C}{N+2}$$

where C is a constant that depends only on s.

Proof. We will apply Chebyshev's Inequality, which states that for a random variable X with $\mu = E(X)$ and $\sigma^2 = V(X)$, we have

$$P(|X - \mu| \geq c\sigma) \leq \frac{1}{c^2},$$

where $c > 0$ is a parameter that can be chosen. We are interested in bounding the probability that Δ^2 is less than $1 - \epsilon$ for some ϵ, so we will choose the parameter c so that $\mu + c\sigma = 1$. Then we have

$$\begin{aligned}
P(|\Delta^2 - \mu| \geq c\sigma) &\leq \frac{1}{c^2} \\
P(\Delta^2 - \mu \geq c\sigma) + P(\mu - \Delta^2 \geq c\sigma) &\leq \frac{1}{c^2} \\
P(\Delta^2 \geq \mu + c\sigma) + P(\Delta^2 \leq \mu - c\sigma) &\leq \frac{1}{c^2} \\
P(\Delta^2 \geq 1) + P(\Delta^2 \leq \mu - c\sigma) &\leq \frac{1}{c^2} \\
P(\Delta^2 \leq \mu - c\sigma) &\leq \frac{1}{c^2}.
\end{aligned}$$

Now we can solve for our chosen value of c using the formulas for the expectation and

variance of $\Delta^2 \sim \text{Beta}((N-k)/2, k/2)$ found in Lemma 4.3.1, where we let $s = k/N$ to simplify the expressions:

$$\begin{aligned}
\mu + c\sigma &= 1 \\
1 - \frac{k}{N} + c\sqrt{\frac{2}{N+2}\left[\frac{k}{N}\left(1 - \frac{k}{N}\right)\right]} &= 1 \\
1 - s + c\sqrt{\frac{2}{N+2}s(1-s)} &= 1 \\
c\frac{\sqrt{2s(1-s)}}{\sqrt{N+2}} &= s \\
c &= \frac{s\sqrt{N+2}}{\sqrt{2s(1-s)}} \\
c &= \sqrt{\frac{s^2(N+2)}{2s(1-s)}} \\
c &= K\sqrt{N+2} \text{ where } K = \sqrt{\frac{s}{2(1-s)}}
\end{aligned}$$

Combining this with our one-sided inequality from Chebyshev's Inequality above, we have

$$\begin{aligned}
P\left(\Delta^2 \leq 1 - s - K\sqrt{N+2}\sqrt{\frac{2}{N+2}s(1-s)}\right) &\leq \frac{1}{K^2(N+2)} \\
P\left(\Delta^2 \leq 1 - s - \frac{s}{\sqrt{2s(1-s)}}\sqrt{2s(1-s)}\right) &\leq \frac{1}{K^2(N+2)} \\
P\left(\Delta^2 \leq 1 - 2s\right) &\leq \frac{C}{N+2} \text{ where } C = \frac{1}{K^2} = \frac{2(1-s)}{s}
\end{aligned}$$

□

Intuitively, the above result tells us that $P(\text{dist}(y+\eta u, M) \approx \eta) \approx 1$ when $k \ll N$, which is to say that the magnitude of our added noise term η is typically about the same as the distance from the noiseless data point $y \in M$. Another interpretation of this fact is that, on a high dimensional sphere, most of the area (and thus probability,

assuming a uniform density) is concentrated near the equator of the sphere. Yet another interpretation is that a random vector on the sphere is typically approximately orthogonal to a fixed k-dimensional plane through the center of the sphere.

4.4 Expected Squared Distance between Noisy Data Points and the Manifold

This section describes the most important theoretical result for the working of the algorithm, which relates the expectation $E(\eta^2|z)$ to the underlying quantity of interest $\text{dist}^2(z, M)$, which we work with instead of the non-squared distance for convenience. The main result from the previous section suggests that we can compute the squared distance function by computing random draws η^2. The main result of this section tells us the bias in this approach, which turns out to be directly related to the dimension k of the manifold M.

Theorem 4.4.1. *Suppose that M is a k-dimensional linear manifold in $\mathbb{R}^N$. Let $z = y + \eta u$ be a random variable where $y \sim \text{Unif}(M)$ and $\eta u \sim \mathcal{N}(0, \sigma^2 I_N)$. Then the squared distance function to the manifold M can be written as*

$$\text{dist}^2(z, M) = E(\eta^2|z) - k\sigma^2.$$

Note that as the k-dimensional volume of M is infinite, the distribution of y is an improper distribution.

Proof. By rotation and translation, we can assume without loss of generality that $M = \{y \in \mathbb{R}^N : y_{k+1} = y_{k+2} = \cdots = y_N = 0\}$. Suppose first that $k = 0$, so $M = \{0\} \subset \mathbb{R}^N$. In this case, $y = 0$ and $z = y + \eta u = \eta u$, and we can exactly recover η by computing $\eta = \|z\|_2$. Then the conditional expectation is given by $E(\eta^2|z) = \|z\|_2^2 = \text{dist}^2(z, M)$.

Now suppose that $k \geq 1$. To compute $E(\eta^2|z)$ as a function of z, we first note that η is a function of y and z, so we can compute the conditional expectation using the formula

$$\begin{aligned} E(\eta^2|z) &= \int_M (\eta(y,z))^2 p(y|z) d\hat{y} \\ &= \int_M (\eta(y,z))^2 \left(\frac{p(z|y)p(y)}{p(z)} \right) d\hat{y}, \quad \text{by Bayes' Theorem.} \end{aligned}$$

where $\hat{y} = (y_1, y_2, \ldots, y_k)$, $\hat{z} = (z_1, z_2, \ldots, z_k)$, and $d\hat{y}$ is the Lebesgue measure on $\mathbb{R}^k$. By assumption, $y \sim \text{Unif}(M)$ is an improper distribution, constructed as a limit of proper uniform distributions on the cube $[-c,c]^k \times \{\vec{0} \in \mathbb{R}^{N-k}\} \subset \mathbb{R}^N$, which for brevity we will simply write as $[-c,c]^k$ or just C. The proper uniform distribution of y on $[-c,c]^k$ is then given by

$$\begin{aligned} p_c(y) &= \text{Unif}([-c,c]^k)(\hat{y}) \\ &= \frac{1}{(2c)^k} \mathbb{1}_{[-c,c]^k}(\hat{y}) \end{aligned}$$

and the improper distribution $p(y)$ is the limit $\lim_{c\to\infty} p_c(y)$.

Note that $z = y + \eta u$ and $\eta u \sim \mathcal{N}(0, \sigma^2 I_N)$, thus

$$\begin{aligned} p(z|y) &= \mathcal{N}(y, \sigma I_N)(z) \\ &= (2\pi\sigma^2)^{-N/2} \exp\left(\frac{1}{2\sigma^2} \sum_{i=1}^{N} (z_i - y_i)^2 \right) \end{aligned}$$

With $p_c(y)$ and $p(z|y)$ computed, we can use the sum and product rules of prob-

ability as follows:

$$\begin{aligned}
p_c(z) &= \int_{\mathbb{R}^k} p(z|y)p_c(y)d\hat{y} \\
&= \frac{1}{(2c)^k}\int_C (2\pi\sigma^2)^{-N/2}\exp\left(\frac{\sum_{i=1}^N (z_i-y_i)^2}{2\sigma^2}\right)d\hat{y} \\
&= \frac{1}{(2c)^k}\int_C (2\pi\sigma^2)^{-N/2}\exp\left(\frac{\sum_{i=1}^k (z_i-y_i)^2}{2\sigma^2}\right)\exp\left(\frac{\sum_{i=k+1}^N (z_i-y_i)^2}{2\sigma^2}\right)d\hat{y} \\
&= \frac{1}{(2c)^k}(2\pi\sigma^2)^{-(N-k)/2}\exp\left(\frac{1}{2\sigma^2}\sum_{i=k+1}^N (z_i-y_i)^2\right)D_c
\end{aligned}$$

where

$$\begin{aligned}
D_c &= \int_C (2\pi\sigma^2)^{-k/2}\exp\left(\frac{\sum_{i=1}^k (z_i-y_i)^2}{2\sigma^2}\right)d\hat{y} \\
&= \int_C \mathcal{N}(\hat{z},\sigma I_k)(\hat{y})d\hat{y}
\end{aligned}$$

and hence $\lim_{c\to\infty} D_c = 1$. Therefore,

$$\begin{aligned}
p(z) &= \lim_{c\to\infty} p_c(z) \\
&= (2\pi\sigma^2)^{-(N-k)/2}\exp\left(\frac{1}{2\sigma^2}\sum_{i=k+1}^N (z_i-y_i)^2\right)
\end{aligned}$$

where $p(z)$ is an improper distribution.

Now we can compute the distribution needed to evaluate the desired conditional expectation:

$$\begin{aligned}
p_c(y|z) &= \frac{p(z|y)p_c(y)}{p_c(z)} \\
&= \frac{(2\pi\sigma^2)^{-N/2}\exp\left(\frac{1}{2\sigma^2}\sum_{i=1}^{N}(z_i-y_i)^2\right)\frac{1}{(2c)^k}\mathbb{1}_{[-c,c]^k}(\hat{y})}{\frac{1}{(2c)^k}(2\pi\sigma^2)^{-(N-k)/2}\exp\left(\frac{1}{2\sigma^2}\sum_{i=k+1}^{N}(z_i-y_i)^2\right)D_c} \\
&= \frac{(2\pi\sigma^2)^{-k/2}\mathbb{1}_{[-c,c]^k}(\hat{y})}{D_c}\exp\left(\frac{1}{2\sigma^2}\sum_{i=1}^{k}(z_i-y_i)^2\right) \\
p(y|z) &= \lim_{c\to\infty} p_c(y|z) \\
&= (2\pi\sigma^2)^{-k/2}\exp\left(\frac{1}{2\sigma^2}\sum_{i=1}^{k}(z_i-y_i)^2\right) \\
&= \mathcal{N}(\hat{z},\sigma^2 I_k)(\hat{y}).
\end{aligned}$$

Finally we can evaluate the conditional expectation itself:

$$\begin{aligned}
E(\eta^2|z) &= \int_M (\eta(y,z))^2 p(y|z)d\hat{y} \\
&= \int\left(\sum_{i=1}^{N}(y_i-z_i)^2\right)\mathcal{N}(\hat{z},\sigma^2 I_k)(\hat{y})d\hat{y} \\
&= \int\sum_{i=1}^{k}(y_i-z_i)^2\mathcal{N}(\hat{z},\sigma^2 I_k)(\hat{y})d\hat{y} + \int\sum_{i=k+1}^{N} z_i^2\mathcal{N}(\hat{z},\sigma^2 I_k)(\hat{y})d\hat{y} \\
&= I_1 + I_2
\end{aligned}$$

where

$$\begin{aligned}
I_1 &= \int \sum_{i=1}^{k} (y_i - z_i)^2 \mathcal{N}(\hat{z}, \sigma^2 I_k)(\hat{y}) d\hat{y} \\
&= \int \sum_{i=1}^{k} (y_i - z_i)^2 \prod_{j=1}^{k} \mathcal{N}(z_j, \sigma^2)(y_j) d\hat{y} \\
&= \sum_{i=1}^{k} \int (y_i - z_i)^2 \mathcal{N}(z_i, \sigma^2)(y_i) \prod_{j \neq i} \mathcal{N}(z_j, \sigma^2)(y_j) d\hat{y} \\
&= \sum_{i=1}^{k} \int (y_i - z_i)^2 \mathcal{N}(z_i, \sigma^2)(y_i) dy_i \prod_{j \neq i} \int \mathcal{N}(z_j, \sigma^2)(y_j) dy_j \\
&= \sum_{i=1}^{k} \int (y_i - z_i)^2 \mathcal{N}(z_i, \sigma^2)(y_i) dy_i \\
&= \sum_{i=1}^{k} \sigma^2 \\
&= k\sigma^2
\end{aligned}$$

and

$$\begin{aligned}
I_2 &= \int \left(\sum_{i=k+1}^{N} z_i^2 \right) (2\pi\sigma^2)^{-k/2} \exp\left(\frac{1}{2\sigma^2} \sum_{i=1}^{k} (z_i - y_i)^2 \right) d\hat{y} \\
&= \left(\sum_{i=k+1}^{N} z_i^2 \right) \int \mathcal{N}(\hat{z}, \sigma^2 I_k)(\hat{y}) d\hat{y} \\
&= \sum_{i=k+1}^{N} z_i^2.
\end{aligned}$$

We can now conclude that

$$\begin{aligned}
E(\eta^2 | z) &= k\sigma^2 + \sum_{i=k+1}^{N} z_i^2 \\
&= k\sigma^2 + \mathrm{dist}^2(z, M)
\end{aligned}$$

□

4.5 Training a Neural Network Function to Approximate the Squared Distance Function

We start by choosing a class of functions V. For each numerical experiment in the later section, V is chosen to be a class of neural network functions given by a specified architecture. We will assume that each function $f_\theta \in V$ is indexed by a vector of parameters θ. In order to learn a function that approximates the squared-distance function, $\text{dist}^2(\cdot, M)$, we specify a loss function and then find the function $f \in V$ that minimizes this loss function.

Throughout, we choose the loss function to be the mean square error

$$\begin{aligned} L(\theta) &= \frac{1}{nm} \sum_{i=1}^{n} \sum_{j=1}^{m} \left(\Phi_\theta(y^i + \eta_j^i u_j^i) - (\eta_j^i)^2 \right)^2 \\ &= \frac{1}{nm} \sum_{i=1}^{n} \sum_{j=1}^{m} \left(\Phi_\theta(z_j^i) - (\eta_j^i)^2 \right)^2 \end{aligned}$$

where $z_j^i = y^i + \eta_j^i u_j^i$ denotes the noise-corrupted vector. We now want to know what function Φ minimizes this loss function in the best-case scenario where we have unlimited training data and a perfectly flexible function class V.

Lemma 4.5.1. *Let the function class V be the class of all functions and let the loss function $L(\theta)$ be the mean square error function. The function $\Phi_\theta(z)$ that minimizes L_θ given unlimited training data is $\Phi_\theta(z) = E(\eta^2|z)$, almost surely.*

Proof. Since we are interested in finding the value $\Phi_\theta(z)$, we begin by fixing the value z. Now, given unlimited training data, we have the loss function given by the limit

$$\begin{aligned} L(\theta) &= \lim_{n,m\to\infty} \frac{1}{nm} \sum_{i=1}^{n} \sum_{j=1}^{m} \left(\Phi_\theta(z) - (\eta_j^i)^2\right)^2 \\ &= \lim_{n,m\to\infty} \frac{1}{nm} \sum_{i=1}^{n} \sum_{j=1}^{m} (\Phi_\theta(z))^2 - 2\Phi_\theta(z)(\eta_j^i)^2 + (\eta_j^i)^4 \\ &= (\Phi_\theta(z))^2 - 2\Phi_\theta(z) \lim_{n,m\to\infty} \frac{1}{nm} \sum_{i=1}^{n} \sum_{j=1}^{m} (\eta_j^i)^2 + \lim_{n,m\to\infty} \frac{1}{nm} \sum_{i=1}^{n} \sum_{j=1}^{m} (\eta_j^i)^4 \\ &= (\Phi_\theta(z))^2 - 2\Phi_\theta(z) E(\eta^2|z) + E(\eta^4|z) \end{aligned}$$

where the convergence $\lim_{n,m\to\infty} \frac{1}{nm} \sum_{i=1}^{n} \sum_{j=1}^{m} (\eta_j^i)^p \to E(\eta|z)^p$ is almost sure convergence given by the strong law of large numbers.

To minimize this quadratic function of $\Phi_\theta(z)$, note that

$$\frac{d}{dc}\left(c^2 - 2cE(\eta^2|z) + E(\eta^4|z)\right) = 2c - 2E(\eta^2|z)$$

and setting this equal to zero yields $c = E(\eta^2|z)$. This value is clearly a minimum, as the derivative changes from begin negative to positive at this value. Thus $\Phi_\theta(z) = E(\eta^2|z)$ minimizes the loss $L(\theta)$ almost surely. □

4.6 Projecting to the Manifold Using the Learned Approximate Squared Distance Function

Our goal in computing the approximate squared distance function is to ultimately be able to compute information related to the underlying k-dimensional manifold M. To this end, we use a gradient-descent-type algorithm to use with the approximate squared distance function to compute a function $\Psi\colon \mathbb{R}^N \to M$ which approximates proj_M, the projection function to M. The gradient-descent-type algorithm we use

Algorithm 4.1: k-Gradient Descent

Input: Initial point $x_0 \in \mathbb{R}^N$, function Φ, dimension k of curve to converge to, stepsize γ.
Output: Output point $x \in \mathbb{R}^N$.

while (stopping criterion not met) **do**
    $i = i + 1$
    $g = \nabla \Phi$
    $H = \nabla^2 \Phi$
    $U, S, V = \text{svd}(H)$
    $M = S[:, 1 : (N - k)]$
    $x_i = x_{i-1} - \gamma M M^{\mathrm{T}} g$
end

Figure 4.2: The k-Gradient Descent Algorithm

we will refer to as *k-gradient descent*, which is essentially the same as the Subspace Constrained Mean-Shift (SCMS) algorithm from [15] and [18], based on the definition of a *principal curve* from [7] and [8], though the function we apply it to is not a probability density function, hence the need for different terminology.

A description of the k-gradient descent is given in 4.2. The main idea behind k-gradient descent is that we want the algorithm to converge to a k-dimensional manifold. Standard gradient descent converges to a set of local minima, typically a set of isolated points, i.e., a 0-dimensional manifold. The k-gradient descent algorithm generalizes gradient descent, which can be thought of as k-gradient descent with $k = 0$. Trying to use standard gradient descent for this application results in points "sliding" along the k-dimensional manifold to converge to local minima instead of converging to the nearest point on the manifold (see Figure 4.3 for an illustration of this phenomenon).

4.7 Applications of Projecting to the Manifold

Common applications of projecting to the manifold are: denoising, sparse error correction, erasure completion, and "style transfer" via projection (see the relevant section below for precisely what this means). For applications to nonlinear manifolds, it

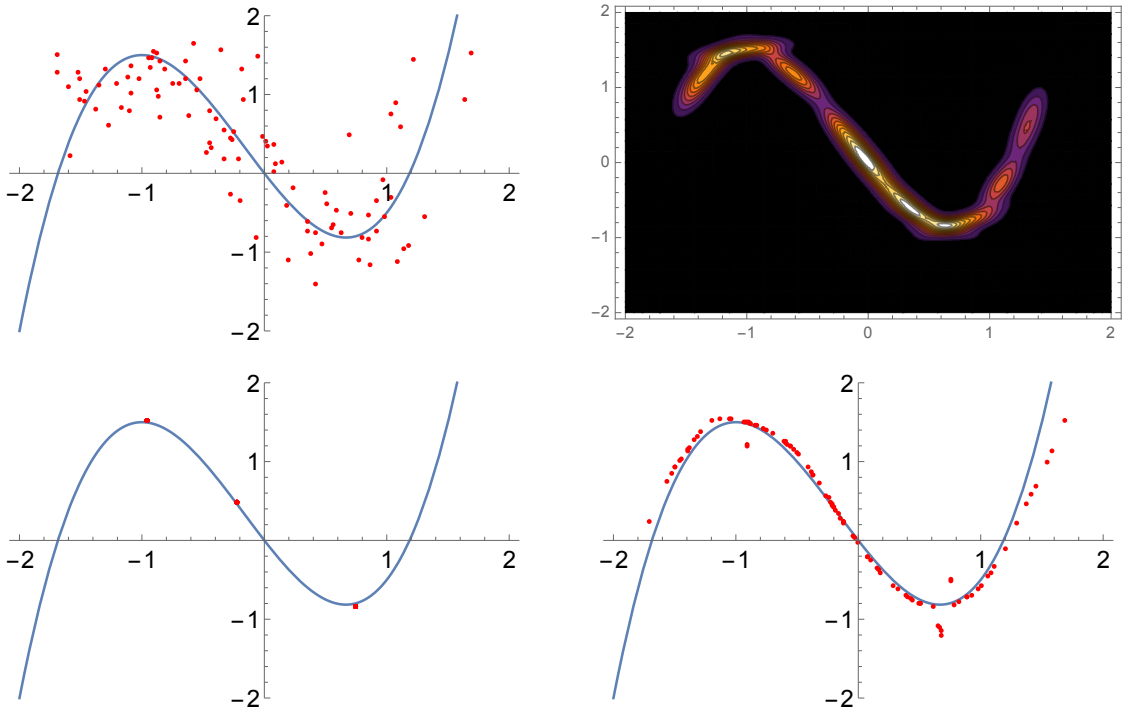

Figure 4.3: The manifold M is the graph of $f(x) = x^3 + \frac{1}{2}x^2 - 2x$, and points $y_i \in M$ are chosen as the $(x_1, f(x_1))$-values where the x_1-values are chosen uniformly at random on $[-1.5, 1.5]$. The height function Φ (level curves plotted in the top right panel) in this example is given by $\Phi(x) = -\sum_{i=1}^{30} G_{\Sigma_i}(x - y_i)$ where G_{Σ_i} is a Gaussian kernel with covariance matrix Σ_i. Each Σ_i is constructed by $\Sigma_i = U_i\Sigma$ where $\Sigma = \text{Diag}(.5, .05)$ and U_i is unitary with the first column vector as the unit tangent vector to $f(x)$. Normally distributed noise is added to the points y_i to get the noisy points z_i (top left). Projected values for the k-gradient descent algorithm are shown with $k = 0$ (bottom left) and $k = 1$ (bottom right). The standard gradient descent algorithm maps all of the z_i to only three local minima, whereas setting $k = 1$ better represents the 1-dimensional structure of the original manifold.

is important to know that error correction and erasure completion are not always exactly recovered, even when the projection function is known precisely. In particular, nonlinear manifolds can have finite *reach*, which limits the effectiveness of this procedure:

Definition 4.7.1. The *reach* of a manifold $M \subset \mathbb{R}^N$ is the largest number $\tau \geq 0$ such that any point $x \in \mathbb{R}^N$ with $\text{dist}(x, M) < \tau$ has a unique nearest point $\text{proj}_M(x) \in M$.

Data Corrupted by Noise

We define noise as a dense (i.e. nonsparse) corruption, possibly affecting every coordinate with some added corruption ηu. Given a corrupted data point $y + \eta u$, so long as the magnitude of the noise is not too large then we can recover the clean data point y by projecting $\hat{y} = \text{proj}_M(y + \eta u)$ using the k-gradient descent algorithm to compute the projection.

Supposing that Φ is a good approximation for $\text{dist}(\cdot, M)$, then so long as

$$\text{dist}(y, M) < \text{reach}(M)$$

there is a unique point $\text{proj}_M(y) \in M$ with $\text{dist}(\text{proj}_M(y), M) = \text{dist}(y, M)$. The reach of the manifold depends on how curved the manifold is, and we make no attempt in this dissertation to estimate that value. We are content with saying that "small" magnitudes of noise will be able to be corrected using projection.

Data Corrupted by Erasures

We define erasures to be coordinates of a data point y that are missing or known to be corrupted of possibly arbitrarily large magnitude, making their informational content negligible. It is important to note that the locations of erasures are known.

If it is known that clean data belongs to a low-dimensional manifold, then it is possible that missing coordinates in the data points can be successfully filled in if there is a unique point on the manifold that has the given coordinates in common with the given data point. Using the approximation Φ of the squared distance function, we can use *partial k-gradient descent*, defined as the k-gradient descent algorithm using the *partial gradient* $\nabla\Phi|_\omega$ where $\omega \in 2^{1,2,\ldots,N}$ is the support of the data, i.e. the set of erasures is $\{1, 2, \ldots, N\} \setminus \omega = \omega^c$. By only updating coordinates known to be erased, we keep the known coordinates the same and eventually converge to a point on the manifold M that has those coordinates in common with the given data point.

Data Corrupted by Errors

We define errors to be sparse corruptions, but allow the magnitudes of errors to be quite large in comparison to the magnitude for noise. Given a corrupted data point $y + e$ where $e \in \mathbb{R}^N$ is a sparse vector, if we can detect the support of e then we can recover the clean data point y by treating the support of e as erasures. We can estimate the support of the error vector in a greedy fashion, by finding which the largest coordinates of $|\operatorname{proj}_M(y) - y|$ after computing $\operatorname{proj}_M(y)$ by running gradient descent.

"Style Transfer" via Projection

This is another common example in machine learning, where we want to take a data point belonging to one manifold M^1 and project it onto another manifold M^2 to get the "style" of the manifold M^2 but the original data from the original point $y \in M^1$.

For example, if we know how to project on to the manifold of images of person A's face, then given a photo of person B in some pose, we could try to synthesize an image of person A in the same pose as person B by projecting the image onto the manifold for person A's face.

4.7.1 Classifying using Distance

A common problem in machine learning is to classify data as belonging to one of several different classes. This is known as the *classification problem*. In particular, if there are only two classes under consideration, we call this the *binary classification problem*, and if there are more than two we call this the *multiclass classification problem*. We can use the squared distance function $\text{dist}^2(\cdot, M)$ or our approximation Φ of the squared distance function to solve this common problem in machine learning.

Binary Classification using Distance

In the case of binary classification, we assume that one of the classes of data corresponds to data from the low-dimensional manifold M. We will call this class the *manifold class*, and the other class the *non-manifold class*. Data in the manifold class does not have to necessarily be exactly on the manifold M, but merely close to it. Because of this, and because there will generally be some approximation error to consider from using the function Φ instead of the true distance function $\text{dist}(\cdot, M)$, we choose a tolerance $\epsilon > 0$ and classify points y as belonging to the manifold class if $\Phi(y) < \epsilon$ and classifying points as belonging to the non-manifold class if $\Phi(y) \geq \epsilon$. Specifically, we construct the classifier function $\mathcal{C} \colon \mathbb{R}^N \to \{0, 1\}$ using the approximate distance function Φ as

$$\mathcal{C}(y) = \begin{cases} 1, & \text{if } \Phi(y) < \epsilon \\ 0, & \text{if } \Phi(y) \geq \epsilon \end{cases}$$

for the chosen threshold $\epsilon > 0$.

Multiclass Classification using Distance

In the case of multiclass classification with C different classes, there are two possibilities: We can have each of the C classes be representing the class of data close to one of C manifolds M^i, $i = 1, 2, \ldots, C$, or we can also choose one of the classes to

be a non-manifold class without an associated manifold M, but the remaining $C-1$ classes need to have associated manifolds M^i, $i = 1, \ldots, C-1$. In this case we can refer to the non-manifold class as an "other" or "unknown" class for data that does not fit any of the other classes. For each of the manifold classes, we can generate training data and compute the approximate squared distance function Φ^g.

To classify in the case where every class has an associated manifold, we classify y as belonging to whichever class has the smallest approximate squared distance $\Phi^g(y)$. Specifically, we construct the classifier function $\mathcal{C}\colon \mathbb{R}^N \to \{0, 1, \ldots C\}$ using the approximate squared distance function Φ as

$$\mathcal{C}(y) = \underset{g\in\{1,2,\ldots,C\}}{\operatorname{argmin}} \Phi^g(y).$$

To classify in the case where there is a non-manifold class, we pick a threshold $\epsilon > 0$ and classify y as belonging to the non-manifold class $g = C$ if $\Phi^g(y) > \epsilon$ for all $g = 1, 2, \ldots, C-1$, and otherwise classify y as belonging to whichever class g has the smallest approximate squared distance $\Phi^g(y)$. Specifically, we construct the classifier function $\mathcal{C}\colon \mathbb{R}^N \to \{0, 1, \ldots C\}$ using the approximate squared distance function Φ as

$$\mathcal{C}(y) = \begin{cases} \operatorname{argmin}_{g\in\{1,2,\ldots,C-1\}} \Phi^g(y) & \text{if } \Phi^g(y) < \epsilon \text{ for some } g \\ C & \text{if } \Phi^g(y) > \epsilon \text{ for all } g \end{cases}$$

for the chosen threshold $\epsilon > 0$.

4.8 Numerical Experiments

We test the algorithm combining the data generation process, the training of the neural network to approximate the squared distance function, and the k-gradient descent algorithm with a variety of numerical experiments. There are several sources

of approximation error between Ψ and proj_M, including

(a) *Small sample error*, arising from the fact that our given data set $\{y_i\}$ is a finite sample from the manifold M, providing incomplete information about M. This error is impossible to completely avoid unless the manifold is known at the start, in which case there is no problem to solve.

(b) *Nonlinearity error*, arising from the violation of the assumption that the underlying manifold M is a linear manifold. The result in 4.4.1 will not hold exactly, but will hold approximately depending on how well the manifold can be approximated near a point $y \in M$ by the tangent plane T_y. Assuming that k is known, the nonlinearity error is given by $E_{\text{nl}} = \text{dist}^2(z, M) - E(\eta^2|z) + k\sigma^2$.

(c) *Variable dimension error*, arising if the manifold M does not have a fixed dimension, i.e., if some points on the "manifold" have neighborhoods that are homeomorphic to $\mathbb{R}^{k_1}$ and others have neighborhoods homeomorphic to $\mathbb{R}^{k_2}$ where $k_1 \neq k_2$.

(d) *Function approximation error*, arising from the fact that Φ must be from a limited function class V, as opposed to the class containing all functions. The function class must be limited to avoid overfitting, but if it is too limited then there might not exist a good approximation in V for the function $E(\eta^2|z)$.

(e) *Training error*, arising from the fact that the optimal $\Phi \in V$ might not be found precisely.

In the following numerical experiments, we will focus on studying nonlinearity error, as the theoretical results above say nothing about how the approach for manifold learning presented here can deal with nonlinear manifolds.

4.8.1 Experiment 1: Bent Line Segments

For this experiment, the manifold M is given by $g([0,1])$ where

$$g(t) = \begin{cases} (1-2t, 0) & , \text{ if } t < 1/2 \\ (2t-1)(\cos(\theta), \sin(\theta)) & , \text{ if } t \geq 1/2 \end{cases}.$$

The values of θ are chosen to be $\theta = \pi, \pi/2, \pi/4, \pi/8, \pi/16$ and $\pi/32$, and the standard deviation of the noise η is chosen to be $\sigma = 0.1$. For this example, we only want to study the nonlinearity error and how it affects the approximate projection Ψ, which is possible in this simple example because the manifold is simple enough for us to calculate $E(\eta^2|z)$ by numerical integration (we can ignore the constant $k\sigma^2$ for the purposes of gradient descent). For the different values of θ, we calculate the function $E(\eta^2|z)$ on a grid of points and use cubic splines to interpolate on the grid. With the interpolated function, we can compute the gradient and the Hessian to use k-gradient descent for computing the approximate projection function Ψ. Noisy data points $z_i = y_i + \eta_i u_i$ are generated and then we plot the points $\Psi(z_i)$.

In Figure 4.4, the level curves of $E(\eta^2|z)$ are shown, along with the manifold M and the points $\Psi(z_i)$. There are two things to note in this example. The first is that the k-gradient descent algorithm can sometimes converge to k-dimensional "valleys" that are not part of the manifold. This happens in two ways in these examples, both by extending the length of set of fixed points of Ψ past the length of the manifold M, and by false "valleys" appearing near the sharply curved pieces of the manifold. The second, more important thing to note is that the more highly curved the manifold (i.e., the smaller the angle θ in this example) the worse the "valley" that corresponds to the manifold approximates the manifold.

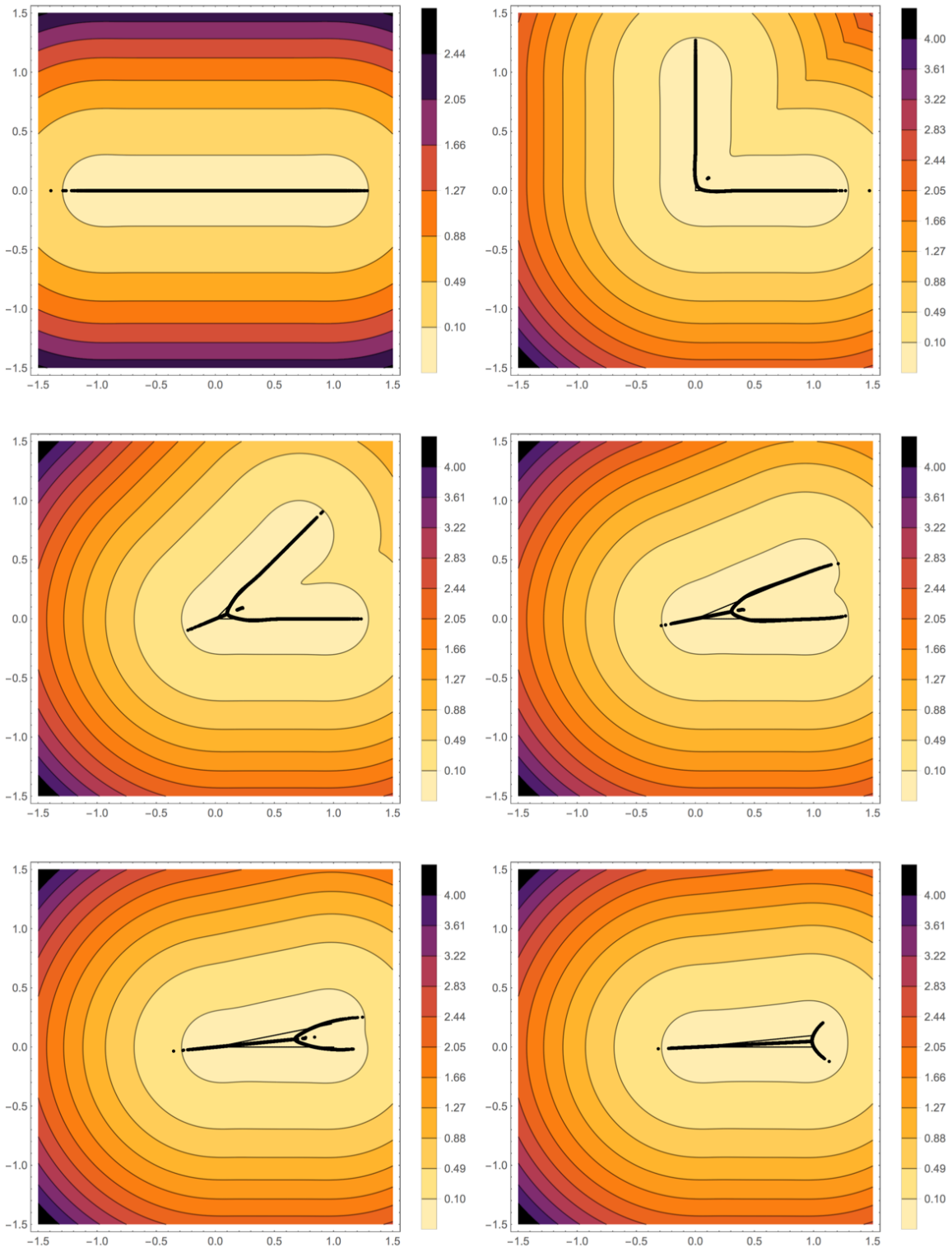

Figure 4.4: Level curves of $E(\eta^2|z)$ and projected points $\Psi(z)$.

4.8.2 Experiment 2: Experiments on Smooth Curves

In these experiments, we test the algorithm on smooth curves in $\mathbb{R}^2$ and $\mathbb{R}^3$. For the examples in $\mathbb{R}^2$, we compute $E(\eta^2|z) - k\sigma^2$ numerically and show how it differs from the desired quantity $\text{dist}^2(z, M)$. In addition, we test the approximate projection function Ψ on some noisy data points of the form $z = y + \eta u$. Note that the difference will depend on the standard deviation σ of the noise vector ηu. The smaller the standard deviation, the smaller the "neighborhoods" considered, and the more linear the manifold appears from the perspective of the noise.

Experiment 2a: Circle in $\mathbb{R}^2$

For this experiment, the manifold M is given by the circle $M = \{x \in \mathbb{R}^2 \colon \|x\| = 2\}$. Level curves for the squared distance function, the approximate squared distance function, and their difference are given in Figure 4.5. Figure 4.6 shows the sample noisy data points (generated with $\sigma = 0.25$) and their images under the approximate projection function Ψ.

Experiment 2b: Flower Curve in $\mathbb{R}^2$

For this experiment, the manifold M is given by $g([0, 1])$ where $g(t) = (\cos(10 * \pi t) + 3)(\cos(2\pi t), \sin(2\pi t))$. Level curves for the squared distance function, the approximate squared distance function, and their difference are given in Figure 4.7. Figure 4.8 shows the sample noisy data points (generated with $\sigma = 0.3$) and their images under the approximate projection function Ψ.

Experiment 2c: Circle in $\mathbb{R}^3$

To test the projection function Ψ, we begin by drawing $n = 30$ points $\{y^i\}_{i=1}^n$ uniformly from the manifold M, generated by randomly rotating the unit circle S^1 in $\mathbb{R}^3$. The approximate squared distance function Φ is chosen to be a neural network

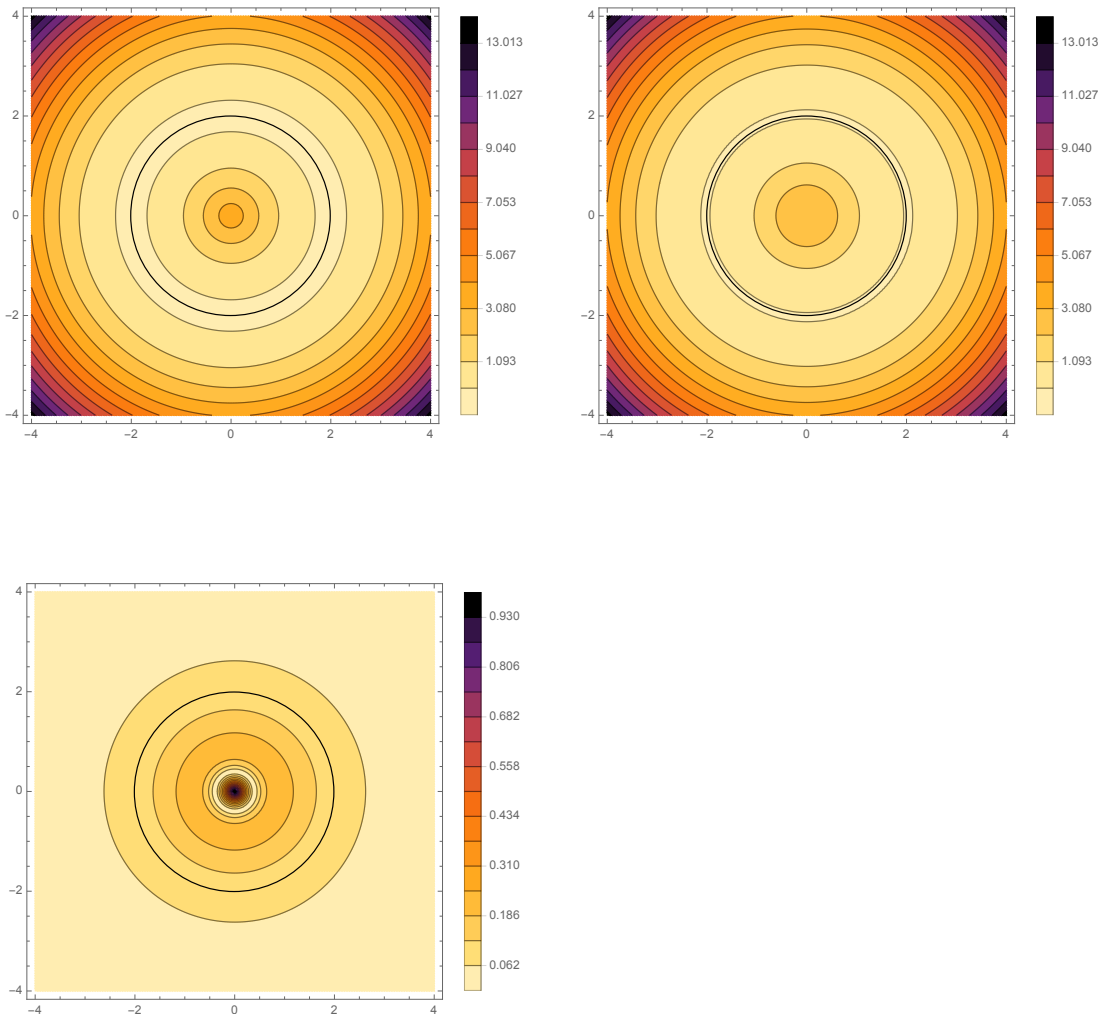

Figure 4.5: The manifold $M = \{x \in \mathbb{R}^2 \colon \|x\| = 2\}$ is shown with the level curves of the squared distance function (top left), the level curves of the approximate squared distance function (top right) image, and the level curves of the difference of the approximate and the true squared distance function (bottom left).

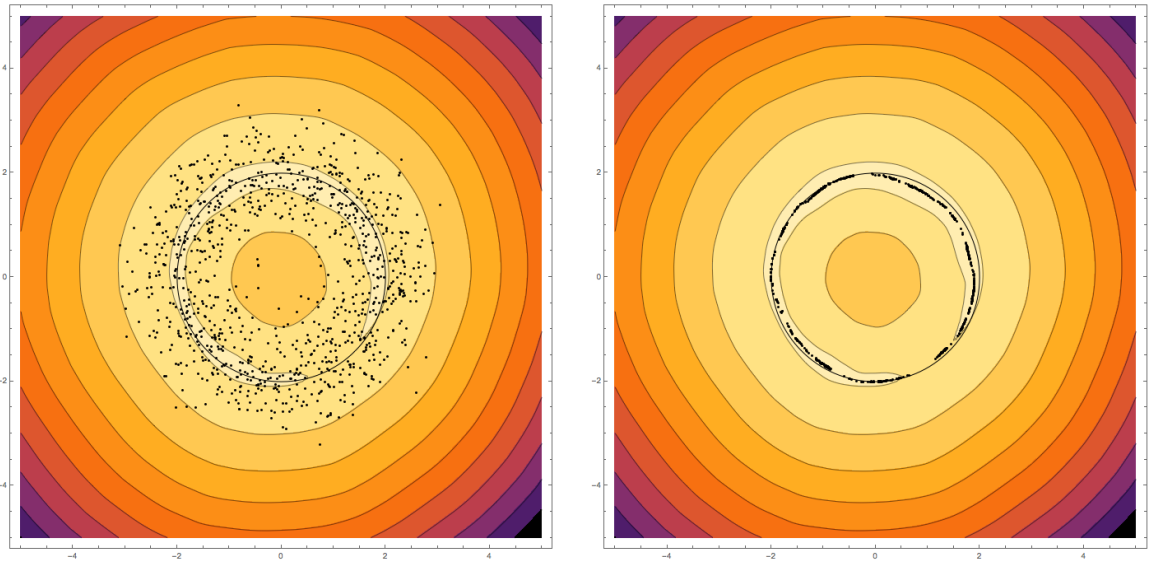

Figure 4.6: Noisy data points and their approximate projection under Ψ.

Experiment with added Noise	Mean = 0.0638	Median = 0.0458
Experiment with added Errors	Mean = 0.0491	Median = 0.0217

Table 4.1: Squared distance from $\Psi(y^i)$ to the manifold M from Experiment 2c, given by a random rotation of the unit circle $S^1 \subset \mathbb{R}^3$. Added noise is normally distributed with standard deviation $\sigma_z = 0.5$ and added errors are normally distributed with standard deviation of $\sigma_e = 10$ affecting two of the three coordinates.

with one hidden layer with 25 nodes, where the relu activation function is used for the hidden layer and the $x_+^2 = x^2 \mathbb{1}_{x \geq 0}$ activation function used for the output layer. The Adam [10] optimizer is used for training. Each training step consists of generating new random training set data $\{z_j^i\} = \{y^i + \eta_j^i u_j^i\}$ and then using the Adam optimizer on that training set. For a test set, we generate 10,000 data points on the circle and then add a noise vector to them, then compute Ψ of each data point. The true distance function to the circle is easy to compute, so we compare the distance from $\Psi(x)$ to the circle for each data point. Results are shown in histogram form in Figure 4.9, with the mean and median show in Table 4.1.

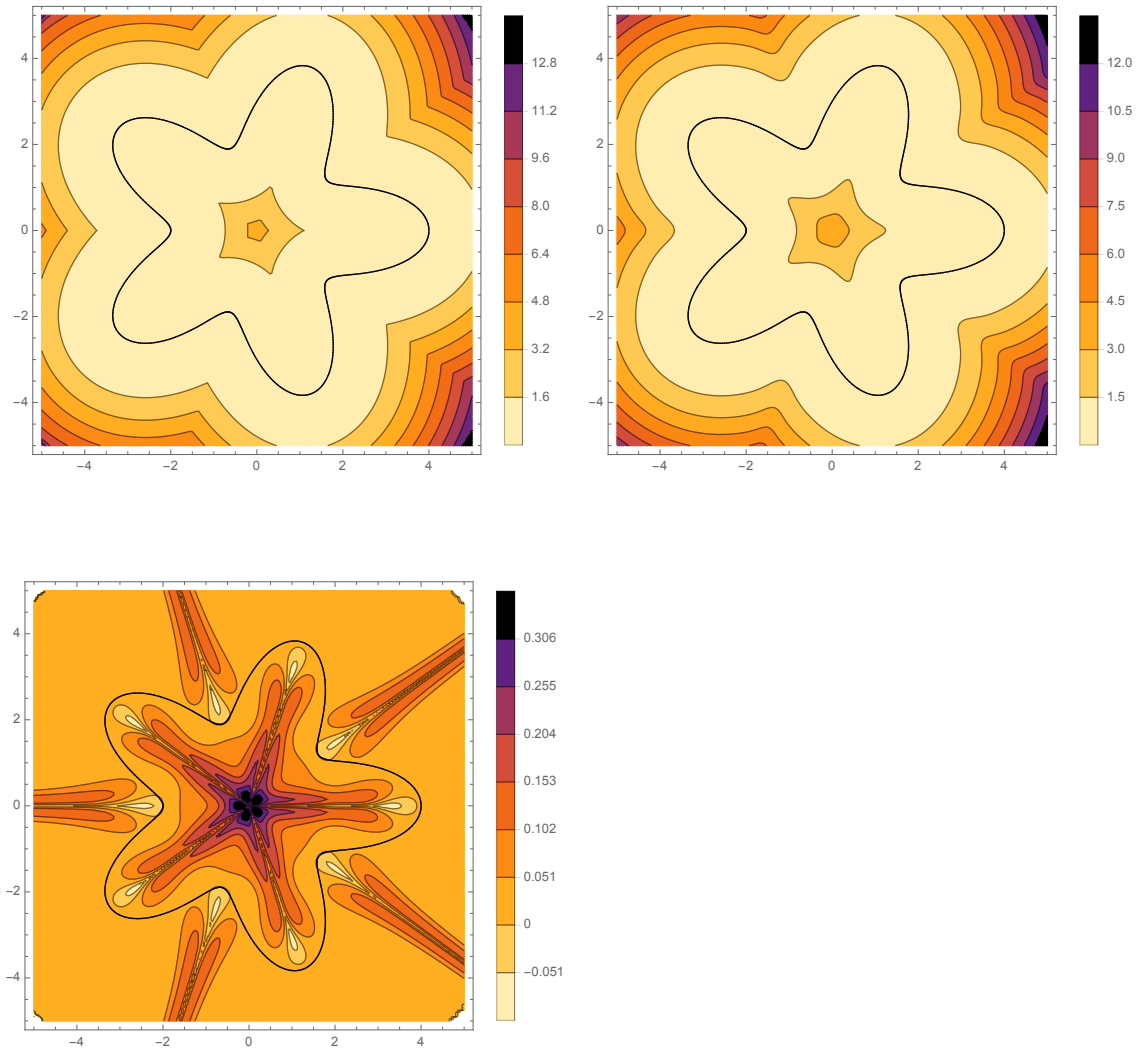

Figure 4.7: The manifold M given by the image of the function $g(t) = (\cos(10\pi t) + 3)(\cos(2\pi t), \sin(2\pi t))$ for $t \in [0, 1]$ is shown with the level curves of the squared distance function (top left), the level curves of the approximate squared distance function (top right) image, and the level curves of the difference of the approximate and the true squared distance function (bottom left).

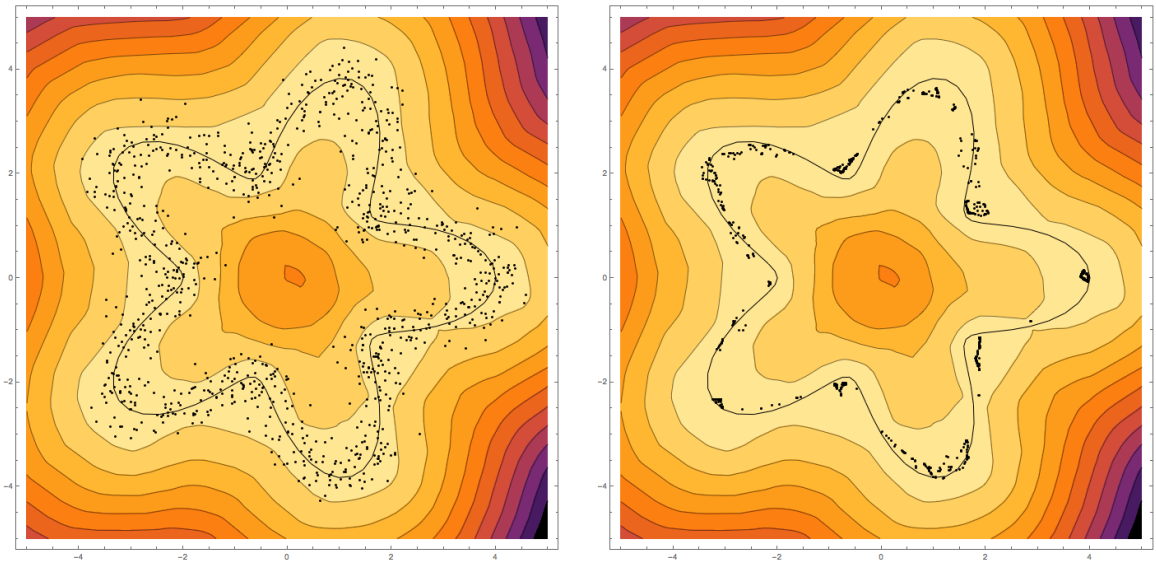

Figure 4.8: Noisy data points and their approximate projection under Ψ.

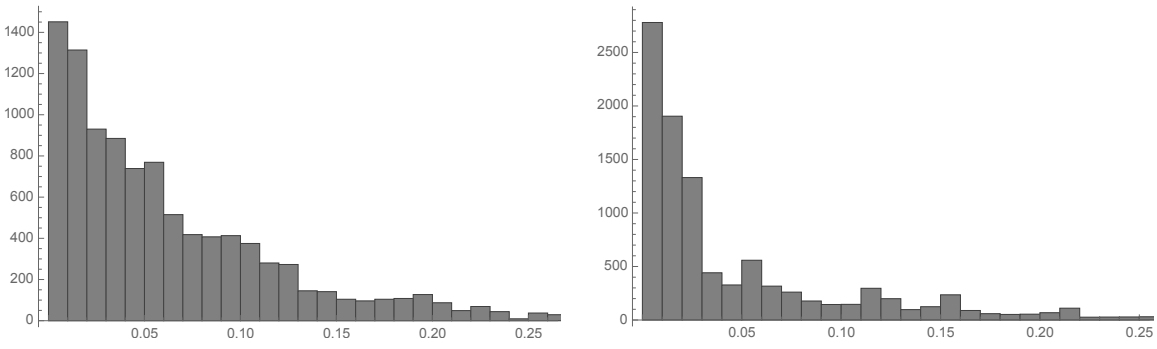

Figure 4.9: Histogram of the squared distance from $\Psi(y^i)$ to the manifold M, which in this experiment is a random rotation of the unit circle $S^1 \subset \mathbb{R}^3$.

4.8.3 Experiment 3: MNIST Experiments

Experiment 3a: MNIST Handwritten Digit Classification

In this experiment, we test the algorithm's ability to classify on real world data. We use data from the MNIST database of handwritten digits, each of which is undersampled by taking a grid of every other pixel. We can think of each class of handwritten digits as forming a manifold M^g with $g = 0, 1, \ldots, 9$. In this experiment, we analyze just the manifolds corresponding to the digits "0" and "1." We learn the two functions Φ^0 and Φ^1 corresponding to those two manifolds that approximate the squared distance M^0 and M^1 respectively. The function class V for this experiment is chosen as the set of neural network functions with two hidden layers, each with size 400. The activation function for the hidden layers is chosen to be the leaky relu function given by

$$f(x) = \begin{cases} x, & \text{if } x \geq 0 \\ .01x, & \text{if } x < 0 \end{cases}.$$

As with the other experiments, the activation on the final layer is x_+^2 to help with approximating the squared distance function. We choose the random noise $\sigma = 0.4$ for both manifolds. Once we have trained Φ^g for each manifold, we can use these to classify new data points y as being either "0" or "1" based on whether or not the approximate squared distance to M^0 or to M^1 is smaller. That is to say, the empirical classification $\hat{g}(y)$ of a new data point y is given by

$$\hat{g}(y) = \begin{cases} 0, & \text{if } \Phi^0(y) < \Phi^1(y) \\ 1, & \text{if } \Phi^0(y) > \Phi^1(y) \end{cases}$$

Classification accuracy on a test set of 5900 new data points is given in Table 4.2

Classification Accuracy of Test Points from M^0	$\text{acc}_0 = 99.98$
Classification Accuracy of Test Points from M^1	$\text{acc}_1 = 98.64$

Table 4.2: Classification accuracy on test sets of 5900 new data points each from M^0 and M^1.

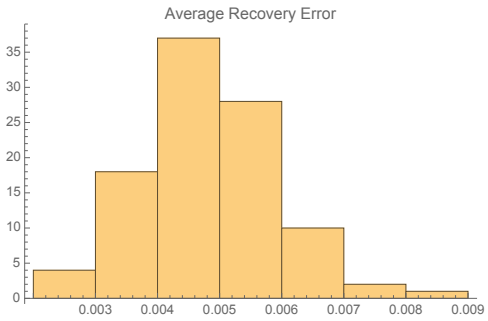

Figure 4.10: A histogram of the average recovery error for Experiment 3b.

Experiment 3b: MNIST Handwritten Digit Error Correction

In this experiment, we use the trained Φ^0 from Experiment 3a above to calculate the approximate projection function Ψ^0. We choose a test set of 100 images of the digit 0. We add a sparse error set to each of the data points y in this experiment by choosing 15 pixels and setting their value to the maximum value of 1 to form a corrupted data point $\overline{x}$. Gradient descent ($k = 0$) is run on the corrupted entries to find $\hat{y} = \Psi(y)$, and the recovery error is computed by $\|y - \hat{y}\|/N$, where $N = 196$ is the size of the undersampled MNIST images. A histogram of the average recovery error over the set of test images is given in Figure 4.10, with the mean average recovery error being 0.00478.

Bibliography

[1] R. W. Bailey. Distributional identities of beta and chi-squared variates: A geometrical interpretation. *The American Statistician*, 46(2), May 1992.

[2] C. Byrne. A unified treatment of some iterative algorithms in signal processing and image reconstruction. *Inverse Problems*, 20:103–120, 2004.

[3] P. L. Combettes and V. R. Wajs. Signal recovery by proximal forward-backward splitting. *Multiscale Model. Simul.*, 4(4):1168–1200, 2005.

[4] C. Fefferman, S. Mitter, and H. Narayanan. Testing the manifold hypothesis. *Journal of the American Mathematical Society*, 29(4):983–1049, Feb 2016.

[5] P. W. Gallagher and Z. Tu. Properties of Pseudocontractive Updates in Convex Optimization. *arXiv e-prints*, page arXiv:1405.7741, May 2014.

[6] E. Golshtein and N. Tretyakov. *Modified Lagrangians and Monotone Maps in Optimization*. Wiley Interscience series in Discrete Mathematics, 1996.

[7] T. Hastie. *Principal Curves and Surfaces*. PhD thesis, Stanford University, 1984.

[8] T. Hastie and W. Stuetzle. Principal curves. *Journal of American Statistical Association*, 84:502–516, 1989.

[9] KDD Cup and Workshop. *The Netflix Prize*, Proceedings of KDD Cup and Workshop, August 12, 2007.

[10] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2014.

[11] Y. Ma and Y. Fu. *Manifold Learning Theory and Applications*. CRC Press, 2011.

[12] R. W. Mann. Mean value methods in iteration. *Proceedings of the American Mathematical Society*, 4(3):506–510, 1953.

[13] MathWorks. Matlab documentation. `https://www.mathworks.com/help/matlab/ref/betainc.html`. Accessed: 2019-11-03.

[14] A. Mead. Review of the development of multidimensional scaling methods. *Journal of the Royal Statistical Society*, 41(1):27–39, 1992.

[15] K. Mohammed and H. Narayanan. Manifold learning using kernel density estimation and local principal components analysis, 2017.

[16] L. T. Nguyen, J. Kim, and B. Shim. Low-rank matrix completion: A contemporary survey. *IEEE Access*, 7:94215–94237, 2019.

[17] N. Ogura and I. Yamada. Non-strictly convex minimization over the fixed point set of an asympotically shrinking nonexpansive mapping. *Numer. Funct. Anal. and Optimiz.*, 23(1,2):113–137, 2002.

[18] U. Ozertem and D. Erdogmus. Locally defined principal curves and surfaces. *Journal of Machine Learning Research*, 12:1249–1286, 2011.

[19] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

[20] B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1319, 1998.

[21] J. B. Tenenbaum, V. d. Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

[22] V. Vasin and A. Ageev. *Ill-Posed Problems with A Priori Information.* De Gruyter, 1995.

[23] S. Villa. Proximal methods. Unpublished manuscript, available online.

[24] Z. Wang, M.-J. Lai, Z. Lu, W. Fan, H. Davulcu, and J. Ye. Orthogonal rank-one matrix pursuit for low rank matrix completion. *SIAM Journal on Scientific Computing*, 37(1):A488–A514, Jan 2015.

[25] Wolfram. Mathematica documentation. `https://reference.wolfram.com/language/ref/BetaRegularized.html`. Accessed: 2019-11-03.

[26] J. Wright, A. Ganesh, S. Rao, Y. Peng, and Y. Ma. Robust principal component analysis: Exact recovery of corrupted low-rank matrices via convex optimization. In Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 2080–2088. Curran Associates, Inc., 2009.