# Low-Rank Matrix Completion

Ryan Kennedy

Written Preliminary Examination II

Dept. of Computer and Information Science

University of Pennsylvania

May 10, 2013

**Abstract**

While datasets are frequently represented as matrices, real-word data is imperfect and entries are often missing. In many cases, the data are very sparse and the matrix must be filled in before any subsequent work can be done. This optimization problem, known as matrix completion, can be made well-defined by assuming the matrix to be low rank. The resulting rank-minimization problem is NP-hard, but it has recently been shown that the rank constraint can be replaced with a nuclear norm constraint and, with high probability, the global minimum of the problem will not change. Because this nuclear norm problem is convex and can be optimized efficiently, there has been a significant amount of research over the past few years to develop optimization algorithms that perform well.

In this report, we review several methods for low-rank matrix completion. The first paper we review presents an iterative algorithm to efficiently complete extremely large matrices. The second paper formulates the problem directly as matrix factorization, which can be optimized using several different methods. Next, we present an algorithm that changes the way that the optimization is carried out in order to achieve better convergence on ill-conditioned matrices. Finally, we describe a related online algorithm for matrix completion. We show how these algorithms are related and directly compare them using experiments on synthetic data.

# Contents

# 1 Introduction

## 1.1 Motivation

It is often the case that real-world datasets can be described in terms of a matrix with missing data. For example, suppose we have a sensor network of $n$ nodes and would like to know the distance between any two sensors. Because each sensor has only a limited sensing range, it can determine the distance between itself and other sensors within a fixed radius. This can be described by a $n \times n$ distance matrix, where the distance between sensors that are too far apart is not directly known. Another example is the *structure-from motion* problem, where we are given a set of images or a video and wish to construct a 3D model of the scene. A common approach to solving this problem is to first identify points in separate images that represent the same 3D location and then construct a matrix where each column corresponds to an image and two consecutive rows correspond to the $x$ and $y$ position of a point as measured in each frame. If the camera is assumed to be affine, then without noise this matrix is known to have a rank of no more than three [19]. The best rank-3 matrix can be easily found using the singular value decomposition of the matrix, which leads directly to an estimate of the 3D locations of the points. Unfortunately, points are usually not visible in every frame and so the measurement matrix has missing data which precludes the use of SVD. Finally, there is the well-known Netflix problem [3] where each row of the matrix consists of a user, each column represents a movie, and an entry in the matrix is the rating that a user gives to a movie. The missing entries of this matrix are ratings users will give movies which they have not yet seen and the goal is to predict these missing values in order to provide movie recommendations to the users.

In general, the matrix completion problem is underdetermined since filling in the missing matrix entries with any values whatsoever will, indeed, complete the matrix. Instead, it is necessary to impose additional constraints on our matrix. The basic idea of low-rank matrix completion is that the data in matrices is highly redundant. For example, in the Netflix problem, people who thought *The Terminator* was a good movie and disliked *Sleepless in Seattle* will probably enjoy *Predator* more than *You've Got Mail*. The reason for this is that there are relatively few underlying characteristics that determine what movies a person likes, such as whether it is an action movie or whether it stars Meg Ryan. Of course, other small features might slightly change a person's rating of a movie – perhaps they didn't like the costume that a character wore – but these are minor influences and can be ignored without a significant loss in accuracy. So, in order to determine a person's movie rating, it is necessary determine the major contributing factors to how people rate movies and then determine how each person uses these factors.

## 1.2 Problem description

More formally, let $M$ be a $n_1 \times n_2$ matrix. Let $\Omega$ be the set of indices of $M$ which are given such that $(i,j) \notin \Omega$ are the missing entries to be imputed. Let $\mathcal{P}_\Omega$ be the projection operator which sets all unknown indices to zero:

$$\mathcal{P}_\Omega(M)_{ij} = \begin{cases} M_{ij} & \text{if } (i,j) \in \Omega \\ 0 & \text{if } (i,j) \notin \Omega \ . \end{cases} \tag{1}$$

If there is no ambiguity, we will drop the $\Omega$ and simply use $\mathcal{P}(\cdot)$.

The redundancy of the data in $M$ can be formally described by the *rank* of $M$, which is the number of linearly-independent rows or columns of $M$. Specifically, if $\text{rank}(M) = k$, then we can decompose $M$ as the product of two matrices,

$$M = UR^T \ , \tag{2}$$

where $U$ is $n_1 \times k$ and $R$ is $n_2 \times k$. For the Netflix problem described above, each row of $R$ represents some characteristic of a movie and each row of $U$ is the importance that a user gives to of each of these characteristics. If it is known *a priori* that $M$ is low rank, then a natural way to fill in the missing values is to find the lowest rank matrix $Z$ which agrees with the given data:

$$\begin{aligned} &\text{minimize} \quad \text{rank}(Z) \\ &\text{subject to} \quad \mathcal{P}(Z) = \mathcal{P}(M) \ . \end{aligned} \tag{3}$$

Unfortunately, rank-minimization is NP-hard [8] and there is no efficient way to solve this problem.

## 1.3 Matrix completion using convex optimization

In 2009, a landmark paper was published by Emmanuel Candés and Benjamin Recht [8] showing that Equation (3) was not nearly as difficult to optimize as it seems, and in fact in many cases it can be solved exactly using convex optimization. A follow-up paper by Candés and Tao [9] improved on their results. We review briefly review their results here.

The central idea of [8] and [9] is to replace the rank function in Equation (3) with the *nuclear norm*. The nuclear norm of a matrix is defined as the sum of its singular values:

$$\|Z\|_* = \sum_i \sigma_i(Z). \tag{4}$$

The nuclear norm of a matrix is related to its rank because while the rank measures the number of non-zero singular values, the nuclear norm is their sum. Thus, the nuclear norm is a relaxation of the rank function. The problem then becomes

$$\begin{array}{ll} \text{minimize} & \|Z\|_* \\ \text{subject to} & \mathcal{P}(Z) = \mathcal{P}(M) \ . \end{array} \tag{5}$$

Crucially, this problem is now convex, and the advantage of working with a convex problem is that any local minimum is in fact the global minimum and thus they can be solved exactly and efficiently using any number of local descent algorithms. The problem in Equation 5 is in fact a semidefinite program, which has been well-studied and many software packages exist to solve them efficiently (for example, see [17, 18]). However, there is no guarantee that solving Equation (5) will provide a good solution to the non-convex Equation (3) because the two are not equivalent. The results of [8] and [9], however, show that most of the time, solving this convex relaxation leads to the exact, unique solution of Equation (3). More specifically, it is shown in [8] that this is the case for matrices which adhere to certain conditions. We explore what these conditions are in the following section.

### 1.3.1 Minimal conditions for exact matrix reconstruction

Even if it is known that $M$ is low-rank, it is not always possible to reconstruct the missing entries. In the extreme case, a matrix with no entries given whatsoever clearly has an infinite number of solutions. More generally, we might ask: what is the minimal number of entries that are needed in order to complete $M$ uniquely? First of all, it's clear that there must be at least as many entries as $M$ has degrees of freedom, or else there will be an infinite number of possible rank-$k$ solutions. The degrees of freedom of a matrix $M$ are given in the following proposition.

**Proposition 1.** *A $n_1 \times n_2$ rank-$k$ matrix has $k(n_1 + n_2) - k^2$ degrees of freedom.*

*Proof.* Let the $U\Sigma V^T = M$ be the singular value decomposition of the rank-$k$ matrix $M$. Because $M$ is of rank $k$, then $\Sigma$ can be written as a $k \times k$ generalized diagonal matrix containing the $k$ non-zero singular values of $M$. Similarly, $U$ and $V$ are of size $n_1 \times k$ and $n_2 \times k$. The degrees of freedom of $M$ can be calculated as follows. First, each of the $k$ singular values contributes one degree of freedom. For $U$, the first column has $n_1 - 1$ degrees of freedom since it has $n_1$ values and is normalized to unit norm. Each successive column of $U$ must also be orthogonal to the ones before it, and so $U$ has

$$(n_1 - 1) + (n_1 - 2) + \cdots + (n_1 - k) = n_1 k - k(k+1)/2$$

degrees of freedom. Similarly, $V$ has $n_2 k - k(k+1)/2$ degrees of freedom, and so $M$ has a total of $k + n_1 k + n_2 k - k(k+1) = k(n_1 + n_2) - k^2$ degrees of freedom. □

We therefore must have, at the very minimum, at least $k(n_1 + n_2) - k^2$ entries in order to complete $M$. By assuming that $k \ll n$, then if $n = \max\{n_1, n_2\}$, this is about $O(kn)$.

It is also necessary to have at least one value given in each column and row of $M$. Consider the case where an entire column of $M$ is missing such that for some $j$ there is no $i$ with $(i, j) \in \Omega$. Let $U\Sigma V^T = M$ again be the singular value decomposition of $M$. The $j^{th}$ column of $M$ is given by

$$M_{:j} = US(V_{j:})^T. \tag{6}$$

Because no entries are given for $M_{:j}$, it's clear that $V_{j:}$ can be replaced with any values whatsoever, and thus there is no unique completion for $M$. The same argument holds for the rows of $M$, and so there must be at least one entry in each row and column of $M$ to have any hope of finding an exact completion. This is related to the *coupon collectors problem* [11], which says that if one of $n$ bins is sampled uniformly at random, $O(n \log n)$ samples are needed to ensure that each bin is selected at least once. So, rather than just needing just $O(kn)$ samples, it may be necessary to have close to $O(kn \log n)$ samples to ensure that there are more samples than degrees of freedom and there is an entry in each row and column if the entries are assumed to be sampled uniformly at random.

Finally, it is necessary to impose constraints on the singular vectors of the matrix $M$. Consider a rank-1 matrix $uv^T$, where $u$ is non-zero only at entry $i$ and zero elsewhere, and similarly $v$ is only non-zero at entry $j$. The matrix $M$ will then be entirely zero except for a single non-zero entry $M_{ij}$, and it's clear that unless this exact entry is given there is no way to reconstruct $M_{ij}$ correctly since all other entries are zero and provide no information about its value. The problem here is that the singular vectors $u$ and $v$ are not "spread out" enough, and so the vast majority of entries of $M$ give no information at all about $u$ and $v$. To deal with this, [9] introduces a notion of *coherence*, which measures how spread out the singular values of $M$ are. The primary result of [9] is the following theorem:

**Theorem 1.** *Let $M \in \mathbb{R}^{n_1 \times n_2}$ be a fixed matrix of constant rank $k = O(1)$ which obeys the incoherence assumptions given in [9] with parameter $\mu$. Let $n = \max\{n_1, n_2\}$. If we observe $m$ entries of $M$, then there exists a positive constant $C$ such that if*

$$m \geq C\mu^2 n \log^2 n \ , \tag{7}$$

*then $M$ is the unique solution to Equation (5) with probability at least $1-n^{-3}$.*

For a proof of this theorem and details on the idea of coherence, see [9].

We have already seen that it is necessary to have at least $O(kn \log n)$ samples – or $O(n \log n)$ if $k = O(1)$ as in this theorem – and this theorem shows that with only an additional logarithmic factor on the number of samples the matrix $M$ can very likely be recovered exactly using convex optimization. The authors then go on to show that the assumptions they propose are held by the vast majority of matrices under several different matrix models. In other words, for most matrices it is possible to reconstruct the matrix exactly from a very small number of entries. This is a remarkable result: not only do the solutions to original problem in Equation (3) and its convex relaxation in Equation (5) coincide for *some* matrices, they are the same for *the vast majority* of matrices. This allows us to solve a convex problem in practice, and very often obtain the solution to a more difficult, non-convex problem.

Since the initial publication of these results, a large amount of research has been devoted to developing improved algorithms for matrix completion. Although semidefinite programming can be used to find a solution to Equation (5), this is inefficient for large matrices and would be impossible to use on matrices as large as the Netflix dataset. This has encouraged efforts to find more efficient algorithms that perform well in practice. In this report we review four results on this topic. First, we describe the Singular Value Thesholding (`SVT`) [6] algorithm which does not require the rank to be specified and iteratively optimizes an approximation of the nuclear-norm objective function. Next, we present the Low-Rank Matrix Fitting (`LMaFit`) [21] algorithm which fixes the rank by explicitly writing the matrix in terms of its low-rank factors and uses an optimization technique based on successive over-relaxation to minimize the error. We then present an algorithm – `ScGrad` [15] – that re-interprets `LMaFit` as optimization on the Grassmann manifold and then improves convergence by changing the metric on the manifold and using conjugate gradients rather than standard gradient descent. Finally, we also describe `Grouse` [1], an algorithm for online matrix completion. We show that `Grouse` is closely related to the incremental SVD algorithm (`ISVD`) [4] and propose a modification that allows `ISVD` to be used efficiently for matrix completion. In addition to showing how these different algorithms are related to each other, we also compare them experimentally using synthetic data.

## 2 Singular Value Thresholding

We begin with the Singular Value Thresholding (`SVT`) algorithm [6], which optimizes an approximation of Equation (5) by adding a Frobenius-norm term to the objective function,

$$\begin{aligned} &\text{minimize} \quad \tau\|Z\|_* + \frac{1}{2}\|Z\|_F^2 \\ &\text{subject to} \quad \mathcal{P}_\Omega(Z) = \mathcal{P}_\Omega(M)\ , \end{aligned} \tag{8}$$

where the parameter $\tau$ trades off between the nuclear and Frobenius norms. Notice that as $\tau \to \infty$, Equation (8) converges to Equation (5). In order to optimize this problem, consider its Lagrangian

$$\mathcal{L}(Z,\Lambda) = \tau\|Z\|_* + \frac{1}{2}\|Z\|_F^2 - \mathrm{tr}(\Lambda^T\,[\mathcal{P}_\Omega(Z) - \mathcal{P}_\Omega(M)]), \tag{9}$$

where $\Lambda$ is introduced as a dual variable. Strong duality trivially holds from Slater's condition since there are no inequality constraints and we consider optimization over both $Z$ and $\Lambda$ in turn. This Lagrangian can be re-written as

$$\mathcal{L}(Z, \Lambda) = \tau \|Z\|_* + \frac{1}{2}\|Z\|_F^2 - \mathrm{tr}(\Lambda^T \mathcal{P}_\Omega(Z)) + \mathrm{tr}(\Lambda^T \mathcal{P}_\Omega(M)) \ . \tag{10}$$

First, consider $\Lambda$ to be fixed, in which case this is an unconstrained minimization problem over the variable $Z$. Because we are optimizing with respect to $Z$, we can remove the term $\mathrm{tr}(\Lambda^T \mathcal{P}_\Omega(M))$ and introduce a term $\frac{1}{2}\mathrm{tr}(\mathcal{P}_\Omega(\Lambda)^T \mathcal{P}_\Omega(\Lambda))$ without changing the location of the minimum, giving

$$\arg\min_X \mathcal{L}(Z, \Lambda) = \arg\min_Z \tau \|Z\|_* + \frac{1}{2}\mathrm{tr}(Z^T Z) - \mathrm{tr}(\Lambda^T \mathcal{P}_\Omega(Z)) + \frac{1}{2}\mathrm{tr}(\mathcal{P}_\Omega(\Lambda)^T \mathcal{P}_\Omega(\Lambda)) \tag{11}$$

$$= \arg\min_Z \tau \|Z\|_* + \frac{1}{2}\sum_{i,j} \left(Z_{ij}^2 - 2\Lambda_{ij} Z_{ij} \mathbb{I}[(i,j) \in \Omega] + \Lambda_{ij}^2 \mathbb{I}[(i,j) \in \Omega]\right) \tag{12}$$

$$= \arg\min_Z \tau \|Z\|_* + \frac{1}{2}\sum_{i,j} \left((Z_{ij} - \Lambda_{ij} \mathbb{I}[(i,j) \in \Omega])^2\right) \tag{13}$$

$$= \arg\min_Z \tau \|Z\|_* + \frac{1}{2}\|Z - \mathcal{P}_\Omega(\Lambda)\|_F^2 \ . \tag{14}$$

Fortunately, the solution to this optimization problem is straightforward to compute. First we define the *singular value shrinkage operator*:

**Definition 1.** *Let $U\Sigma V^T = Z$ be a singular value decomposition of the matrix $Z \in \mathbb{R}^{n_1 \times n_2}$. The* singular value shrinkage operator *is defined as*

$$\mathcal{D}_\tau(Z) = U \max\{S - \tau, 0\} V^T. \tag{15}$$

This operator shrinks all singular values by $\tau$ without letting them become negative. We now present the following theorem from [6]:

**Theorem 2.** *Let $\tau \geq 0$ be a real number and let $Y \in \mathbb{R}^{n_1 \times n_2}$ be a real-valued matrix. Then the solution of the problem*

$$\arg\min_Z \tau \|Z\|_* + \frac{1}{2}\|Z - Y\|_F^2 \tag{16}$$

*is achieved by the matrix $Z = \mathcal{D}_\tau(Y)$.*

*Proof.* First, observe that (16) is a convex function: norms are convex since by the definition of a norm we have $\rho(\alpha Z + (1-\alpha)Y) \leq \alpha\rho(Z) + (1-\alpha)\rho(Y)$ for the norm $\rho(\cdot)$ with $\alpha \in [0,1]$. Convexity is also invariant under summation and affine maps and so (16) is convex. Therefore any local minimum is also a global minimum. We now show that zero is in the subgradient of Equation (16) at the point $Z = \mathcal{D}_\tau(Y)$, in which case we have proven that $\mathcal{D}_\tau(Y)$ is a global minimum.

Let $U\Sigma V$ be a singular value decomposition of $Z$. Taking the subgradient of the functional gives

$$Z - Y + \tau\partial\|Z\|_*, \tag{17}$$

where $\partial\|Z\|_*$ is the subgradient of the nuclear norm function. In order to show that 0 is in this set, is suffices to show that $Y - Z$ is in the subgradient of $\|Z\|_*$. It is known that $Z \in \partial\|Z\|_*$ if $Z$ can be decomposed into $Z = UV^T + W$, where $W$ obeys the properties $U^T W = 0$, $WV = 0$ and $\|W\|_2 \leq 1$ (we refer the reader to [14, 20] for details, which are beyond the scope of this report). Let us now decompose $Y$ into the sum of two SVD's to separate the singular values greater than and less than $\tau$. We have

$$Y = U_0\Sigma_0 V_0^T + U_1\Sigma_1 V_1^T \ , \tag{18}$$

where the decomposition $U_0\Sigma_0V_0^T$ corresponds to singular values $\leq \tau$ and $U_1\Sigma_1V_1^T$ corresponds to those $> \tau$. Our proposed solution $Z$ is given by $Z = \mathcal{D}_\tau(Y) = U_1(\Sigma_1 - \tau I)V_1^T$ since the smaller singular values drop out. We then have

$$\begin{aligned} Y - Z &= U_0\Sigma_0V_0^T + U_1\Sigma_1V_1^T - U_1(\Sigma_1 - \tau I)V_1^T && (19)\\ &= U_0\Sigma_0V_0^T + \tau U_1V_1^T && (20)\\ &= \tau\left(U_1V_1^T + \frac{1}{\tau}U_0\Sigma_0V_0^T\right) \ . && (21) \end{aligned}$$

We can then write $Y - Z = \tau\left(U_1V_1^T + W\right)$ where $W = \frac{1}{\tau}U_0\Sigma_0V_0^T$, which is now in the desired form. Observe that $U_1^TW = U_1^T\frac{1}{\tau}U_0\Sigma_0V_0^T = 0$ since $U_0$ and $U_1$ composed of singular vectors from the SVD of $Y$ and are thus orthogonal. Similarly, $WV = 0$. Finally, we use the definition $\|W\|_2 = \sigma_{\max}(W)$. Since $\Sigma_0$ was chosen by construction to have a singular values of at most $\tau$, then the $\frac{1}{\tau}$ factor means that $\|W\|_2 \ \leq 1$. Therefore $Y - Z$ is in the subgradient of $\|Z\|_*$, which completes the proof. □

Using this theorem, it's clear that the Lagrangian in Equation (8) can be minimized with respect to $Z$ by setting $Z = \mathcal{D}_\tau(\mathcal{P}_\Omega(\Lambda))$. Additionally, because the entries in $\Lambda$ are the dual variables for the constraints $\mathcal{P}_\Omega(Z) = \mathcal{P}_\Omega(M)$, they only need to be considered on the set $\Omega$, which simplifies the update to be $Z = \mathcal{D}_\tau(\Lambda)$.

Now consider the derivative of the Lagrangian with respect to $\Lambda$,

$$\frac{\partial}{\partial \Lambda}\mathcal{L}(Z, \Lambda) = \mathcal{P}_\Omega(M - Z) \ . \tag{22}$$

The `SVT` algorithm uses the current value of $Z$ and takes a step in the direction of this gradient (since we are maximizing with respect to $\Lambda$) using the update

$$\Lambda = \Lambda + \delta \mathcal{P}_\Omega(M - Z) \ . \tag{23}$$

The Singular Value Thresholding algorithm then consists of the following two iterative steps:

$$\begin{cases} Z = \mathcal{D}_\tau(\Lambda) \\ \Lambda = \Lambda + \delta \mathcal{P}_\Omega(M - Z) \ . \end{cases} \tag{24}$$

The final algorithm is show in Algorithm 1. In all the algorithms presented in this paper, the notation $(\cdot)'$ will be used to denote the updated value of a variable at each iteration.

**Algorithm 1** Singular Value Thresholding [6]

```
1: procedure SVT($M, \tau, \delta, k_{\max}$)
2:     Initialize $Y = 0$
3:     for $k \leftarrow 1, \dots, k_{\max}$ do
4:         Shrink singular values to update $Z$
5:         $U \Sigma V^T = SVD(Y)$
6:         $\hat{\Sigma}_{ii} = \max\{\Sigma_{ii} - \tau, 0\}$
7:         $Z' = U \hat{\Sigma} V^T$
8:         Update $Y$
9:         $Y' = Y + \delta \mathcal{P}_\Omega(M - Z')$
10:    end for
11:    return $Z$
12: end procedure
```

Note that this algorithm assumes that $M$ has no noise. However, the authors of [6] observe that `SVT` still performs well when there is noise, although they suggest stopping the algorithm early in order to ensure that the matrix has a low rank.

## 3 Low-Rank Matrix Fitting

Rather than just assuming that the matrix $M$ has low rank, in many cases the exact rank of $M$ is known. Even if the exact rank is not known, several ranks can be searched over to find the best fit to the data. Computationally, the advantage of knowing the exact rank of $M$ *a priori* is that $M$ can be written explicitly as the product of two factors

$$M = UR^T, \tag{25}$$

where $U$ is of size $n_1 \times k$ and $R$ is $n_2 \times k$, for $M$ of rank $k$. We can then define an optimization problem directly on the two factors, $U$ and $R$. One natural problem is the following

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\|UR^T - Z\|_F^2 \\ \text{subject to} & \mathcal{P}(Z) = \mathcal{P}(M)\ . \end{array} \tag{26}$$

Because using a decomposition of the matrix into its factors is equivalent to strictly enforcing a rank constraint, this problem is no longer convex as `SVT` was. However, it can be shown that an initialization given by a “trimmed” SVD of $\mathcal{P}_\Omega(M)$ is, under some assumptions, close enough to the global minimum that local descent approaches are guaranteed to find it [13]. It has also recently been shown that alternately optimizing over $U$ and $R$ will find the global solution under assumptions similar to those for the nuclear norm minimization problem [12]. Because of this, even though the problem is no longer convex this is a reasonable cost function to use.

There are now three variable matrices: $U$, $R$ and $Z$. This problem can be thought of as searching over the set of rank-$k$ matrices to find a point $UR^T$ which is closest to the set of matrices which agree with $M$ at all given points, in terms of the Frobenius norm. The point it’s closest to in this case is the matrix $Z$. This is shown graphically in Figure 1. Note that if $M$ has an exact rank-$k$ reconstruction, the objective function achieves a cost of 0 by setting both $Z$ and $UR^T$ to this reconstruction (and graphically, the two sets in Figure 1 will overlap).

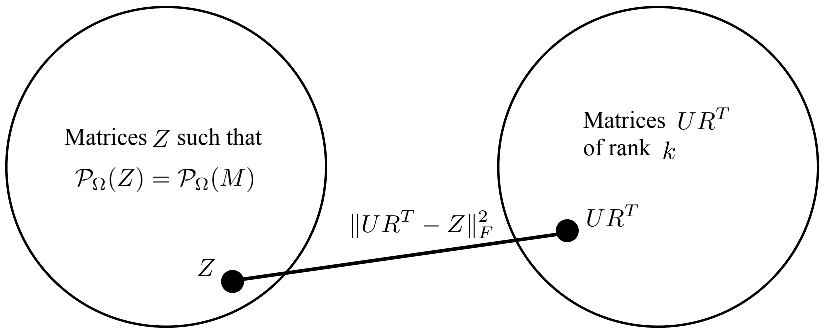

Matrices $Z$ such that
$\mathcal{P}_\Omega(Z) = \mathcal{P}_\Omega(M)$
Matrices $UR^T$ of rank $k$
$\|UR^T - Z\|_F^2$
$Z$
$UR^T$

Figure 1: Illustration of the objective function from Equation (26). Two matrices are searched over: $Z$, which must agree with $M$ on the observed entries, and $UR^T$, which must be of rank $k$. The optimal pair of matrices minimizes the distance between them.

## 3.1 Solving for each matrix separately

The most straightforward way to optimize Equation (26) is to fix two of the three matrices and solve for each one in turn. First, fixing $R$ and $Z$ we can take the derivative of the objective with respect to $U$ and set it equal to zero, giving the normal equation

$$UR^TR = ZR \;, \tag{27}$$

which has the solution

$$U = ZR(R^TR)^{-1} = Z(R^+)^T \;, \tag{28}$$

where $R^TR$ is invertible as long as $R$ has linearly-independent columns.

Similarly, when $U$ and $Z$ are fixed, the optimal $R$ is given by $R = Z^T(U^+)^T$. Finally, we fix $U$ and $R$. Because we require that $\mathcal{P}(Z) = \mathcal{P}(M)$, the optimal $Z$ is found by enforcing this constraint and letting all other entries be exactly those of $UR^T$ so that the error is minimized:

$$Z_{ij} = \begin{cases} M_{ij} & \text{if } (i,j) \in \Omega \\ (UR^T)_{ij} & \text{if } (i,j) \notin \Omega \;. \end{cases} \tag{29}$$

If we define the residual matrix as $E = \mathcal{P}_\Omega(M - UR^T)$ then this can be written as

$$Z = UR^T + E \;. \tag{30}$$

Before proceeding, we show how this algorithm can be generalized by noting its relation to fixed-point iteration methods.

## 3.2 Optimization using non-linear fixed-point iteration

As a different approach to optimizing Equation 26, consider its Lagrangian

$$\mathcal{L}(U, R, Z, \Lambda) = \frac{1}{2}\|UR^T - Z\|_F^2 - \mathrm{tr}(\Lambda^T \mathcal{P}_\Omega(Z - M)), \tag{31}$$

where $tr(X^T Y) = \sum_{i,j} X_{ij} Y_{ij}$ is the matrix inner product. Note here that $\Lambda$ is defined such that $\Lambda = \mathcal{P}_\Omega(\Lambda)$ so that there are only dual variables defined for the entries in $\Omega$.

Now we can differentiate the Lagrangian. In the following, we make use of the fact that $\|X\|_F^2 = tr(X^T X)$. Also, because the projection operator is linear we have $\mathcal{P}(A + B) = \mathcal{P}(A) + \mathcal{P}(B)$. The Lagrangian can be written as

$$\mathcal{L}(U, R, Z, \Lambda) = \frac{1}{2}\mathrm{tr}((RU^T - Z^T)(UR^T - Z)) - \mathrm{tr}(\Lambda^T \mathcal{P}_\Omega(Z - M)) \ . \tag{32}$$

Differentiating the Lagrangian gives

$$\frac{\partial \mathcal{L}}{\partial U} = (UR^T - Z)R \ , \tag{33}$$

$$\frac{\partial \mathcal{L}}{\partial R} = U^T(UR^T - Z) \ , \tag{34}$$

$$\frac{\partial \mathcal{L}}{\partial \Lambda} = -\mathcal{P}_\Omega(Z - M) \ . \tag{35}$$

and

$$\frac{\partial \mathcal{L}}{\partial Z} = (Z - UR^T) - \Lambda \ , \tag{36}$$

Setting each equal to zero gives the conditions

$$\begin{cases} (UR^T - Z)R = 0 \\ U^T(UR^T - Z) = 0 \\ \mathcal{P}_\Omega(Z - M) = 0 \\ \mathcal{P}_{\Omega^C}(Z - UR^T) = 0, \end{cases} \tag{37}$$

where the last equation was derived by noting that $\Lambda_{ij} = 0$ for $(i, j) \notin \Omega$.

The system in Equation (37) is a set of non-linear homogeneous equations. We will attempt to solve these equations using fixed-point iteration. Because this technique is commonly used to solve linear systems of equations, we first briefly review how this works.

### 3.2.1 Fixed-point iteration for linear systems

Suppose we are given the linear system of equations $Ax = b$. Let $A = B + C$ be a decomposition of the matrix $A$ into the sum of two matrices. Clearly we sill have $(B + C)x = b$, which can be rewritten as $Bx = b - Cx$. If $B$ is invertible, then we can write

$$x = B^{-1}(b - Cx) \ . \tag{38}$$

This shows that $x$ is a *fixed point* of the function $f(x) = B^{-1}(b - Cx)$ since $f(x) = x$. One method to try to find such a fixed-point is to initialize $x^0$ and repeatedly let $x^k = f(x^{k-1})$. At each iteration a solution is found to the system of equations

$$Bx^k = b - Cx^{k-1}. \tag{39}$$

There are, of course, many ways that an arbitrary matrix $A$ can be decomposed into $B + C$. One method is to let $A = L + D + U$ where $L$ is the lower-triangular portion of $A$, $D$ is the diagonal and $U$ is the upper triangular portion. The fixed-point equation is then

$$(L + D)x^k = b - Ux^{k-1} \ , \tag{40}$$

which is known as the *Gauss-Seidel* method. Similarly, the *Jacobi* method decomposes $A$ into $A = D + R$ where $D$ is the diagonal of $A$ and $R = A - D$ is the remainder.

Another method, known as *successive over-relaxation*, is an extension of the Gauss-Seidel method, where the update is done using a weighted combination of the previous iterate and the Gauss-Seidel update:

$$x^k = \omega x^k_{GS} + (1 - \omega)x^{k-1} \ . \tag{41}$$

Multiplying both sides by $D$ gives

$$Dx^k = \omega D x^k_{GS} + (1 - \omega)Dx^{k-1} \ . \tag{42}$$

For the proper choice of $\omega$, this method can lead to an accelerated convergence over Gauss-Seidel. Also, because $L$ is triangular both here and in Gauss-Seidel, fixed-point iteration can be solved using forward substitution. This allows each variable to be updated and then used immediately after in the updates of successive variables.

We now return to solving the non-linear system given in Equation (37). Rearranging the first equation gives $UR^TR = ZR$ which can be rewritten as

$$U = ZR(R^TR)^+ = Z(R^+)^T \; . \tag{43}$$

Note that this is the same update rule for $U$ that was obtained by fixing $R$ and $Z$ and solving for $U$. Similarly, the second condition in Equation (37) yields the same update for $R$ as we encountered before, and the last two conditions give the same update for $Z$. In this way, we are sequentially solving for each variable and then using its new value to solve for subsequent values. In this way, the optimization can be though of as a non-linear Gauss-Seidel method. This idea can be extended to use a successive over-relaxation method for the non-linear system of equations. In this case, the matrices $U$ and $R$ are updated using a weighted combination of the the current values and the next iterate, which leads to Algorithm 2. With a choice of $\omega = 1$, this reduces to reduces to the alternating optimization that was derived first.

**Algorithm 2** Low-rank Matrix Fitting - Successive Over-Relaxation. [21]

```
 1: procedure LMaFit($M, \Omega, \omega$)
 2:     Initialize $U, R, Z$
 3:     for $k \leftarrow 1, \ldots, k_{\max}$ do
 4:         $E' = \mathcal{P}_\Omega(M - UR^T)$
 5:         $Z' = UR^T + \omega E'$
 6:         $U' = \omega Z'(R^+)^T + (1-\omega)U$
 7:         $R' = \omega Z'^T(U'^+)^T + (1-\omega)R$
 8:     end for
 9:     return $U, R$
10: end procedure
```

## 3.3 Choosing $\omega$

The step size $\omega$ has a significant effect on the success of `LMaFit`: if it is too small then convergence will be slow and if it is too large then the algorithm may not converge at all. The authors of [21] use the following heuristic to dynamically select a value for $\omega$ at each iteration. Let $E$ denote the residual matrices from the previous iteration and let $E'(\omega)$ be the error for the current iteration using a step size of $\omega$. The ratio of errors is given by

$$\gamma(\omega) = \frac{\|E'(\omega)\|_F}{\|E\|_F} \ . \tag{44}$$

If $\gamma(\omega) > 1$, indicating that the error has not decreased, then $\omega$ is set to 1 which corresponds to a Gauss-Seidel iteration. Otherwise, the error has decreased and the current step is accepted. Additionally, if $\gamma(\omega)$ is greater than a parameter $\gamma_1$, then the error ratio is not as small as is desired and so $\omega$ is increased with the hope that the error will be decreased faster at the next iteration. In our experiments (Section 6.1), we used code downloaded from the authors' website.

## 3.4 Relationship to SVT

Recall that `LMaFit` with $\omega = 1$ is defined by the following iteration:

$$\begin{cases} Z = UR^T + \mathcal{P}_\Omega(M - UR^T) \\ U = Z(R^+)^T \\ R = Z^T(U^+)^T \ . \end{cases} \tag{45}$$

A similar algorithm could be made by combining the $U$ and $R$ matrices into $X = UR^T$, although it is then no longer possible to use the new version of $U$ to subsequently update $R$:

$$\begin{cases} Z = X + \mathcal{P}_\Omega(M - X) \\ X = ZX^+Z \end{cases} \tag{46}$$

Now recall that the SVT algorithm with $\delta = 1$ is defined by the iteration

$$\begin{cases} Z = Z + \mathcal{P}_\Omega(M - X) \\ X = \mathcal{D}_\tau(Z) \ . \end{cases} \tag{47}$$

These two algorithms are very similar. Although it is hard to say what effect the small differences between these two algorithms has on their optimizations, it is interesting to see how two methods which were derived from very different perspectives – `SVT` constraints the values to agree and minimizes a combination of the nuclear and Frobenius norm while `LMaFit` fixes the rank and minimizes the reconstruction error – arrived at similar algorithms.

## 3.5 An alternative approach

In order to easily relate this algorithm to another approach in the next section, we will first rewrite the algorithm. Recall the set of updates given in Algorithm 2:

$$\begin{cases} E' = \mathcal{P}_\Omega(M - UR^T) \\ Z' = UR^T + \omega E' \\ U' = \omega Z'(R^+)^T + (1-\omega)U \\ R' = \omega Z'^T(U'^+)^T + (1-\omega)R \ . \end{cases} \tag{48}$$

This can be rearranged so that the updates for $U$ and $R$ look more like a "gradient" step in some direction:

$$\begin{cases} E' = \mathcal{P}_\Omega(M - UR^T) \\ Z' = UR^T + \omega E' \\ U' = U + \omega[Z'(R^+)^T - U] \\ R' = R + \omega[Z'^T(U'^+)^T - R] \ . \end{cases} \tag{49}$$

Rather than considering the low-rank completion $UR^T$ as two matrices, we will write it as the product of three matrices $USV^T$ such that $U$ and $V$ are orthogonal and $S$ is symmetric. We assume that $S$ is invertible and have the updates

$$\begin{cases} E' = \mathcal{P}_\Omega(M - USV^T) \\ Z' = USV^T + \omega E' \\ U' = U + \omega[Z'VS^{-1} - U] \\ V' = V + \omega[Z'^T US^{-T} - V] \\ S' = U'^T Z' V' \end{cases} \tag{50}$$

The update for $S$ is derived from noting that $Z$ is the current estimate of the completion of $M$ and so we would like our factorization to reflect this, giving $USV^T = Z$ or $S = U^T ZV$. However, because $U$ and $R$ are restricted to be orthogonal, it is now necessary to re-othogonalize them after each update.

This can be rewritten further by pulling out a factor in the updates for

$U$ and $V$ and rewriting the update for $S$ as

$$\begin{cases} E' = \mathcal{P}_\Omega(M - USV^T) \\ Z' = USV^T + \omega E' \\ U' = U + \omega(Z' - USV^T)VS^{-1} \\ V' = V + \omega(Z' - USV^T)^T US^{-T} \\ S' = U'^T[USV^T + \omega E']V' \end{cases} \tag{51}$$

or, after simplifying by using the definition of $E$, as

$$\begin{cases} E' = \mathcal{P}_\Omega(M - USV^T) \\ Z' = USV^T + \omega E' \\ U' = U + \omega E'VS^{-1} \\ V' = V + \omega E'^T US^{-T} \\ S' = U'^T[USV^T + \omega E']V' \ . \end{cases} \tag{52}$$

Expressing the updates in this way will be useful in connection with the next section.

## 4 Optimization on Grassmann Manifolds

Consider again the cost function optimized in the previous section:

$$\begin{aligned} &\text{minimize}_{U,R,Z} \quad \frac{1}{2}\|Z - UR^T\|_F^2 \\ &\text{subject to} \quad \mathcal{P}_\Omega(Z) = \mathcal{P}(M) \ . \end{aligned} \tag{53}$$

It's clear that for any choice of $U$ and $R$, the optimal $Z$ will equal $UR^T$ on $\Omega^C$ to minimize the cost and equal $M$ on $\Omega$ to enforce the equality constraint. Thus, the cost will be zero on $\Omega^C$, and we can write the equivalent problem

$$\text{minimize}_{U,R} \quad \frac{1}{2}\|\mathcal{P}_\Omega(M - UR^T)\|_F^2 \ . \tag{54}$$

Let $R$ be renamed to $V = R$ to illustrate its connection to the singular value decomposition, and then constrain $U$ and $V$ to have orthonormal columns by introducing a scaling matrix $S$, giving the problem

$$\text{minimize}_{U,S,V} \quad \frac{1}{2}\|\mathcal{P}_\Omega(M - USV^T)\|_F^2 \ , \tag{55}$$

where $USV^T$ represents an arbitrary rank-$k$ matrix. By forcing $U$ and $V$ to be orthonormal, $U$ and $V$ can be considered to be points on the *Grassmann manifold* or *Grassmanian*, which can then be optimized over. We review these topics for completeness before proceeding (for more detail, see [10]).

## 4.1 Overview of Grassmann Manifolds

Consider the $n$-dimensional space $\mathbb{R}^n$. Let $\mathcal{G}(n,k)$ be the set of all $k$-dimensional linear subspaces of $\mathbb{R}^n$. This set $\mathcal{G}(n,k)$ is known as the *Grassmann manifold* or *Grassmannian*. For example, if $n = 2$ then the ambient space is a plane and the Grassmannian $\mathcal{G}(2,1)$ is the set of all lines through the origin. This can be represented concretely as follows. In the space $\mathbb{R}^n$, a linear subspace of dimension $k$ can be represented by a matrix $U$ with $k$ orthogonal columns which span this subspace. This matrix $U$ represents a point on the Grassmann manifold $\mathcal{G}(n,k)$. However, this representation is not unique since any matrix with the same column span will represent the same point on the Grassmannian. In other words, the matrix $U$ is independent of the choice of a basis – as long as the basis spans the appropriate subspace – and all such choices yield a representation of the same point on the Grassmannian. More formally, consider the product

$$U' = UO \;, \tag{56}$$

where $O \in \mathcal{O}(k)$ is an orthogonal matrix. Because $\text{span}(U') = \text{span}(U)$, both matrices $U$ and $U'$ represent equivalent points.

The Grassmannian can also be defined as a *quotient space*, which is a set with an equivalence relation. In this case, the space is the set of all $n \times k$ orthogonal matrices and two matrices are considered equivalent if their columns span the same subspace. This is written as

$$\mathcal{G}(n,k) = \mathcal{V}(n,k)/\mathcal{O}(k) \;, \tag{57}$$

where $\mathcal{V}(n,k)$ is the set of $n \times k$ orthogonal matrices and $\mathcal{O}(k)$ is the so-called *orthogonal group* consisting of all $k \times k$ orthogonal matrices. The set $\mathcal{V}(n,k)$ is known as the *Stiefel manifold* and itself can be written as the quotient space

$$\mathcal{V}(n,k) = \mathcal{O}(n)/\mathcal{O}(n-k) \tag{58}$$

of all $n$-dimensional orthogonal matrices whose first $k$ columns coincide, and so the Grassmann manifold can also be written as

$$\mathcal{G}(n,k) = \mathcal{O}(n)/(\mathcal{O}(n-k) \times \mathcal{O}(k)) \;. \tag{59}$$

In order to perform optimization on the Grassmannian, there must be a *metric* defined on the manifold so that distances can be computed. Because the Grassmannin is a quotient space within the orthogonal group, the canonical metric is found by restricting the metric of the orthogonal group to the Grassman manifold. In [10], it is shown that that this metric is given by

$$g(X, Y) = \operatorname{tr}\left(X^T Y\right), \tag{60}$$

for $X, Y$ in the tangent space of the manifold. The authors of [10] then proceed to give an explicit equation for calculating the gradient of a function on the Grassmann manifold in the following way. For matrix function $F(U) : \mathcal{G}(n, k) \to \mathbb{R}$ which maps a point on the Grassmann manifold to a real number, denote its gradient on the Grassmann manifold to be $\triangledown F$. The gradient at point $U$ must satisfy the condition

$$g(\triangledown F, X) = \operatorname{tr}\left\{\left(\frac{\partial F}{\partial U}\right)^T X\right\}, \tag{61}$$

for all $X$ in the tangent space of the manifold at $U$. This says that the inner product between the gradient vector and another vector in the tangent space at that point is equal to the directional derivative of $F$ at $U$ in the direction $X$. Substituting in the form of the metric gives the equation

$$\operatorname{tr}\left(\triangledown F^T X\right) = \operatorname{tr}\left\{\left(\frac{\partial F}{\partial U}\right)^T X\right\}. \tag{62}$$

The gradient vector must itself be in the tangent space, which is given by the set of matrices $U$ satisfying $U^T \triangledown F = 0$. Although setting $\triangledown F = \frac{\partial F}{\partial U}$ would satisfy the equation, it might not be in the tangent space. Instead, we project it onto the tangent space using

$$\triangledown F = (I - UU^T)\frac{\partial F}{\partial U}. \tag{63}$$

This formula is remarkably simple: the gradient – which lies in the tangent space of the manifold – can be computed by differentiating the cost function and projecting it onto the subspace orthogonal to $U$.

## 4.2 Scaled Gradients on Grassmann Manifolds

Consider again Equation (55), reproduced below:

$$\text{minimize}_{U,S,V} \quad \frac{1}{2}\|\mathcal{P}_\Omega(M - USV^T)\|_F^2 \ . \tag{64}$$

Given values for $U$ and $V$, define the cost function

$$F(U,V) = \min_S \frac{1}{2}\|\mathcal{P}_\Omega(M - USV^T)\|_F^2 \ . \tag{65}$$

This cost function depends only on the subspaces spanned by both $U$ and $V$, and we consider each to be a point on the Grassmanian $\mathcal{G}(n,k)$ (we will address the optimization over $S$ later). The derivatives of the cost function are given by

$$\frac{\partial F}{\partial U} = \mathcal{P}_\Omega(USV^T - M)VS^T \tag{66}$$

and

$$\frac{\partial F}{\partial V} = \mathcal{P}_\Omega(USV^T - M)^T US \ . \tag{67}$$

Define $E = \mathcal{P}_\Omega(M - USV^T)$ as before and then these can be written as

$$\frac{\partial F}{\partial U} = -EVS^T \tag{68}$$

and

$$\frac{\partial F}{\partial V} = -E^T US \ . \tag{69}$$

If one were to perform gradient descent directly on $U$ and $V$, the updates would be

$$U' = U + \omega EVS^T \tag{70}$$

and

$$V' = V + \omega E^T US \ . \tag{71}$$

Compare these updates to those for `LMaFit` given in Equation (52); they are very similar! The difference between gradient descent with $F(\cdot)$ and `LMaFit` is that in `LMaFit` the scaling by $S$ is replaced by $S^{-T}$. However, because we are operating on the Grassmann manifold, the gradients of the cost function – which lie in the tangent space of the manifold at the current point – are given by projecting the partial derivatives onto the subspaces orthogonal to each matrix, giving

$$\nabla F_U = (I - UU^T)\frac{\partial F}{\partial U}; \quad \nabla F_V = (I - VV^T)\frac{\partial F}{\partial V} \ . \tag{72}$$

Nonetheless, there is a close connection between these updates and `LMaFit`, with the primary difference begin the scaling by $S$ or $S^{-1}$ at the end of the updates. It is, in fact, possible to alter these updates so that they resemble the updates of `LMaFit`. This is done in [15] by changing the metric on the Grassmann manifold. In particular, consider the metric

$$d_D(X, Y) = \operatorname{tr}\left(DX^TY\right), \tag{73}$$

where $D \in \mathbb{R}^{k\times k}$ is a symmetric positive-definite matrix. If $D = I$, then this reduces to the canonical metric on $\mathcal{G}(n, k)$. To find the gradient directions on the manifold under this metric, we follow [10] as in the previous section to get

$$\nabla F_U = (I - UU^T)\frac{\partial F}{\partial U}D^{-1} \tag{74}$$

and

$$\nabla F_V = (I - VV^T)\frac{\partial F}{\partial V}D^{-1}. \tag{75}$$

Notice that if we let $D = SS^T$ and $D = S^TS$ for $\nabla F_U$ and $\nabla F_V$, respectively, then $\frac{\partial F}{\partial U}D^{-1}$ and $\frac{\partial F}{\partial V}D^{-1}$ become the same as those from Equation (52) from `LMaFit`.

To see how this scaling affects the optimization, consider the second derivative if $F$ with respect to $U$,

$$\frac{\partial^2 F}{\partial U^2} = SS^T \ . \tag{76}$$

Geometrically, this can be thought of as a quadratic “bowl” where the steepness increases much more quickly along the singular vectors of $SS^T$ which have larger singular values. Thus, any errors in these directions will have more of an effect than the errors in directions associated with smaller singular vectors. From a gradient descent point of view this makes sense: we are interested in decreasing the errors as quickly as possible and so the focus should be on the directions which have the greatest variation in the data.

However, this anisotropic scaling can slow convergence when there is a large difference among the singular values since very little attention is paid to the directions with smaller singular values. Alternatively, by using a scaled metric and letting $D = SS^T$, we see that $\frac{\partial^2 F}{\partial U^2} D^{-1} = SS^T(SS^T)^{-1} = I$ and the second derivative is constant in all directions. The goal of this change is to speed convergence for ill-conditioned matrices which have a large spread in their singular values.

By stepping directly in the direction of the negative gradient, each update will likely result in a new $U$ and $R$ that are no longer on the Grassmannian. The authors of [10] do give an explicit expression for geodesics on the manifold, although this requires calculating the SVD of the gradient. Instead, in order to maintain orthogonality of $U$ and $V$, the authors of [15] use a QR factorization at each iteration. Also, in order to update the singular value matrix $S$, it can either be updated similarly to the update in Equation (52), or the optimal $S$ can be computed exactly given $U$ and $V$ as in Equation (65) by solving a least-squares problem. We will use Equation (52) in our experiments.

Note that these updates are essentially the same as those in Equation (52) and thus the algorithm is essentially the same as `LMaFit`. The advantage of this algorithm is that it is defined with respect to the Grassmann manifold where more complex optimization methods such as conjugate gradients are well-defined and can be used to speed convergence, as we show next.

## 4.3 Scaled Conjugate Gradients

The method of conjugate gradients is commonly used to solve linear systems. Rather than stepping in the direction of the negative gradient as in gradient descent, the search directions are taken to be a linear combination of the gradient and the previous search direction. The purpose of this is to choose directions which are *conjugate* (orthogonal with respect to a given matrix) in order to avoid unnecessary searching in directions that have already been searched over and to avoid the oscillations that can occur with gradient descent. When solving a linear system of equations, conjugate gradients will obtain the exact solution in a finite number of steps [16]. For nonlinear problems, conjugate gradients no longer has such a guarantee but can still produce accurate results faster than gradient descent. Details of the conjugate gradient algorithm are beyond the scope of this work but a straightforward introduction can be found in [16].

The implementation of conjugate gradients is somewhat more complex on manifolds than in Euclidean space, however. We have already shown how gradient directions can be efficiently computed on Grassman manifolds, but because conjugate gradients also depends on the previous search direction, this direction must be *transported* to our current location. While this simply involves shifting the vector in Euclidean space, on a manifold a simple shift would not necessarily keep the gradient in the corresponding tangent space. The authors of [15] remedy this by simply projecting it into the tangent space. More concretely, if $G$ is a vector in the tangent space of the manifold at point $U$, then the parallel transport of $G$ the point $U+V$ is given by

$$T_{U+V}(W) = [I - (U+V)(U+V)^T]W \ . \tag{77}$$

We then have two tangent vectors at each point: the gradient of the function and the transport of the previous search direction. Conjugate gradients then takes a linear combination of these two directions to find a new search direction [16]. The authors of [15] use what is known as the *Polak-Ribiére* update: if $D^{k-1}$ and $G^k$ are the search direction at iteration $k-1$ (appropriately transported) and $G^k$ is the gradient, then the new search direction is given by

$$D^k = -G^k + \beta^k D^{k-1} \ . \tag{78}$$

where

$$\beta^k = \max\left\{0, < G^k - G^{k-1}, G^k > / < G^{k-1}, G^{k-1} >\right\} \ . \tag{79}$$

The resulting algorithm is given in Algorithm 3.

## 5 Online algorithms for matrix completion

All algorithms we have thus far considered are *batch* in that they operate on the full data matrix. Another class of algorithms is *online* which consider only one column at a time. These algorithms make it possible to perform matrix completion before all the data is gathered, such as with streaming data, although here we consider their use in batch matrix completion, which we do by successively iterating over all columns of the given matrix in a random order.

To derive these online algorithms, we re-cast the error function as one that operates on a per-column basis. As before, let $M$ be the matrix to be completed with given entries $\Omega$ and let $UR^T$ be the current rank-$k$ estimate of the completion of $M$.

**Algorithm 3** Scaled Conjugate Gradients on the Grassmannian[15]

```
1: procedure ScGrad($M, \Omega, \omega$)
2:     Initialize $U, S, V$
3:     for $k \leftarrow 1, \dots, k_{\max}$ do
4:         Compute error
5:         $E' = \mathcal{P}_\Omega(M - USV)$
6:         Compute gradients
7:         $\nabla F'_U = (I - UU^T)E'VS^{-1}$
8:         $\nabla F'_V = (I - VV^T)E^TUS^{-T}$
9:         Transport previous gradient and search direction
10:        $\nabla F_U = (I - UU^T)F_U$
11:        $\nabla F_V = (I - VV^T)V_U$
12:        $D_U = (I - UU^T)D_U$
13:        $D_V = (I - VV^T)D_V$
14:        Compute new search direction
15:        $\beta = \frac{<\nabla F'_U - \nabla F_U, \nabla F'_U> + <\nabla F'_V - \nabla F_V, \nabla F'_V>}{<\nabla F_U, \nabla F_U> + <\nabla F_V, \nabla F_V>}$
16:        $D'_U = -\nabla F'_U + \beta D_U$
17:        $D'_V = -\nabla F'_V + \beta D_V$
18:        Take a step and orthogonalize
19:        $U' = \mathrm{orth}(U + \omega D'_U)$
20:        $V' = \mathrm{orth}(V + \omega D'_V)$
21:        Update S
22:        $S' = U'^T[USV^T + \omega E']V'$
23:    end for
24:    return $U, S, V$
25: end procedure
```

## 5.1 Stochastic gradient descent on the Grassmannian

Let $v_i$ be column $i$ of the matrix $M$. Define the error for $v_i$ to be

$$F(U;v) = \min_w \|U_{\Omega_i} w - v_{\Omega_i}\|_2^2 \,, \tag{80}$$

where $U_{\Omega_i}$ and $v_{\Omega_i}$ denote the restriction of the matrix $U$ and vector $v$ to only their observed entries. The weight vector $w$ is given by $w = U_{\Omega_i}^{+} v_{\Omega_i}$ and so $U_{\Omega_i} w$ is the best reconstruction of $v_{\Omega_i}$ that is possible using the current estimate $U$ under the $\ell_2$ norm.

Observe that this cost function does not depend on the basis with which $U$ is expressed and so we consider $U$ to be a point on the Grassmannian just as in Section 4. The derivative of $F$ is given by

$$\frac{\partial F}{\partial U} = -2rw^T \tag{81}$$

where $r$ is the residual vector and is equal to $v_{\Omega_i} - U_{\Omega_i} w$ on $\Omega_i$ and 0 elsewhere. The gradient with respect to the manifold is then

$$\nabla F = -2(I - UU^T) rw^T \tag{82}$$

Because only one column is being considered at a time, it would not be useful to find $U$ so that $F(U;v)$ is fully minimized since this will likely introduce error for the columns of $M$ that are not being considering. Instead, we take a single gradient descent step. However, unlike the `ScGrad` algorithm which takes a step directly in the gradient direction and then must retract back onto the manifold to ensure that $U$ remains orthogonal, we will step in the direction of the gradient but travel *along the associated geodesic* on the manifold, which will ensure that $U$ remains within the manifold and no retraction is necessary. Fortunately, there is a simple expression for geodesics on Grassmann manifolds (see [10]). The resulting update for a step size of $\omega$ is given by

$$U' = U + \left[ \frac{\cos(2\omega\|r\|\|w\|) - 1}{\|w\|} Uw + \sin(2\omega\|r\|\|w\|) \frac{r}{\|r\|} \right] \frac{w^T}{\|w\|} \,. \tag{83}$$

Details of this derivation are beyond the scope of this review and can be found in [1]. Although this expression seems complex, it is composed of simple operations and is extremely fast to compute. The resulting algorithm is known as `Grouse` [1]. However, rather than using `Grouse` as written, we will rewrite it using its relationship to the incremental singular value decomposition.

## 5.2 Incremental SVD

Another online algorithms is the incremental SVD (`ISVD`) algorithm [4], which has a close connection with `Grouse`, which we show in the next section. As its name implies, `ISVD` is a method for computing the SVD of a matrix using one column at a time. For the time being, suppose that the matrix $M$ has no missing entries and the goal is to compute the SVD of $M$. Let $USV^T$ be the the SVD of $M$, and then let a new column be appended into $M$:

$$M' = \begin{bmatrix} M & v \end{bmatrix} = \begin{bmatrix} USV^T & v \end{bmatrix} \ . \tag{84}$$

We would like to find $U', S'$ and $V'$ such that $U'S'V'^T = M'$ without computing the full SVD of $M'$. To do this, let $w = U^+ v$ and $r = v - Uw$ be the least-squares weights and associated residual. The matrix $M'$ can be written as

$$M' = \begin{bmatrix} USV^T & v \end{bmatrix} = \begin{bmatrix} U & \frac{r}{\|r\|} \end{bmatrix} \begin{bmatrix} S & w \\ 0 & \|r\| \end{bmatrix} \begin{bmatrix} V & 0 \\ 0 & 1 \end{bmatrix}^T \ , \tag{85}$$

which can be confirmed by multiplying the matrices out. Observe that because $r$ is orthogonal to $U$ the left-side matrix is orthogonal. Additionally, the right-side matrix is also orthogonal, and so the only reason that this is not the SVD already is that the center matrix is not diagonal. However, it is very close to diagonal and its SVD can be computed as $\hat{U}\hat{S}\hat{V}^T = SVD\left(\begin{bmatrix} S & w \\ 0 & \|r\| \end{bmatrix}\right)$. The the updates are given by

$$U' = \begin{bmatrix} U & \frac{r}{\|r\|} \end{bmatrix} \hat{U} \ , \tag{86}$$

$$V' = \begin{bmatrix} V & 0 \\ 0 & 1 \end{bmatrix} \hat{V} \tag{87}$$

and

$$S' = \hat{S}. \tag{88}$$

Note that because we assumed that there were no missing data, this update is exact: after processing every column of $M$, the SVD of $M$ will have been found. To deal with missing data, $w$ and $r$ can be defined with respect to only the data that are available as was done for `Grouse` (Section 5.1). Furthermore, notice that with each iteration the rank of the SVD is increased by one. As a heuristic to keep the solution rank-$k$, the singular vectors corresponding to the smallest singular value of $S$ can be dropped at each iteration.

However, each iteration will still add a new row to $V$ corresponding to the new column. This is problematic when using `ISVD` in a matrix completion framework where it is desirable to visit columns multiple times. The obvious solution – simply remove the old row of $R$ and update it with the new values – does not work because the SVD requires that $R$ be orthogonal. Instead, we can "downdate" the SVD estimate: just as Equation (85) can be used to update an SVD when adding a column, there is an update for removing a column of the SVD (see [5]). Each iteration then consists of downdating a column before the subsequent update.

## 5.3 Relationship between `Grouse` and `ISVD`

We will now rewrite `Grouse` as we will use it for online matrix completion, which relies on a relationship between `Grouse` and `ISVD` that was recently made explicit in [2]. Consider again the `ISVD` algorithm, where we have

$$\begin{bmatrix} USV^T & v \end{bmatrix} = \begin{bmatrix} U & \frac{r}{\|r\|} \end{bmatrix} \begin{bmatrix} S & w \\ 0 & \|r\| \end{bmatrix} \begin{bmatrix} V & 0 \\ 0 & 1 \end{bmatrix}^T . \tag{89}$$

The singular value matrix $S$ can be moved from the center to the right-side matrix. If we let $R = SV^T$, this becomes

$$\begin{bmatrix} UR^T & v \end{bmatrix} = \begin{bmatrix} U & \frac{r}{\|r\|} \end{bmatrix} \begin{bmatrix} I & w \\ 0 & \|r\| \end{bmatrix} \begin{bmatrix} R & 0 \\ 0 & 1 \end{bmatrix}^T . \tag{90}$$

Of course, the right-side matrix may not be orthogonal, but we can still take the SVD of the center matrix as

$$\hat{U}\hat{S}\hat{V}^T = SVD\left(\begin{bmatrix} I & w \\ 0 & \|r\| \end{bmatrix}\right) \tag{91}$$

and then update

$$U' = \begin{bmatrix} U & \frac{r}{\|r\|} \end{bmatrix} \hat{U}; \quad R' = \begin{bmatrix} R & 0 \\ 0 & 1 \end{bmatrix} \hat{R}\hat{S}^T , \tag{92}$$

followed by subsequently dropping the last column of $U'$ and $R'$ to maintain a constant rank. Remarkably, it can be shown that this update to $U$ is the

**Algorithm 4** Grouse

```
1: procedure GROUSE($M, \Omega, \omega$)
2:     Initialize $U, R$
3:     for $k \leftarrow 1, \ldots, k_{\max}$ do
4:         Randomly select a column
5:         $v_i$ = randomly-selected column $i$ of M.
6:         Remove row $i$ of $R$
7:         Calculate weights and residual
8:         $w = U^{+}_{\Omega_i} v_{\Omega_i}$
9:         $r_{\Omega_i} = v_{\Omega_i} - U_{\Omega_i}$
10:        $r_{\Omega_i^C} = 0$
11:        Update
12:        $\hat{U}\hat{S}\hat{V}^T = SVD\left(\begin{bmatrix} I & w \\ 0 & \|r\| \end{bmatrix}\right)$
13:        $U' = \begin{bmatrix} U & \frac{r}{\|r\|} \end{bmatrix} \hat{U}$
14:        $R' = \begin{bmatrix} R & 0 \\ 0 & 1 \end{bmatrix} \hat{V}\hat{S}^T$
15:        Drop last singular values and vectors
16:    end for
17:    return $U, S, V$
18: end procedure
```

`Grouse` algorithm! Specifically, this is `Grouse` for a specific choice of step size. This version of `Grouse` is shown in Algorithm 4.

This connection has several interesting implications. First, it becomes possible to run `Grouse` without explicitly choosing a step size (although one can still have control over the step size by scaling the residual). Additionally, there is a simultaneous update for $R$, which eliminates the need to solve for $R$ at the end of the algorithm. Finally, and most interestingly, notice that the difference between the `Grouse` and `ISVD` updates is that `ISVD` uses an estimate of the singular values while `Grouse` does not. This relationship is somewhat analogous to the difference between `LMaFit` and `ScGrad` when the metric is not scaled by letting $D = I$.

However, there is another difference between `Grouse` and `ISVD`. Because `Grouse` does not require $R$ to be orthogonal, a row of $R$ can simply be removed before being subsequently updated while with `ISVD` a downdate was required in order to maintain orthogonality of $V$. This downdate is problematic: it does not make sense if these algorithms are to interpreted as stochastic gradient descent where we should continue from any given point and not need to "undo" a previous step. More importantly, this `ISVD` framework does not perform well.

## 5.4 `ISVD` variant for matrix completion

In order to adapt `ISVD` to matrix completion, we propose an algorithm which requires no downdate step. Let $UR^T$ with $U, R \in \mathbb{R}^{n\times k}$ the current estimate of the matrix factorization, where $U$ is orthogonal and no restrictions are placed on $R$. $R$ is then decomposed into its *polar decomposition*,

$$R = PS, \tag{93}$$

where $P \in \mathbb{R}^{n\times k}$ is orthogonal and $S \in \mathbb{R}^{k\times k}$ is symmetric positive semi-definite. If the SVD of $R$ is given by $R = \tilde{U}\tilde{S}\tilde{V}^T$ then this decomposition is given explicitly as

$$P = \tilde{U}\tilde{V}^T \ , \tag{94}$$

and

$$S = \tilde{V}\tilde{S}\tilde{V}^T \ . \tag{95}$$

An equivalent definition of $S$ is $S = \sqrt{R^T R}$, where $\sqrt{\cdot}$ is the matrix square root (this is well-defined since $R^T R$ is positive semi-definite). $S$ is therefore proportional to an estimate of the square root of the covariance matrix of the data while $U$ and $P$ are orthonormal bases for the column and row space, respectively. The estimated matrix factorization is now $USP^T$. This decomposition is very similar to a singular value decomposition, although $S$ is not necessarily diagonal. Nonetheless, it can be used in place of the singular value matrix in `ISVD`, giving

$$\begin{bmatrix} UR^T & v \end{bmatrix} = \begin{bmatrix} U & \frac{r}{\|r\|} \end{bmatrix} \begin{bmatrix} S & w \\ 0 & \|r\| \end{bmatrix} \begin{bmatrix} P & 0 \\ 0 & 1 \end{bmatrix}^T . \tag{96}$$

Just as in `ISVD`, let $\hat{U}\hat{S}\hat{V}^T = SVD\left(\begin{bmatrix} S & w \\ 0 & \|r\| \end{bmatrix}\right)$. The updates are then

$$U' = \begin{bmatrix} U & \frac{r}{\|r\|} \end{bmatrix} \hat{U}; \quad R' = \begin{bmatrix} P & 0 \\ 0 & 1 \end{bmatrix} \hat{R}\hat{S}^T . \tag{97}$$

However, we found that using this polar decomposition when the matrix is very sparse leads to a relatively slow convergence because $S$ tends to have large entries compared to the residual for the current point and so each iteration makes relatively little progress. The reason for this is that as less data is visible, the residual will tend to decrease. We found that scaling down the singular value matrix $S$, which gives more weight to the current point, helps speed convergence. In our experiments, we scale $S$ by $\frac{1}{m}$ – where $m$ is the number of columns of $W$ – and so use the factorization

$$P = m\tilde{U}\tilde{V}^T , \tag{98}$$

and

$$S = \frac{1}{m}\tilde{V}\tilde{S}\tilde{V}^T . \tag{99}$$

An advantage of this algorithm – which we call `ISVD2` – over `ISVD` is that no downdate is required because $R$ does not need to be orthogonal. Instead, at each iteration a row can be removed before the update occurs as is done with `Grouse`. There is also an advantage of this algorithm over `ScGrad` because `ScGrad` does not use geodesics on the Grassmannian in which case it is necessary to re-orthogonalize the matrices at each step. In contrast, `Grouse` takes a step directly along the geodesic and so remains on the manifold. It is plausible that this proposed `ISVD2` framework also performs gradient descent along a geodesic using a metric based on the matrix $S$, but this remains to be shown. The resulting algorithm is given in Algorithm 5.

**Algorithm 5** ISVD2

```
1: procedure ISVD2($M, \Omega, \omega$)
2:     Initialize $U, R$
3:     for $k \leftarrow 1, \ldots, k_{\max}$ do
4:         Randomly select a column
5:         $v_i$ = randomly-selected column $i$ of M.
6:         Zero out row $i$ of $R$
7:         $R_{i:} = 0$
8:         Calculate weights and residual
9:         $w = U^{+}_{\Omega_i} v_{\Omega_i}$
10:        $r_{\Omega_i} = v_{\Omega_i} - U_{\Omega_i}$
11:        $r_{\Omega_i^C} = 0$
12:        Compute polar decomposition of $R$
13:        $\tilde{U}\tilde{S}\tilde{V}^T = SVD(R)$
14:        $P = \tilde{U}\tilde{V}^T$
15:        $S = \tilde{V}\tilde{S}\tilde{V}^T$
16:        Update
17:        $\hat{U}\hat{S}\hat{V}^T = SVD\left(\begin{bmatrix} S & w \\ 0 & \|r\| \end{bmatrix}\right)$
18:        $U' = \begin{bmatrix} U & \frac{r}{\|r\|} \end{bmatrix} \hat{U}$
19:        $R' = \begin{bmatrix} P & 0 \\ 0 & 1 \end{bmatrix} \hat{V}\hat{S}^T$
20:        Drop last singular values and vectors
21:    end for
22:    return $U, R$
23: end procedure
```

# 6 Comparison of algorithms

An overview of the different algorithms we have presented is given in Table 1. On paper, there are many similarities, advantages and disadvantages of these algorithms. For example, `SVT` does not require the rank to be specified but requires the SVD of a large (but sparse) matrix to be computed at each iteration and so is relatively slow[1]. It is also interesting that `LMaFit` can be interpreted as gradient descent method on the Grassmannian manifold, which `ScGrad` extends to conjugate gradients. Similarly, `Grouse` performs stochastic gradient descent on the Grassmannian and it is possible that `ISVD2` alters the metric on the manifold just as `ScGrad` does. However, `Grouse` and `ISVD2` have the advantage of not requiring re-orthogonalization.

## 6.1 Experiments

In order to compare the actual performance between the various algorithms, we have performed experiments on synthetic data. We test how each algorithm performs with respect to the amount of missing data, the spread of the singular values, and the amount of noise. For the algorithms `SVT`, `LMaFit` and `ScGrad`, code was available online from the authors and this was used for the experiments. We implemented our own versions of `Grouse` and `ISVD2` and initialized them using an SVD of $\mathcal{P}_\Omega(M)$, where $M$ is the ground-truth matrix. For `SVT`, we set $\tau = \|M\|_F$ in all experiments. All experiments measure the relative RMSE error as $\|\mathcal{P}_\Omega(\hat{M} - M)\|_F / \|\mathcal{P}_\Omega(M)\|_F$, where $\hat{M}$ is the low-rank estimate of $M$.

### 6.1.1 Effect of missing data

First, we test the effect of missing data. We generated a $500 \times 500$ rank-5 matrix as the product of two $500 \times 5$ factors with entries drawn from a standard normal distribution. A variable number of entries were removed uniformly at random. The results for removing $30\%, 60\%$, and $90\%$ of the data are shown in Figure 2 after each algorithm was run for 5 minutes. In general, the amount of missing data had little effect on the performance of the algorithms and all performed well. However, `SVT` was slower than the

[1]Recent work has made use of duality theory to show how `SVT` can be implemented without computing an SVD at each iteration [7], but we do not consider this here.

`SVT`

- Minimizes $\tau\|Z\|_* + \frac{1}{2}\|Z\|_F^2$ subject to $\mathcal{P}_\Omega(Z - M) = 0$
- Requires large (but sparse) SVD at each iteration

`LMaFit`

- Fixes rank and minimizes $\|UR^T - Z\|$ with $\mathcal{P}_\Omega(Z - M) = 0$
- Uses successive over-relaxation

`ScGrad`

- Re-interprets `LMaFit` as optimization on the Grassmannian
- Uses conjugate gradients to speed convergence
- Requires re-orthogonalization at each iteration

`Grouse`

- Gradient descent along geodesics of the Grassmannian
- Online – operates on one column at a time

`ISVD2`

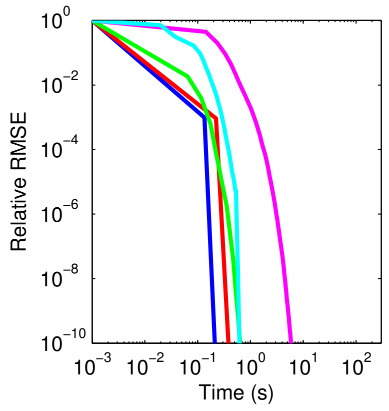
- (Gradient descent along geodesics of the Grassmannian?)
- Online
- Takes into account estimate singular values

Table 1: **Overview comparison of matrix completion algorithms.**

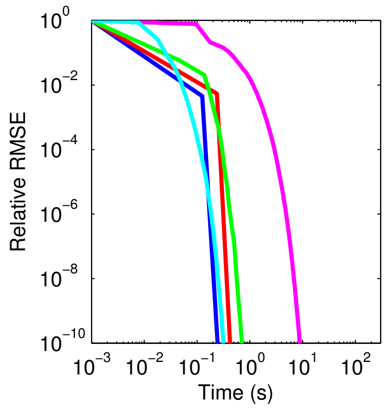
other algorithms to converge and became even more so when more data was missing.

### 6.1.2 Effect of singular values

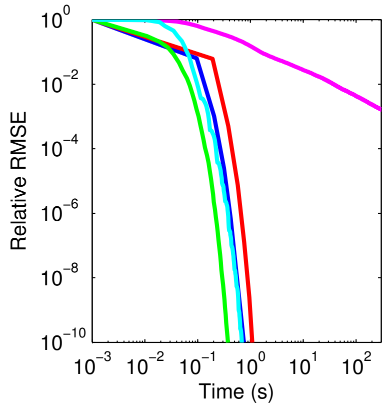
Next, we test how the spread of the singular values affects each algorithm. We generated a $500 \times 500$ rank-5 matrix with random left and right singular vectors with no noise. 90% of the entries were removed uniformly at random. The singular values were selected to increase logarithmically between 100 and $s_{\max}$, where $s_{\max}$ was varied from 1e3 to 1e7. The results are shown in Figure 3.



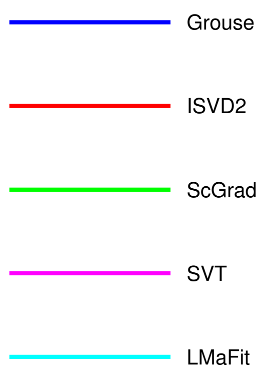
(a) 30% missing data



(b) 60% missing data



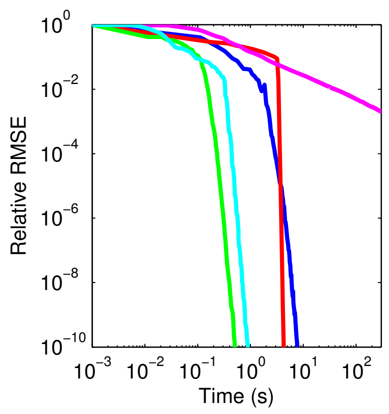
(c) 90% missing data



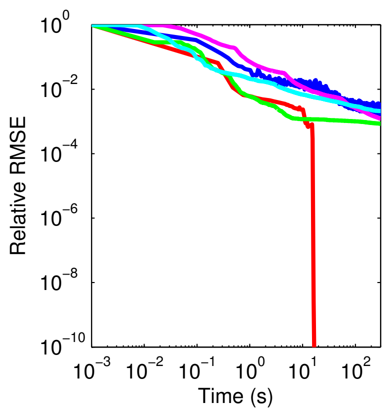
(d)

Figure 2: Effect of missing data. Matrices were generated as the product of two $500 \times 5$ factors with normally-distributed entries. All algorithms perform well, although `SVT` is slow, especially when the proportion of missing data is high. For 30% and 60% missing data, `SVT` correctly estimated the rank to be 5, while for 90% the estimated rank after 5 minutes was 41.

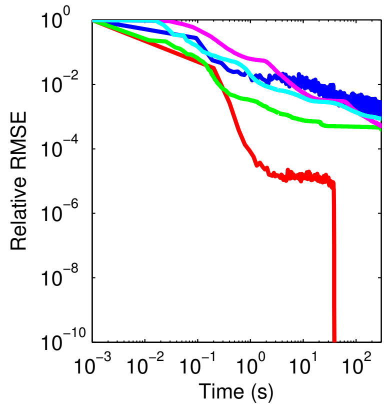
The spread of the singular values had a large effect on the convergence of the algorithms. For a moderate spread in the singular values with $s_{\max} = 1\text{e}3$ – for which the largest singular value was 10 times the smallest – `ScGrad` performed best, although all algorithms converged quickly with the exception of `SVT`. However, for larger spreads in the singular values, only `ISVD2` converged within 5 minutes. This is somewhat surprising since `ScGrad` was specifically designed to handle a large spread in the singular values. This could be because `ScGrad` requires a re-orthogonalization at each iteration.



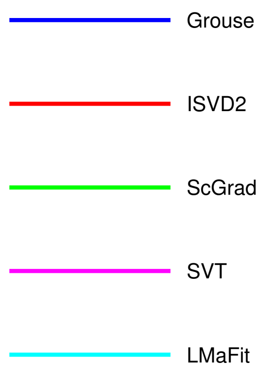
(a) $s_{\max} = 1\text{e}3$



(b) $s_{\max} = 1\text{e}5$



(c) $s_{\max} = 1\text{e}7$



(d)

Figure 3: Effect of singular values. Matrices were generated as the product of two random $500 \times 5$ factors. Singular values were set to vary logarithmically from 1e2 up to $s_{\max}$, where $s_{\max}$ ranged from 1e3 to 1e7. `ScGrad` performs best for moderate spread in the singular values, but only `ISVD2` converges within 5 minutes for a larger spread. `SVT` is slow to converge in all cases and over-estimates the rank of the matrices as 35, 18 and 8, respectively.

### 6.1.3 Effect of noise

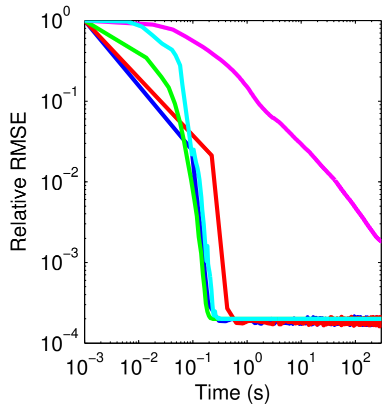
Lastly, we test the effect of noise. We generated a $500 \times 500$ rank-5 matrix as the product of two $500 \times 5$ factors with entries drawn from a standard normal distribution and removed 90% of the entries uniformly at random. Normally-distributed noise was added independently to each observed entry and the standard deviation of the noise was varied to be 1e-3, 1e-2 and 1e-1. Results are shown in Figure 4. All algorithms performed similarly and converged to the same error level, again with the exception of `SVT`. In this case, `SVT` over-estimated the ranks of the matrices as being 45, 50, and 75, respectively.

## 7 Conclusion

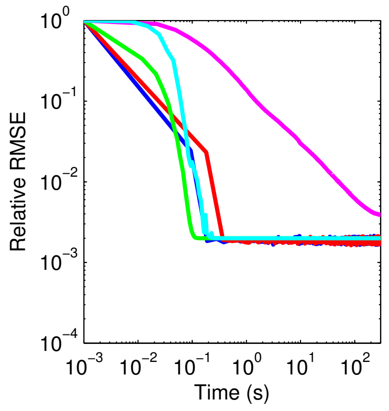
Much research has been done in low-rank matrix completion in the past few years due to its wide applicability. In this report, we reviewed several of these methods. Although these methods differ significantly in how they are derived, they all have a very close connection to one another. For example, we showed that `SVT` and `LMaFit` are both very similar alternating optimizations. `LMaFit` and `ScGrad` can also both be viewed as local descent methods on the Grassmannian. `Grouse` was also derived as a descent method on the Grassmannian, and is very similar to the incremental SVD algorithm.

Most algorithms also performed well in practice. The notable exception was `SVT`, which was slower than the others, was very sensitive to the choice of its parameter $\tau$, and often over-estimated the rank of the matrix – especially when noise was present. Although `SVT` may have performed better with a better choice of $\tau$ for some experiments, this is still a disadvantage since finding the optimal $\tau$ for each experiment is not straightforward. Most algorithms also had trouble when the spread of the singular values was very large, where `ISVD2` performed the best and even outperforming `ScGrad` which was designed to improve convergence on these ill-conditioned matrices.

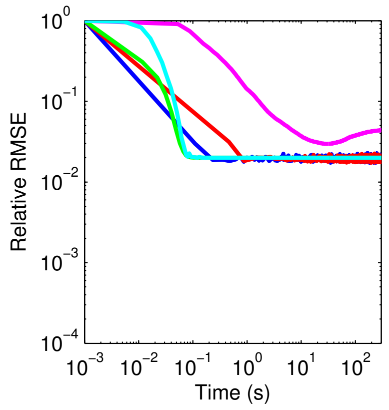
In light of the connections between these different algorithms, future work could look into exactly how these differences affect the optimization problems that are being optimized. More research also needs to be done into `ISVD2` and its relationship to `Grouse`.



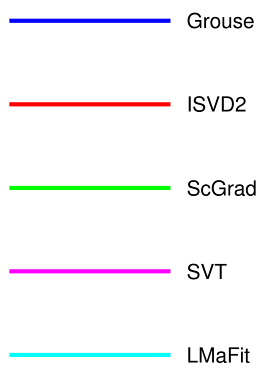
(a) $\sigma$ = 1e-3



(b) $\sigma$ = 1e-2



(c) $\sigma$ = 1e-1



(d)

Figure 4: Effect of noise. Matrices were generated as the product of two $500 \times 5$ factors with normally-distributed entries. Gaussian noise was added independently to each All algorithms performed very similarly with the exception of `SVT` which was slower. `SVT` also over-estimated the ranks of the matrices as being 45, 50, and 75, respectively.

## References

[1] Balzano, L., Nowak, R., and Recht, B. Online identification and tracking of subspaces from highly incomplete information. In *Allerton Conference on Communication, Control, and Computing* (2010), IEEE, pp. 704–711.

[2] Balzano, L., and Wright, S. On the relationship of grouse to the isvd. *Technical Report* (2012).

[3] Bennett, J., and Lanning, S. The netflix prize. In *Proceedings of KDD cup and workshop* (2007), vol. 2007, p. 35.

[4] Brand, M. Incremental singular value decomposition of uncertain data with missing values. In *European Conference on Computer Vision*. Springer, 2002, pp. 707–720.

[5] Brand, M. Fast low-rank modifications of the thin singular value decomposition. *Linear algebra and its applications 415*, 1 (2006), 20–30.

[6] Cai, J.-F., Candès, E. J., and Shen, Z. A singular value thresholding algorithm for matrix completion. *SIAM Journal on Optimization 20*, 4 (2010), 1956–1982.

[7] Cai, J.-F., and Osher, S. Fast singular value thresholding without singular value decomposition. *UCLA CAM Report* (2010), 10–24.

[8] Candès, E. J., and Recht, B. Exact matrix completion via convex optimization. *Foundations of Computational mathematics 9*, 6 (2009), 717–772.

[9] Candès, E. J., and Tao, T. The power of convex relaxation: Near-optimal matrix completion. *Information Theory, IEEE Transactions on 56*, 5 (2010), 2053–2080.

[10] Edelman, A., Arias, T. A., and Smith, S. T. The geometry of algorithms with orthogonality constraints. *SIAM journal on Matrix Analysis and Applications 20*, 2 (1998), 303–353.

[11] Erdos, P., and Rényi, A. On a classical problem of probability theory. *Magyar Tud. Akad. Mat. Kutató Int. Közl 6*, 1-2 (1961), 215–220.

[12] Jain, P., Netrapalli, P., and Sanghavi, S. Low-rank matrix completion using alternating minimization. *arXiv preprint arXiv:1212.0467* (2012).

[13] Keshavan, R. H., Montanari, A., and Oh, S. Matrix completion from a few entries. *Information Theory, IEEE Transactions on 56*, 6 (2010), 2980–2998.

[14] Lewis, A. S. The mathematics of eigenvalue optimization. *Mathematical Programming 97*, 1-2 (2003), 155–176.

[15] Ngo, T., and Saad, Y. Scaled gradients on grassmann manifolds for matrix completion. In *Advances in Neural Information Processing Systems* (2012), pp. 1421–1429.

[16] Shewchuk, J. R. An introduction to the conjugate gradient method without the agonizing pain, 1994.

[17] Sturm, J. F. Using sedumi 1.02, a matlab toolbox for optimization over symmetric cones. *Optimization methods and software 11*, 1-4 (1999), 625–653.

[18] Toh, K.-C., Todd, M. J., and Tütüncü, R. H. Sdpt3a matlab software package for semidefinite programming, version 1.3. *Optimization Methods and Software 11*, 1-4 (1999), 545–581.

[19] Tomasi, C., and Kanade, T. Shape and motion from image streams under orthography: a factorization method. *International Journal of Computer Vision 9*, 2 (1992), 137–154.

[20] Watson, G. Characterization of the subdifferential of some matrix norms. *Linear Algebra and its Applications 170* (1992), 33–45.

[21] Wen, Z., Yin, W., and Zhang, Y. Solving a low-rank factorization model for matrix completion by a nonlinear successive over-relaxation algorithm. *Mathematical Programming Computation 4*, 4 (2012), 333–361.