An accelerated proximal gradient algorithm for nuclear norm regularized linear least squares problems

Kim-Chuan Toh * Sangwoon Yun †

March 27, 2009; Revised, Nov 11, 2009

Abstract

The affine rank minimization problem, which consists of finding a matrix of minimum rank subject to linear equality constraints, has been proposed in many areas of engineering and science. A specific rank minimization problem is the matrix completion problem, in which we wish to recover a (low-rank) data matrix from incomplete samples of its entries. A recent convex relaxation of the rank minimization problem minimizes the nuclear norm instead of the rank of the matrix. Another possible model for the rank minimization problem is the nuclear norm regularized linear least squares problem. This regularized problem is a special case of an unconstrained nonsmooth convex optimization problem, in which the objective function is the sum of a convex smooth function with Lipschitz continuous gradient and a convex function on a set of matrices. In this paper, we propose an accelerated proximal gradient algorithm, which terminates in $O(1/\sqrt{\epsilon})$ iterations with an ϵ-optimal solution, to solve this unconstrained nonsmooth convex optimization problem, and in particular, the nuclear norm regularized linear least squares problem. We report numerical results for solving large-scale randomly generated matrix completion problems. The numerical results suggest that our algorithm is efficient and robust in solving large-scale random matrix completion problems. In particular, we are able to solve random matrix completion problems with matrix dimensions up to 10^5 each in less than 10 minutes on a modest PC.

Key words. Iteration complexity, matrix completion, nuclear norm minimization, proximal gradient, singular value decomposition

AMS subject classification: 90C22, 90C25, 90C51, 65F10

*Department of Mathematics, National University of Singapore, 2 Science Drive 2, Singapore 117543 (mattohkc@nus.edu.sg); and Singapore-MIT Alliance, 4 Engineering Drive 3, Singapore 117576.

†Singapore-MIT Alliance, 4 Engineering Drive 3, Singapore 117576. (smaysw@nus.edu.sg).

1 Introduction

Let $\Re^{m\times n}$ be the space of $m \times n$ matrices endowed with the standard trace inner product $\langle X, Y\rangle = \text{trace}(X^T Y)$. For $X \in \Re^{m\times n}$, the Frobenius norm and spectral norm of X are denoted by $\|X\|_F$ and $\|X\|_2$, respectively. The nuclear norm of X is defined to be $\|X\|_* = \sum_{i=1}^{q} \sigma_i(X)$ where $\sigma_i(X)$'s are the singular values of X and $q = \min\{m, n\}$. In our notation, $\|x\|_p = \left(\sum_{j=1}^{n} |x_j|^p\right)^{1/p}$ for any $x \in \Re^n$ and $1 \le p < \infty$; Diag(x) denotes the diagonal matrix with the vector x on its main diagonal.

The affine rank minimization problem consists of finding a matrix of minimum rank that satisfies a given system of linear equality constraints, namely,

$$\min_{X\in\Re^{m\times n}} \Big\{\text{rank}(X) \,:\, \mathcal{A}(X) = b\Big\}, \tag{1}$$

where $\mathcal{A} : \Re^{m\times n} \to \Re^p$ is a linear map and $b \in \Re^p$. We denote the adjoint of $\mathcal{A}$ by $\mathcal{A}^*$. The problem (1) has appeared in the literature of diverse fields including machine learning [1, 3], control [17, 18, 31], and Euclidean embedding [42]. In general, this affine rank minimization problem (1) is an NP-hard nonconvex optimization problem. A recent convex relaxation of this affine rank minimization problem introduced in [19] minimizes the nuclear norm over the same constraints:

$$\min_{X\in\Re^{m\times n}} \Big\{\|X\|_* \,:\, \mathcal{A}(X) = b\Big\}. \tag{2}$$

The nuclear norm is the best convex approximation of the rank function over the unit ball of matrices. A particular class of (1) is the matrix completion problem; see Section 3. In the matrix completion problem, we are given a random subset of entries of a matrix, and we would like to recover the missing entries such that the resulting matrix has the lowest possible rank.

When the matrix variable is restricted to be diagonal, the problems (1) and (2) reduce to the following linearly constrained nonsmooth minimization problems respectively:

$$\min_{x\in\Re^n} \Big\{\|x\|_0 \,:\, Ax = b\Big\}, \tag{3}$$

where $\|x\|_0$ denotes the number of nonzero components in the vector x, $A \in \Re^{p\times n}$, and

$$\min_{x\in\Re^n} \Big\{\|x\|_1 \,:\, Ax = b\Big\}. \tag{4}$$

The problem (4) has attracted much interest in compressed sensing [8, 9, 10, 14, 15] and is also known as the basis pursuit problem. Recently, Recht et al. [38] established analogous theoretical results in the compressed sensing literature for the pair (1) and (2).

In the basis pursuit problem (4), b is a vector of measurements of the signal x obtained by using the sampling matrix A. If this observation b is contaminated with noise, then $Ax = b$ might not be feasible and so an appropriate norm of the residual $Ax - b$ should be minimized or constrained. In this case, the appropriate models to consider can either be

the following ℓ_1-regularized linear least squares problem (also known as the basis pursuit de-noising problem) [12]:

$$\min_{x\in\Re^n} \ \frac{1}{2}\|Ax-b\|_2^2+\mu\|x\|_1, \tag{5}$$

where μ is a given positive parameter; or the Lasso problem [40]:

$$\min_{x\in\Re^n} \ \Big\{\|Ax-b\|_2^2 \ : \ \|x\|_1\le t\Big\}, \tag{6}$$

where t is a given positive parameter. It is not hard to see that the problem (5) is equivalent to (6) in the sense that a solution of (5) is also that of (6) for some parameters μ, t, and vice versa. Compressed sensing theory shows that a sparse signal of length n can be recovered from $m < n$ measurements by solving any appropriate variant of (5) or (6), provided that the matrix A satisfies certain restricted isometry property. Many algorithms have been proposed to solve (5) and (6), targeting particularly large-scale problems; see [24, 40, 46] and references therein.

Just like the basis pursuit problem, the data in a matrix completion problem may be contaminated with noise, and there may not exist low-rank matrices that satisfy the affine constraints in (2). This motivates us to consider an alternative convex relaxation to the affine rank minimization problem, namely, the following nuclear norm regularized linear least squares problem:

$$\min_{X\in\Re^{m\times n}} \ \frac{1}{2}\|\mathcal{A}(X)-b\|_2^2+\mu\|X\|_*, \tag{7}$$

where $\mu > 0$ is a given parameter. Because (7) is a matrix generalization of (5), it is natural for us to explore the possibility of extending some of the algorithms developed for (5) to solve (7). We note that (7) has an optimal solution since the function $\|\cdot\|_*$ is coercive.

The problem (7) is motivated from our desire to handle matrix completion problems where the data matrices are contaminated with noise. However, its applicability goes beyond matrix completions. For example, the problem (7) arises naturally in simultaneous dimension reduction and coefficient estimation in multivariate linear regression [45]. It also appears in multi-class classification and multi-task learning; see [37] and the references therein.

In this paper, we will develop an accelerated proximal gradient method for a general unconstrained nonsmooth convex minimization problem which includes (7) as a special case. Specifically, the minimization problem we consider has the form:

$$\min_{X\in\Re^{m\times n}} \ \ F(X) := f(X)+P(X), \tag{8}$$

where $P:\Re^{m\times n}\to(-\infty,\infty]$ is a proper, convex, lower semicontinuous (lsc) [39] function and f is convex smooth (i.e., continuously differentiable) on an open subset of $\Re^{m\times n}$ containing $\mathrm{dom}P=\{X \mid P(X)<\infty\}$. We assume that $\mathrm{dom}P$ is closed and ∇f is Lipschitz continuous on $\mathrm{dom}P$, i.e.,

$$\|\nabla f(X)-\nabla f(Y)\|_F\le L_f\|X-Y\|_F \quad \forall X,\, Y\in \mathrm{dom}P, \tag{9}$$

for some positive scalar L_f. The problem (7) is a special case of (8) with $f(X) = \frac{1}{2}\|\mathcal{A}(X) - b\|_2^2$ and $P(X) = \mu\|X\|_*$ with $\mathrm{dom}P = \Re^{m\times n}$.

Recently, Beck and Teboulle [4] proposed a fast iterative shrinkage-thresholding algorithm (abbreviated FISTA) to solve (8) for the vector case where $n = 1$ and $\mathrm{dom}P = \Re^m$, targeting particularly (5) arising in signal/image processing, and reported promising numerical results for wavelet-based image deblurring. This algorithm is in the class of accelerated proximal gradient algorithms that were studied by Nesterov, Nemirovski, and others; see [32, 33, 34, 36, 43] and references therein. These accelerated proximal gradient algorithms have an attractive iteration complexity of $O(1/\sqrt{\epsilon})$ for achieving ϵ-optimality; see Section 2. We extend Beck and Teboulle's algorithm to solve the matrix problem (8), and in particular (7) that arises from large-scale matrix completions. More importantly, we also design practically efficient variants of the algorithm by incorporating linesearch-like, continuation, and truncation techniques to accelerate the convergence.

We should mention that the FISTA algorithm of Beck and Teboulle in [4] has also been extended in [26] to the problem (8) with $P(X) = \mu\|X\|_*$. But as the authors of [26] noted, our algorithms were developed independently of theirs. In addition, the numerical experiments in [26] focused on the problem (7) arising from multi-task learning on relatively small data sets. In contrast, for our paper, the numerical experiments focus on the problem (7) arising from large scale matrix completion problems.

Besides (7), another matrix minimization problem that often occur in practice is the following regularized semidefinite linear least squares problem:

$$\min\Big\{\frac{1}{2}\|\mathcal{A}(X) - b\|_2^2 + \mu\langle I,\, X\rangle \;:\; X \in \mathcal{S}_+^n\Big\}, \tag{10}$$

where $\mathcal{S}_+^n$ is the cone of $n \times n$ symmetric positive semidefinite matrices, and I is the identity matrix. An example of (10) comes from regularized kernel estimation in statistics [28]. The problem (10) is also a special case of (8) with $f(X) = \frac{1}{2}\|\mathcal{A}(X) - b\|_2^2 + \mu\langle I,\, X\rangle$ and

$$P(X) = \begin{cases} 0 & \text{if } X \in \mathcal{S}_+^n; \\ \infty & \text{else.} \end{cases}$$

Note that the term $\langle I,\, X\rangle$ is actually the nuclear norm $\|X\|_*$ when $X \in \mathcal{S}_+^n$, and $\mathrm{dom}P = \mathcal{S}_+^n$. The theoretical results and algorithms that we will develop for (7) can easily be adapted for the symmetric matrix problem (10). Thus in this paper, we will concentrate mainly on the problem (7) and only briefly explain how the results and algorithms for (7) are adapted for (10).

The paper is organized as follows. In section 2, we introduce some gradient algorithms using proximal regularization and accelerated versions which can be applied to solve (8). We also summarize the iteration complexities of the algorithms. In section 3, we introduce the matrix completion problem and review recently developed algorithms for solving this problem. Then we describe our proposed algorithm, which is called the accelerated proximal gradient singular value thresholding algorithm, for solving the nuclear norm regularized linear least squares problem (7) and introduce three techniques to accelerate the convergence

of our algorithm. In section 4, we compare our algorithm with a fixed point continuation algorithm [30] for solving (7) on randomly generated matrix completion problems with moderate dimensions. We also present numerical results for solving a set of large-scale randomly generated matrix completion problems with/without noise. The numerical results show that our algorithm is efficient and robust. In particular, our algorithm is able to solve a random matrix completion problem with $m = n = 10^5$ in less than 10 minutes. We also present numerical results for solving large-scale random semidefinite matrix completion problems of the form given in (10) and matrix completion problems arising from real applications. We further evaluate the performance of our algorithm by applying it to the problem (7) arising from simultaneous coefficient estimation and dimension reduction in multivariate linear regression [45], as well as to the problem (10) arising from regularized kernel estimation [28]. We give our conclusions in section 5.

For readers who are interested in using our proposed algorithm, we have created the NNLS webpage: `http://www.math.nus.edu.sg/~mattohkc/NNLS.html`, which contains MATLAB implementations of the algorithm and additional material including the data used in the numerical experiments.

2 Proximal gradient algorithms

In this section we introduce some proximal gradient algorithms and their accelerated versions which can be applied to solve (8). We also summarize their iteration complexities.

For any $Y \in \mathrm{dom}P$, consider the following quadratic approximation of $F(\cdot)$ at Y:

$$\begin{aligned} Q_\tau(X,Y) &:= f(Y) + \langle \nabla f(Y), X - Y\rangle + \frac{\tau}{2}\|X-Y\|_F^2 + P(X) \\ &= \frac{\tau}{2}\|X - G\|_F^2 + P(X) + f(Y) - \frac{1}{2\tau}\|\nabla f(Y)\|_F^2, \end{aligned} \tag{11}$$

where $\tau > 0$ is a given parameter, $G = Y - \tau^{-1}\nabla f(Y)$. Since (11) is a strongly convex function of X, $Q_\tau(X,Y)$ has a unique minimizer which we denote by

$$S_\tau(G) := \arg\min\{Q_\tau(X,Y) \mid X \in \mathrm{dom}P\}. \tag{12}$$

First, we present a general algorithm that uses (12) to update the current iterate.

Algorithm 1:

Choose $X^0 = X^{-1} \in \mathrm{dom}P$, $t^0 = t^{-1} \in [1, \infty)$. For $k = 0, 1, 2, ...$, generate X^{k+1} from X^k according to the following iteration:

Step 1. Set $Y^k = X^k + \frac{t^{k-1}-1}{t^k}(X^k - X^{k-1})$.

Step 2. Set $G^k = Y^k - (\tau^k)^{-1}\nabla f(Y^k)$, where $\tau^k > 0$, and compute $S_{\tau^k}(G^k)$.

Step 3. Choose a stepsize $\alpha^k > 0$ and set $X^{k+1} = X^k + \alpha^k(S_{\tau^k}(G^k) - X^k)$.

Step 4. Choose $t^{k+1} \in [1, \infty)$ satisfying

$$(t^{k+1})^2 - t^{k+1} \leq (t^k)^2. \tag{13}$$

We note that Y^k may be outside of $\mathrm{dom}P$, and hence we need f to be smooth outside of $\mathrm{dom}P$. For the regularized semidefinite linear least squares problem (10), Y^k may not be a positive semidefinite matrix, but $f(X) = \frac{1}{2}\|\mathcal{A}(X) - b\|_2^2 + \mu\langle I, X\rangle$ is smooth on $\Re^{n\times n}$ and so we are able to compute $S_{\tau^k}(G^k)$. In addition, since $S_{\tau^k}(G^k) \in \mathcal{S}_+^n$, we have $X^k \in \mathcal{S}_+^n$ if $\alpha^k \leq 1$ for all k.

For the vector case where $n = 1$ in (8), Fukushima and Mine [22] studied a proximal gradient descent method using (11) to compute a descent direction (i.e., Algorithm 1 with $t^k = 1$ for all k) with stepsize α^k chosen by an Armijo-type rule. If P is separable, the minimum point $S_\tau(G)$ can be found in closed form, which is an advantage of algorithms using (12) to update the current point (i.e., $\alpha^k = 1$ for all k) or compute a direction for large-scale optimization problems [24, 44, 46, 47]. When the Algorithm 1, with fixed constants $\tau^k > 0$, $t^k = 1$, and $\alpha^k = 1$ for all k, is applied to the problem (5), i.e., (8) with $f(X) = \frac{1}{2}\|AX - b\|_2^2$, $P(X) = \mu\|X\|_1$ and $n = 1$ (hence $X \in \Re^m$), it is the popular *iterative shrinkage/thresholding* (IST) algorithms that have been developed and analyzed independently by many researchers [13, 20, 21, 24].

When $P \equiv 0$ in the problem (8), Algorithm 1 with $t^k = 1$ for all k reduces to the standard gradient algorithm. For the gradient algorithm, it is known that the sequence of function values $F(X^k)$ can converge to the optimal function value $\inf_{X\in\Re^{m\times n}} F(X)$ at a sublinear convergence rate that is no worse than $O(1/k)$. That is, $F(X^k) - \inf_{X\in\Re^{m\times n}} F(X) \leq O(1/k)\ \forall k$. The following theorem gives the $O(L_f/\epsilon)$ iteration complexity for the Algorithm 1 with $\tau^k = L_f$, $t^k = 1$, and $\alpha^k = 1$ for all k when it is applied to solve (8). Since $\Re^{m\times n}$ can be identified with $\Re^{mn}$ and $\mathrm{dom}P \subset \mathrm{dom}f$, its proof is nearly identical to [4, Theorem 3.1] and is omitted. In what follows, $\mathcal{X}^*$ denotes the set of optimal solutions.

Theorem 2.1 *Assume that $\mathcal{X}^* \neq \emptyset$. Let $\{X^k\}$ be the sequence generated by the Algorithm 1 with $\tau^k = L_f$, $t^k = 1$, and $\alpha^k = 1$ for all k. Then, for any $k \geq 1$, we have*

$$F(X^k) - F(X^*) \ \leq \ \frac{L_f\|X^0 - X^*\|_F^2}{2k} \qquad \forall X^* \in \mathcal{X}^*.$$

By Theorem 2.1, it can be shown that for IST algorithms, we have

$$F(X^k) - \inf_{X \in \Re^{m \times n}} F(X) \leq O(L_f/k) \quad \forall k,$$

and so, for any $\epsilon > 0$, these algorithms terminate in $O(L_f/\epsilon)$ iterations with an ϵ-optimal solution. Hence the sequence $\{X^k\}$ converges slowly.

In the smooth setting, i.e., when $P \equiv 0$ in the problem (8), Nesterov [33] proposed an algorithm that does not require more than one gradient evaluation at each iteration, but use only interpolation strategy to achieve $O(1/k^2)$ iteration complexity. Recently, Beck and Teboulle [4] extended Nesterov's algorithm in [33] to solve (8) with $n = 1$ and $\mathrm{dom}P = \Re^m$ and showed that the generated sequence $\{X^k\}$ can achieve the following complexity:

$$F(X^k) - \inf_{X \in \Re^{m \times n}} F(X) \ \leq \ O(L_f/k^2) \quad \forall k,$$

so that for any $\epsilon > 0$, the algorithm terminates in $O(\sqrt{L_f/\epsilon})$ iterations with an ϵ-optimal solution. And more recently, Tseng [43] proposed a unified framework and simpler analysis of the $O(\sqrt{L_f/\epsilon})$ algorithms extended to solve (8). As noted in [43], the condition (13) allows $\{t^k\}$ to increase, but not too fast. For fastest convergence, $\{t^k\}$ should increase as fast as possible. The choice $t^k = \frac{k+2}{2}$ satisfies (13). We can alternatively solve (13) with the inequality replaced by equality, yielding

$$t^{k+1} = \frac{1 + \sqrt{1 + 4(t^k)^2}}{2},$$

which tends to ∞ somewhat faster and is used in [4]. This choice will be used for our proposed algorithm in Section 3.

The following lemma is the well known property for a smooth function with Lipschitz continuous gradient; see [5].

Lemma 2.1 *Assume that ∇f is Lipschitz continuous on* $\mathrm{dom}P$ *satisfying* (9). *Then*

$$f(X) \leq f(Y) + \langle \nabla f(Y), X - Y \rangle + \frac{L_f}{2}\|X - Y\|_F^2, \quad \forall X, Y \in \mathrm{dom}P.$$

By Lemma 2.1, we obtain that for any $\tau \geq L_f$,

$$F(S_\tau(G)) \ \leq \ Q_\tau(S_\tau(G), Y), \tag{14}$$

where $G = Y - \tau^{-1}\nabla f(Y)$.

The following theorem gives the $O(\sqrt{L_f/\epsilon})$ iteration complexity for the Algorithm 1 when $t^k \geq \frac{k+2}{2}$, $\tau^k = L_f$, and $\alpha^k = 1$ for all k. For the proof, see [43, Corollary 2].

Theorem 2.2 *Let $\{X^k\}$, $\{Y^k\}$, $\{t^k\}$ be the sequences generated by the Algorithm 1 with $t^0 = 1$, $t^k \geq \frac{k+2}{2}$, $\tau^k = L_f$, and $\alpha^k = 1$ for all k. Then, for any $X \in \mathrm{dom}P$ with $F(X) \leq \inf_{X \in \Re^{m \times n}} F(X) + \epsilon$, we have*

$$\min_{i=0,1,\ldots,k+1} \{F(X^i)\} \leq F(X) + \epsilon \text{ whenever } k \geq \sqrt{\frac{4L_f \|X - X^0\|_F^2}{\epsilon}} - 2.$$

Remark 1 *A drawback of the Algorithm 1 with $\tau^k = L_f$ for all k is that the Lipschitz constant L_f is not always known or computable in advance, but this can be relaxed by making an initial estimate of L_f and increasing the estimate by a constant factor and repeating the iteration whenever (14) is violated; see [4].*

3 An accelerated proximal gradient algorithm for matrix completion problems

In this section, we discuss the application of the Algorithm 1, with $t^k \geq \frac{k+2}{2}$, $\tau^k = L_f$, $\alpha^k = 1$ for all k, to the matrix completion problem. We design efficient heuristics to enable our proposed algorithm to handle large-scale matrix completion problems, and also to solve them very efficiently. In section 3.1, we briefly introduce the matrix completion problem. In section 3.2, we review recent algorithms proposed to solve the matrix completion problem. In section 3.3, we describe our proposed algorithm for solving the nuclear norm regularized linear least squares problem (7). In section 3.4, we introduce three techniques to accelerate the convergence of our algorithm.

3.1 Introduction of the matrix completion

Recent interests in many areas of engineering and science have focused on the recovery of an unknown matrix from a sampling of its entries. This is known as the matrix completion problem. But this problem is ill-posed because there are fewer samples than entries and we have infinitely many completions. In many instances, however, the matrix we wish to recover has low rank or nearly low rank. The well-known Netflix problem [2] is a good example: in the area of recommender systems, users submit ratings on a subset of entries in a database, and the vendor provides recommendations based on the user's preferences. This yields a matrix M with users as rows and items as columns whose (i, j)-th entry M_{ij} is the rating given by the i-th user to j-th item. Because most users only rate a few items, one would like to infer their preference for unrated items. That means we want to fill in the missing entries of the matrix based on the small portion of entries observed. In this case, the data matrix of all user-ratings may be approximately low-rank because it is commonly believed that only a few factors contribute to an individual's preferences.

The above matrix completion problem can be cast as the following minimization problem:

$$\min_{X\in\Re^{m\times n}} \Big\{\mathrm{rank}(X) \,:\, X_{ij} = M_{ij},\ (i,j)\in\Omega\Big\}, \tag{15}$$

where M is the unknown matrix with p available sampled entries and Ω is a set of pairs of indices of cardinality p. This is a special case of the affine rank minimization (1) with $\mathcal{A}(X) = X_\Omega$, where X_Ω is the vector in $\Re^{|\Omega|}$ obtained from X by selecting those elements whose indices are in Ω. Recently, Candés and Recht [11] proved that a random low-rank matrix can be recovered exactly with high probability from a rather small random sample of its entries, and it can be done by solving an aforementioned convex relaxation (2) of (1), i.e.,

$$\min_{X\in\Re^{m\times n}} \Big\{\|X\|_* \,:\, X_{ij} = M_{ij},\ (i,j)\in\Omega\Big\}. \tag{16}$$

In [11], the convex relaxation (16) was solved using SDPT3 [41], which is one of the most advanced semidefinite programming solvers. The problem (16) can be reformulated as a semidefinite program as follows; see [38] for details:

$$\begin{array}{ll} \min\limits_{X,W_1,W_2} & \frac{1}{2}\left(\langle W_1, I_m\rangle + \langle W_2, I_n\rangle\right) \\ \text{subject to} & X_{ij} = M_{ij},\ (i,j)\in\Omega, \quad \left(\begin{array}{cc} W_1 & X \\ X^T & W_2 \end{array}\right) \succeq 0. \end{array} \tag{17}$$

But the problem (17) has one $(m+n)\times(m+n)$ semidefinite constraint and p affine constraints. The solver SDPT3 and others like SeDuMi are based on interior-point methods and they are not suitable for problems with large $m+n$ or p because the computational cost grows like $O(p(m+n)^3 + p^2(m+n)^2 + p^3)$ and the memory requirement grows like $O((m+n)^2 + p^2)$.

3.2 Review of existing algorithms for matrix completion

In this section, we review two recently developed algorithms [6, 30] for solving the matrix completion problem.

First of all, we consider the following minimization problem:

$$\min_{X\in\Re^{m\times n}} \frac{\tau}{2}\|X - G\|_F^2 + \mu\|X\|_*, \tag{18}$$

where G is a given matrix in $\Re^{m\times n}$. If $G = Y - \tau^{-1}\mathcal{A}^*(\mathcal{A}(Y) - b)$, then (18) is a special case of (11) with $f(X) = \frac{1}{2}\|\mathcal{A}(X) - b\|_2^2$ and $P(X) = \mu\|X\|_*$ when we ignore the constant term. The problem (18) has a unique analytical solution which can be computed via the singular value decomposition (SVD) of G. Suppose the SVD of G is given by:

$$G = U\Sigma V^T, \quad \Sigma = \mathrm{Diag}(\sigma),$$

where U and V are respectively $m\times q$ and $n\times q$ matrices with orthogonal columns, $\sigma\in\Re^q$ is the vector of positive singular values arranged in descending order $\sigma_1 \geq \sigma_2 \geq \ldots \geq$

$\sigma_q > 0$, with $q \leq \min\{m, n\}$. (Unless specified otherwise, we will always assume that the SVD of a matrix is given in the reduced form above.) For a given vector $x \in \Re^q$, we let $x_+ = \max\{x, 0\}$, where the maximum is taken component-wise. By [6, Theorem 2.1] or [30, Theorem 3], the solution $S_\tau(G)$ of (18) is given by

$$S_\tau(G) = U\text{Diag}((\sigma - \mu/\tau)_+)V^T. \tag{19}$$

Hence, by using SVD of G, we can obtain a closed form solution of (18).

Recently, Ma et al. [30] proposed a fixed point continuation (abbreviated FPC) algorithm for solving (7) and a Bregman iterative algorithm for solving (2). Their numerical results on randomly generated and real matrix completion problems demonstrated that the FPC algorithm is much faster than semidefinite solvers such as SDPT3. This FPC algorithm for solving (7) is a matrix extension of the fixed point continuation algorithm proposed in [24] for an ℓ_1-regularized convex minimization problem in $\Re^m$. At each iteration k, the FPC algorithm solves (18) with $\tau^k > \lambda_{\max}(\mathcal{A}^*\mathcal{A})/2$, where $\lambda_{\max}(\mathcal{A}^*\mathcal{A})$ is the largest eigenvalue of $\mathcal{A}^*\mathcal{A}$. Hence this algorithm (without continuation) is a special case of the Algorithm 1, with $\tau^k > \lambda_{\max}(\mathcal{A}^*\mathcal{A})/2$, $t^k = 1$, $\alpha^k = 1$ for all k, and can be expressed as follows:

$$\begin{cases} X^k = S_{\tau^k}(G^k) \\ G^{k+1} = X^k - (\tau^k)^{-1}\mathcal{A}^*(\mathcal{A}(X^k) - b). \end{cases} \tag{20}$$

They proved the following global convergence result for the fixed point algorithm when it is applied to solve (7). For the proof, see [30, Theorem 4].

Theorem 3.1 *Let $\{X^k\}$ be the sequence generated by the fixed point algorithm with $\tau^k > \lambda_{\max}(\mathcal{A}^*\mathcal{A})/2$. Then X^k converges to some $X^* \in \mathcal{X}^*$.*

However this algorithm may terminate in $O(L_f/\epsilon)$ iterations with an ϵ-optimal solution; see Theorem 2.1 with $\tau^k \geq \lambda_{\max}(\mathcal{A}^*\mathcal{A})$.

Recently, Cai, et al. [6] proposed a singular value thresholding (abbreviated SVT) algorithm for solving the following Tikhonov regularized version of (2):

$$\min_{X \in \Re^{m\times n}} \Big\{\|X\|_* + \frac{1}{2\beta}\|X\|_F^2 \,:\, \mathcal{A}(X) = b\Big\}, \tag{21}$$

where $\beta > 0$ is a given parameter. Note the addition of the strongly convex quadratic term $\|X\|_F^2/(2\beta)$ to (2) in (21). The SVT algorithm can be expressed as follows:

$$\begin{cases} X^k = S_{\tau^k}(G^k) \\ G^{k+1} = G^k - \delta^k\mathcal{A}^*(\mathcal{A}(X^k) - b), \end{cases} \tag{22}$$

where $\tau^k = 1$ for all k and δ^k is a positive step size. In (22), $S_{\tau^k}(G^k)$ has the form in (27) but with μ replaced by β. This SVT algorithm is a gradient method applied to the

dual problem of (21), where each step moves the current dual iterate in the direction of the gradient.

They proved the following global convergence for the SVT algorithm. For the proof, see [6, Corollary 4.5].

Theorem 3.2 *Let $\{X^k\}$ be the sequence generated by the SVT algorithm applied to (21) with $0 < \delta^k < 2/\lambda_{\max}(\mathcal{A}^*\mathcal{A})$ for all k. Then X^k converges to the unique solution of (21).*

They also showed that if β goes to ∞, then the sequence of optimal solutions X^*_β for (21), which is assumed to be feasible, converges to the optimal solution of (2) with minimum Frobenius norm. Hence the SVT algorithm approximately solves (2), in particular (16) for sufficiently large β. The numerical results in [6] demonstrated that the SVT algorithm is very efficient in recovering large-scale matrices of small rank when it is applied to (21) with an appropriately large parameter β to approximately solve (16).

The main computational cost in each iteration of the FPC and SVT algorithms lies in computing the SVD of G^k. In [30], Ma et al. uses a fast Monte Carlo algorithm such as the Linear Time SVD algorithm developed by Drineas et al. [16] to reduce the time for computing the SVD. In addition, they compute only the predetermined sv^k largest singular values and corresponding singular vectors to further reduce the computational time at each iteration k. The expected number sv^k is set by the following procedure. In the k-th iteration, sv^k is set to be equal to the number of components in the vector ϱ^{k-1} that are no less than $\upsilon^k \max(\varrho^{k-1})$, where υ^k is a small positive number and $\max(\varrho^{k-1})$ is the largest component in the vector ϱ^{k-1} used to form $X^{k-1} = U^{k-1}\mathrm{Diag}(\varrho^{k-1})(V^{k-1})^T$. And if the non-expansive property (see [30, Lemma 1]) is violated 10 times, sv^k is increased by 1. In contrast, Cai, et al. [6] used PROPACK [27] (a variant of the Lanczos algorithm) to compute a partial SVD of G^k. They also compute only the predetermined sv^k largest singular values and corresponding singular vectors to reduce the computational time at each iteration k. The procedure to set sv^k is as follows. In the k-th iteration, set $\mathrm{sv}^k = r^{k-1} + 1$ where r^{k-1} is the number of positive singular values of X^{k-1}. If all of the computed singular values are greater than or equal to β, then sv^k is increased by 5 repeatedly until some of the singular values fall below β.

3.3 An accelerated proximal gradient singular value thresholding algorithm

In this section, we describe an accelerated proximal gradient singular value thresholding algorithm for solving (7).

The following lemma shows that the optimal solution set of (7) is bounded.

Lemma 3.1 *For each $\mu > 0$, the optimal solution set $\mathcal{X}^*$ of (7) is bounded. In addition, for any $X^* \in \mathcal{X}^*$, we have*

$$\|X^*\|_F \leq \chi \tag{23}$$

where

$$\chi = \begin{cases} \min\{\|b\|_2^2/(2\mu), \|X_{LS}\|_*\} & \text{if } \mathcal{A} \text{ is surjective} \\ \|b\|_2^2/(2\mu) & \text{otherwise.} \end{cases}$$

with $X_{LS} = \mathcal{A}^*(\mathcal{A}\mathcal{A}^*)^{-1}b$.

Proof. Considering the objective value of (7) at $X = 0$, we obtain that for any $X^* \in \mathcal{X}^*$,

$$\mu\|X^*\|_* \leq \frac{1}{2}\|\mathcal{A}(X^*) - b\|_2^2 + \mu\|X^*\|_* \leq \frac{1}{2}\|b\|_2^2.$$

This together with the fact that $\|X^*\|_F \leq \|X^*\|_*$ implies $\|X^*\|_F \leq \|b\|_2^2/(2\mu)$. In addition, if $\mathcal{A}$ is surjective, then by considering the objective value of (7) at $X = X_{LS}$, we obtain that for any $X^* \in \mathcal{X}^*$,

$$\mu\|X^*\|_* \leq \frac{1}{2}\|\mathcal{A}(X^*) - b\|_2^2 + \mu\|X^*\|_* \leq \mu\|X_{LS}\|_*.$$

This together with the fact that $\|X^*\|_F \leq \|X^*\|_*$ implies (23) ∎

Since (7) is a special case of (8) with $f(X) = \frac{1}{2}\|\mathcal{A}(X) - b\|_2^2$ and $P(X) = \mu\|X\|_*$, we can solve (18) with $\tau = L_f$ to update the current matrix at each iteration. Hence the updated matrix is given by

$$S_\tau(G) = U\mathrm{Diag}\Big((\sigma - \mu/\tau)_+\Big)V^T,$$

where U, V, σ are obtained from the SVD: $Y - \tau^{-1}(\mathcal{A}^*(\mathcal{A}(Y) - b)) = U\mathrm{Diag}(\sigma)V^T$.

We now describe formally the accelerated proximal gradient singular value thresholding (abbreviated APG) algorithm for solving (7).

APG algorithm:

For a given $\mu > 0$, choose $X^0 = X^{-1} \in \Re^{m\times n}$, $t^0 = t^{-1} = 1$. For $k = 0, 1, 2, ...$, generate X^{k+1} from X^k according to the following iteration:

Step 1. Set $Y^k = X^k + \frac{t^{k-1}-1}{t^k}(X^k - X^{k-1})$.

Step 2. Set $G^k = Y^k - (\tau^k)^{-1}\mathcal{A}^*(\mathcal{A}(Y^k) - b)$, where $\tau^k = L_f$. Compute $S_{\tau^k}(G^k)$ from the SVD of G^k.

Step 3. Set $X^{k+1} = S_{\tau^k}(G^k)$.

Step 4. Compute $t^{k+1} = \frac{1+\sqrt{1+4(t^k)^2}}{2}$.

We note that this algorithm is a special case of the Algorithm 1 with $t^0 = 1$, $t^{k+1} = \frac{1+\sqrt{1+4(t^k)^2}}{2}$, $\tau^k = L_f$, and $\alpha^k = 1$ for all k.

The following corollary gives an upper bound on the number of iterations for the APG algorithm to achieve ϵ-optimality.

Corollary 3.1 *Let $\{X^k\}$, $\{Y^k\}$, $\{t^k\}$ be the sequences generated by APG. Then, for any $k \geq 1$, we have*

$$F(X^k) - F(X^*) \leq \frac{2L_f \|X^* - X^0\|_F^2}{(k+1)^2}, \qquad \forall X^* \in \mathcal{X}^*. \tag{24}$$

Hence

$$F(X^k) - F(X^*) \leq \epsilon \quad \text{whenever } k \geq \sqrt{\frac{2L_f}{\epsilon}} \left(\|X^0\|_F + \chi\right) - 1, \tag{25}$$

where χ is defined as in Lemma 3.1.

Proof. Since $\Re^{m\times n}$ can be identified with $\Re^{mn}$ and $\mathcal{X}^* \neq \emptyset$, By [4, Theorem 4.1], we obtain (24). By using the triangular inequality, $\|X^* - X^0\|_F \leq \|X^*\|_F + \|X^0\|_F$, and Lemma 3.1, we get the required result in (25). ∎

We note that the $O(\sqrt{L_f/\epsilon})$ iteration complexity result in Corollary 3.1 holds with τ^k smaller than L_f as long as (14) is satisfied with $\tau = \tau^k$, $G = G^k$, $Y = Y^k$; see [4] for details.

Remark 2 *We observe that the APG algorithm is as simple as the FPC and SVT algorithms, and yet it has a better iteration complexity. As indicated in section 3.2, the FPC algorithm has the $O(L_f/\epsilon)$ iteration complexity. For the SVT algorithm, there is no known iteration complexity. Hence the main advantage of the APG algorithm over the FPC and the SVT algorithms is its $O(\sqrt{L_f/\epsilon})$ iteration complexity. We can see this iteration complexity advantage over the FPC (without continuation) from the numerical tests in section 4.1.*

Remark 3 *The APG algorithm can easily be modified to solve the regularized semidefinite linear least squares problem (10). The only change required is to replace $S_{\tau^k}(G^k)$ in Step 2 of the algorithm by $S_{\tau^k}(G^k) = Q\text{Diag}((\lambda - \mu/\tau)_+)Q^T$, where $Q\text{Diag}(\lambda)Q^T$ is the eigenvalue decomposition of G^k. Since (10) is a special case of (8) and $\mathcal{X}^* \neq \emptyset$, the $O(\sqrt{L_f/\epsilon})$ iteration complexity result in Corollary 3.1 still holds when the APG algorithm is applied to solve (10).*

Just like the FPC and SVT algorithms, the APG algorithm needs to compute the SVD of G^k at each iteration. Our implementation uses PROPACK to compute a partial SVD of G^k. But PROPACK can not automatically compute only those singular values greater

than μ/τ^k even though one can choose the number of singular values to compute. Hence we must choose the predetermined number sv^k of singular values to compute in advance at each iteration k. We use the following simple formula to update sv^k. Starting from $\mathrm{sv}^0 = 5$, we set sv^{k+1} as follows:

$$\mathrm{sv}^{k+1} = \begin{cases} \mathrm{svp}^k + 1 & \text{if } \mathrm{svp}^k < \mathrm{sv}^k \\ \mathrm{svp}^k + 5 & \text{if } \mathrm{svp}^k = \mathrm{sv}^k \end{cases} \tag{26}$$

where svp^k is the number of positive singular values of X^k.

3.4 Accelerating the APG algorithm

In this section, we introduce three techniques to accelerate the convergence of the APG algorithm.

The first technique uses a linesearch-like technique. In practice, it is often too conservative to set $\tau^k = L_f$ for all k in the APG algorithm. To accelerate the convergence of the APG algorithm, it is desirable to take a smaller value for τ^k by performing a linesearch-like procedure. In the algorithm below, we incorporated the linesearch-like acceleration strategy in the APG algorithm.

APGL algorithm.

Let $\mu > 0$ be a fixed regularization parameter, and $\eta \in (0, 1)$ be a given constant. Choose $X^0 = X^{-1} \in \Re^{m\times n}$. Set $t^0 = t^{-1} = 1$ and $\tau^0 = L_f$.
For $k = 0, 1, 2, ...$, generate X^{k+1} according to the following iteration:

Step 1. Set $Y^k = X^k + \frac{t^{k-1}-1}{t^k}(X^k - X^{k-1})$.

Step 2. Set $\widehat{\tau}_0 = \eta\tau^{k-1}$.

For $j = 0, 1, 2, \ldots,$
Set $G = Y^k - (\widehat{\tau}_j)^{-1}\mathcal{A}^*(\mathcal{A}(Y^k) - b)$, compute $S_{\widehat{\tau}_j}(G)$.
If $F(S_{\widehat{\tau}_j}(G)) \le Q_{\widehat{\tau}_j}(S_{\widehat{\tau}_j}(G), Y^k)$,
set $\tau^k = \widehat{\tau}_j$, stop;
else,
$\widehat{\tau}_{j+1} = \min\{\eta^{-1}\widehat{\tau}_j, L_f\}$
end
end

Step 3. Set $X^{k+1} = S_{\tau^k}(G)$.

Step 4. Compute $t^{k+1} = \frac{1+\sqrt{1+4(t^k)^2}}{2}$.

Besides accelerating the convergence of the APG algorithm, the linesearch-like procedure in Step 2 of the APGL algorithm has another important advantage. One can see from Step 2 of the APGL algorithm that τ^k has the form $\tau^k = \eta^{l_k} L_f$ for some nonnegative integer l_k for each k. Thus τ^k is typically smaller than L_f, and this implies that the vector $(\sigma - \mu/\tau^k)_+$ would have fewer positive components when we compute $S_{\tau^k}(G)$ from the SVD of G. Thus the iterate X^{k+1} would generally have lower rank compared to that of the corresponding iterate generated by the APG algorithm.

As indicated in [6], the convergence of the SVT algorithm for the matrix completion problem of the form (21) is guaranteed provided that $0 < \delta^k < 2$ for all k; see Theorem 3.2. But it was observed that this stepsize range is too conservative and the convergence of the algorithm is typically very slow. Hence in [6], the actual algorithm used a much larger stepsize ($\delta^k = 1.2\frac{nm}{p}$ for all k) to accelerate the convergence of the SVT algorithm and the authors gave a heuristic justification. But for our APGL algorithm above, the complexity result in Corollary 3.1 remains valid as noted in section 3.3.

The second technique to accelerate the convergence of the APG algorithm is a continuation technique. When we update the current point X^k at iteration k, we have to obtain $S_{\tau^k}(G^k)$ via the SVD of G^k. This is the main computation of the APG algorithm. However it suffices to know those singular values and corresponding singular vectors greater than the scaled regularization parameter μ/τ^k. Hence if the parameter is larger, then the number of singular values to be evaluated is smaller. But the target parameter $\bar{\mu}$ is usually chosen to a moderately small number. This motivates us to use the continuation technique employed in [24, 30]. If the problem (7) is to be solved with the target parameter value $\mu = \bar{\mu}$, we propose solving a sequence of problems (7) defined by a decreasing sequence $\{\mu^0, \mu^1, \ldots, \mu^\ell = \bar{\mu}\}$ with a given finite positive integer ℓ. When a new problem, associated with μ^{j+1} is to be solved, the approximate solution for the current problem with $\mu = \mu^j$ is used as the starting point. In our numerical experiments in sections 4.2, 4.3, and 4.4, we set $\bar{\mu} = 10^{-4}\mu^0$ with the initial $\mu^0 = \|\mathcal{A}^*(b)\|_2$, and update $\mu^k = \max\{0.7\mu^{k-1}, \bar{\mu}\}$ at iteration k. The above updating formula for μ^k is guided by the observation that a greater reduction on μ^k results in more singular values being evaluated but l is smaller, while a smaller reduction on μ^k has the opposite effect. The reduction factor 0.7 was found after some experimentation. Another updating rule for μ^k used in FPC algorithm [30] was tested in our setting but it was worse than the above updating rule.

Our computational experience indicate that the performance of the APG and FPC algorithms with continuation techniques is generally superior to that of directly applying the APG and FPC algorithms to (7) with the target value $\mu = \bar{\mu}$; see section 4.1.

The third technique to accelerate the convergence of the APG algorithm is a truncation technique. We note that for the APG algorithm without any acceleration techniques, the iterate X^k is generally not low-rank before the final phase of the algorithm, but the number of positive singular values of X^k typically would separate into two clusters with the first cluster having much large mean value than that of the second cluster. One may view the

number of singular values in the first cluster as a good estimate on the rank of the low-rank optimal solution of (7). The second cluster of smaller positive singular values (at the level of about μ/τ^k) can be attributed to the presence of noise in the given data b, and also the fact that X^k has yet to converge to a low-rank optimal solution of (7). We have observed that the second cluster of smaller singular values can generally be discarded without affecting the convergence of the APG algorithm. This motivates us to set the second cluster of small positive singular values to zero when the new iterate is updated. The procedure is as follows. Let the SVD of G^k be $G^k = U\mathrm{Diag}(\sigma)V^T$, and $\varrho = (\sigma - \mu/\tau^k)_+$. Let q be the number of positive components of ϱ. We evaluate $\chi_j := \mathrm{mean}(\varrho(1:j))/\mathrm{mean}(\varrho(j+1:q))$ for $j = 1, \ldots, q-1$. If $\chi_j < \mathrm{gap}\ \forall\ j = 1, \ldots, q-1$ (where gap is a given positive scalar), set $X^{k+1} = U\mathrm{Diag}(\varrho)V^T$; otherwise, let r be the smallest integer such that $\chi_r \geq \mathrm{gap}$, set $X^{k+1} = U\mathrm{Diag}(\bar{\varrho})V^T$, where $\bar{\varrho}$ is obtained from ϱ by setting $\bar{\varrho}_i = \varrho_i$ for $i = 1, ..., r$, and $\bar{\varrho}_i = 0$ for $i = r+1, ..., q$. In our implementation of the APG algorithm, we set gap = 5 after some experimentation on small problems.

Remark 4 *As noted in [6], there are two main advantages of the SVT algorithm over the FPC (without continuation) and the APG algorithms when the former is applied to solve the problem (21) arising from matrix completions. First, selecting a large β in (21) gives a sequence of low-rank iterates. In contrast, the parameter μ for (7) is chosen to be a moderately small number and so many iterates at the initial phase of the FPC or APG algorithms may not have low rank even though the optimal solution itself has low rank. We observed this property when we applied the FPC and APG algorithms to solve the matrix completion problem. But this drawback can be ameliorated by using the linesearch-like and continuation techniques. Second, the matrix G^k in (22) is sparse for all k because the sparsity pattern of Ω is fixed throughout. This makes the SVT algorithm computationally attractive. In contrast, the matrix G^k in (20) and the APG algorithm may not be sparse. But, by using the continuation and truncation techniques, X^k in (20) and Y^k in the APG algorithm can keep the low-rank property for all k. In addition, X^k, Y^k need not be computed explicitly, but their low-rank SVD factors are stored so that matrix-vector products can be evaluated in PROPACK or any other iterative algorithms for computing a partial SVD. Also the matrices $\mathcal{A}^*(\mathcal{A}(X^k) - b)$ and $\mathcal{A}^*(\mathcal{A}(Y^k) - b)$ are sparse because of the sparsity pattern of Ω. Therefore the matrix G^k in (20) and the APG algorithm is typically the sum of a low-rank matrix and a sparse matrix. The last property can also make the FPC and APG algorithms computationally as attractive as the SVT algorithm.*

3.5 A stopping condition for the APG and APGL algorithms

The natural stopping condition for the unconstrained convex minimization problem (7) is that $\delta(X) := \mathrm{dist}(0, \partial(f(X) + \mu\|X\|_*))$ is sufficiently small, where $f(X) = \frac{1}{2}\|\mathcal{A}(X) - b\|_2^2$ and $\partial(\cdot)$ denotes the sub-differential. Here $\mathrm{dist}(x, S)$ denotes the distance between a point x and a set S.

In practice, it is difficult to compute $\delta(X)$, but in the course of running the APG and APGL algorithms, one can actually get a good upper bound on $\delta(X)$ without incurring extra computational cost as follows. At the k-th iteration, let $G^k = Y^k - (\tau^k)^{-1}\nabla f(Y^k)$. Observe that

$$\partial(\mu\|X^{k+1}\|_*) \;\ni\; \tau^k(G^k - X^{k+1}) \;=\; \tau^k(Y^k - X^{k+1}) - \nabla f(Y^k).$$

Let

$$S^{k+1} := \tau^k(Y^k - X^{k+1}) + \nabla f(X^{k+1}) - \nabla f(Y^k) = \tau^k(Y^k - X^{k+1}) + \mathcal{A}^*(\mathcal{A}(X^{k+1}) - \mathcal{A}(Y^k)). \quad (27)$$

Thus we have $S^{k+1} \in \partial(f(X^{k+1}) + \mu\|X^{k+1}\|_*)$ and hence the following upper bound for $\delta(X^{k+1})$:

$$\delta(X^{k+1}) \;\le\; \|S^{k+1}\|_F.$$

From the above, we derive the following stopping condition for the APG and APGL algorithms:

$$\frac{\|S^{k+1}\|_F}{\tau^k \max\{1, \|X^{k+1}\|_F\}} \;\le\; \text{Tol}, \quad (28)$$

where Tol is a moderately small tolerance.

4 Numerical experiments on matrix completion problems

In this section, we report some numerical results for solving a collection of matrix completion problems of the form (7) and (10). In section 4.1, we compare our APG algorithm with a variant of the FPC algorithm [30] for solving (7) on randomly generated matrix completion problems with moderate dimensions. In section 4.2, we present some numerical results for solving (7) on a set of large-scale randomly generated matrix completion problems with noise/without noise. In section 4.3, we present some numerical results for solving (10) on a set of large-scale randomly generated semidefinite matrix completion problems with noise/without noise. In section 4.4, we present some numerical results for solving (7) on real matrix completion problems. In section 4.5, we evaluate the performance of the APGL algorithm for solving (7) on randomly generated multivariate linear regression problems. In section 4.6, we report numerical results of the APGL algorithm for solving the problem of the form (10) arising from regularized kernel estimation.

We have implemented the APG and APGL algorithms in MATLAB, using PROPACK package to evaluate partial singular value decompositions. All runs are performed on an Intel Xeon 3.20GHz, running Linux and MATLAB (Version 7.6). Throughout the experiments, we choose the initial iterate to be $X^0 = 0$.

The random matrix completion problems we consider in our numerical experiments are generated as in [11]. For each (n, r, p) triple, where n (we set $m = n$) is the matrix dimension, r is the predetermined rank, and p is the number entries to sample, we repeat the following procedure 5 times. We generate $M = M_L M_R^T$ as suggested in [11], where M_L and M_R are $n \times r$ matrices with i.i.d. standard Gaussian entries. We then select a subset Ω of p elements uniformly at random from $\{(i, j) : i = 1, \ldots, n,\ j = 1, \ldots, n\}$. Hence

$$b = \mathcal{A}(M)$$

where the linear map $\mathcal{A}$ is given by

$$\mathcal{A}(X) = X_\Omega.$$

Here, X_Ω is the vector in $\Re^p$ that is obtained from X by selecting those elements whose indices are in Ω. For the above linear map, we have the Lipschitz constant $L_f = 1$ for (7). For each of the test problems, we set the regularization parameter in (7) to be $\bar{\mu} = 10^{-4}\mu^0$ where $\mu^0 = \|\mathcal{A}^*(b)\|_2$.

We also conduct numerical experiments on random matrix completion problems with noisy data. For the noisy random matrix completion problems, the matrix M are corrupted by a noise matrix Ξ, and

$$b = \mathcal{A}(M + \sigma\Xi),$$

where the elements of Ξ are i.i.d. standard Gaussian random variables. In our experiments, σ is chosen to be

$$\sigma \ = \ nf \, \frac{\|\mathcal{A}(M)\|_F}{\|\mathcal{A}(\Xi)\|_F},$$

where nf is a given noise factor.

The stopping criterion we use for the APG and APGL algorithms in our numerical experiments is as follows:

$$\frac{\|S^k\|_F}{\tau^{k-1} \max\{1, \|X^k\|_F\}} < \text{Tol}, \tag{29}$$

where Tol is a moderately small number, and S^k is defined as in (27). In our experiments, unless otherwise specified, we set $\text{Tol} = 10^{-4}$. In addition, we also stop the APG and APGL algorithms when

$$\frac{\left|\|\mathcal{A}(X^k) - b\|_2 - \|\mathcal{A}(X^{k-1}) - b\|_2\right|}{\max\{1, \|b\|_2\}} < 5 \times \text{Tol}. \tag{30}$$

We measure the accuracy of the computed solution X_{sol} of an algorithm by the relative error defined by:

$$\text{error} \ := \ \|X_{\text{sol}} - M\|_F / \|M\|_F, \tag{31}$$

where M is the original matrix.

Table 1: Comparison of APG and FPC, without using any acceleration techniques.

Unknown M				**FPC**			**APG**		
n/r	p	p/d_r	μ	iter	#sv	error	iter	#sv	error
100/10	5666	3	8.21e-03	7723	61	1.88e-01	655	13	1.06e-03
200/10	15665	4	1.05e-02	12180	96	2.45e-01	812	12	1.02e-03
500/10	49471	5	1.21e-02	10900	203	5.91e-01	1132	16	7.63e-04

Table 2: Comparison of APG and FPC, with continuation techniques.

Unknown M				**FPC-M**			**APG-C**		
n/r	p	p/d_r	μ	iter	#sv	error	iter	#sv	error
100/10	5666	3	8.21e-03	429	32	1.06e-03	74	10	1.46e-04
200/10	15665	4	1.05e-02	278	49	4.38e-04	73	10	1.02e-04
500/10	49471	5	1.21e-02	484	125	5.50e-04	72	10	8.06e-05

4.1 Comparison of the APG and FPC algorithms

In this section, we compare the performance of the APG and FPC algorithms with/without continuation techniques to solve (7) for randomly generated matrix completion problems with moderate dimensions.

As noted in section 3.2, the FPC algorithm solves the problem (7) and this algorithm (without continuation technique) can be considered as a special case of the Algorithm 1. As suggested in [30], we set $\tau^k = 1$ for all k in (20).

In this section, we set Tol in (29) to 5×10^{-5} to prevent the FPC algorithm from terminating prematurely with a poor relative error in the computed solution. In order to free ourselves from the distraction of having to estimate the number of singular values that are greater than μ^k/τ^k when computing $S_{\tau^k}(G^k)$ from a partial SVD of G^k, we compute the full SVD of G^k at each iteration k.

For the APG algorithm with continuation technique (which we call APG-C), we update $\mu^k = \max\{\kappa\mu^{k-1}, \bar{\mu}\}$ at iteration k whenever $k = 0 \pmod 3$ or (29) is satisfied and $\kappa = 0.7$. This updating formula for μ^k is guided by the observation mentioned in section 3.4. The thresholds 0.7 and 3 were found after some experimentation.

For the FPC algorithm with continuation technique in [30], the update strategy for μ^k is $\mu^k = \max\{0.25\mu^{k-1}, \bar{\mu}\}$ whenever (29) is satisfied with Tol $= 10^{-5}$ (In [30], the denominator of (29) is replaced by $\|X^{k-1}\|_F$). We call this version as FPC-M.

Tables 1 and 2 report the average number of iterations, the average number (#sv) of nonzero singular values of the computed solution matrix, and the average relative error (31), of 5 random runs without noise. The tables also report the mean value of the regularization parameter μ for the 5 runs and gives the ratio (denoted by p/d_r) between the number of sampled entries and the degrees of freedom in an $n \times n$ matrix of rank r. As indicated in [6], an $n \times n$ matrix of rank r depends on $d_r := r(2n - r)$ degrees of freedom. As can be seen

from Table 1, APG outperforms FPC greatly in terms of the number of iterations when no acceleration techniques are used. In addition, the solutions computed by the APG algorithm are much more accurate than those delivered by the FPC algorithm. From Table 2, we see that the APG-C algorithm also substantially outperformed the FPC-M algorithm in terms of iteration counts, and the former also delivered more accurate solutions. In conclusion, the APG algorithm is superior to the FPC algorithm with/without continuation. This conclusion is not surprising given that the former has better iteration complexity result than the latter.

4.2 Numerical experiments on large random matrix completion problems

In this section, we report the performance of the APGL algorithm with continuation and truncation techniques on randomly generated matrix completion problems with and without noise. We created the random matrices the same way described at the beginning of section 4.

Table 3 reports the results for random matrix completion problems without noise. In the table, we report the ratio p/d_r, the mean value of the regularization parameter μ, the mean value of the number of iterations, the number (#sv) of positive singular values of the computed solution matrix, the CPU time (in seconds), and the relative error (31) of 5 runs. As indicated in the table, it took the APGL algorithm no more than 70 iterations on the average to solve all the problems in our experiments. In addition, the errors of the computed solutions are all smaller than 4×10^{-4}. Observe that the APGL algorithm was able to solve random matrix completion problems with $m = n = 10^5$ each in less than 10 minutes. For each of the problems with $m = n = 10^5$, the percentage of entries sampled is about 0.12%. It is rather surprising that the original random low-rank matrix M can be recovered given only such a small portion of its entries.

Table 4 reports the same results as Table 3 for random matrix completion problems with noisy data. As indicated in the table, the APGL algorithm took no more than 65 iterations on the average to solve all the problems in our experiments, and yet the errors are all smaller than the noise level ($nf = 0.1$) in the given data. The errors obtained here are consistent with (actually more accurate) the theoretical result established in [7].

Table 3: Numerical results on random matrix completion problems without noise.

Unknown M					**Results**			
n	p	r	p/d_r	μ	iter	#sv	time	error
1000	119406	10	6	1.44e-02	38	10	2.66e+00	2.94e-04
	389852	50	4	5.39e-02	40	50	1.55e+01	3.08e-04
	569900	100	3	8.63e-02	48	100	4.79e+01	3.82e-04

Table 3: Numerical results on random matrix completion problems without noise.

Unknown M					**Results**			
n	p	r	p/d_r	μ	iter	#sv	time	error
5000	597973	10	6	1.38e-02	48	10	1.20e+01	2.53e-04
	2486747	50	5	6.10e-02	55	50	1.19e+02	3.79e-04
	3957533	100	4	1.03e-01	53	100	2.97e+02	2.77e-04
10000	1199532	10	6	1.36e-02	48	10	2.36e+01	2.62e-04
	4987078	50	5	5.96e-02	67	50	3.14e+02	1.96e-04
	7960222	100	4	9.94e-02	67	100	8.61e+02	2.69e-04
20000	2401370	10	6	1.35e-02	57	10	5.60e+01	2.09e-04
30000	3599920	10	6	1.35e-02	53	10	7.91e+01	2.72e-04
50000	5998352	10	6	1.35e-02	63	10	1.76e+02	1.75e-04
100000	12000182	10	6	1.34e-02	69	10	4.63e+02	2.16e-04

Table 4: Numerical results on random matrix completion problems with noise. The noise factor nf is set to 0.1.

Unknown M					**Results**			
n	p	r	p/d_r	μ	iter	#sv	time	error
1000 /0.10	119406	10	6	1.44e-02	34	10	2.62e+00	4.44e-02
	389852	50	4	5.39e-02	38	50	1.40e+01	5.45e-02
	569900	100	3	8.65e-02	44	100	3.99e+01	6.32e-02
5000 /0.10	597973	10	6	1.38e-02	41	10	1.41e+01	4.47e-02
	2486747	50	5	6.10e-02	54	50	1.18e+02	4.92e-02
	3957533	100	4	1.03e-01	47	100	2.37e+02	5.63e-02
10000 /0.10	1199532	10	6	1.37e-02	45	10	2.89e+01	4.48e-02
	4987078	50	5	5.97e-02	62	50	2.89e+02	5.01e-02
	7960222	100	4	9.95e-02	64	100	8.17e+02	5.81e-02
20000 /0.10	2401370	10	6	1.36e-02	49	10	5.98e+01	4.48e-02
30000 /0.10	3599920	10	6	1.35e-02	51	10	1.01e+02	4.49e-02
50000 /0.10	5998352	10	6	1.35e-02	55	10	2.02e+02	4.51e-02
100000 /0.10	12000182	10	6	1.34e-02	59	10	5.50e+02	4.52e-02

4.3 Numerical experiments on random positive semidefinite matrix completion problems

Here evaluate the performance of the APGL algorithm, with continuation and truncation techniques, for solving semidefinite linear least squares problems (10) arising from positive semidefinite random matrix completions. The format of the experiment is the same as that in section 4.2, and the data matrix M is generated in the same way but with $M_R = M_L$. The index set Ω is generated by selecting p elements uniformly at random from the set $\{(i,j) : i = 1,\ldots,j,\ j = 1,\ldots,n\}$.

From Tables 5 and 6, we observe that the APGL algorithm performed very well on random positive semidefinite matrix completion problems.

Table 5: Numerical results on semidefinite random matrix completion problems without noise.

Unknown M					**Results**			
n	p	r	p/d_r	μ	iter	#sv	time	error
1000	60497	10	6	1.59e-02	38	10	2.47e+00	2.41e-04
	195544	50	4	6.34e-02	39	50	9.86e+00	3.22e-04
	285424	100	3	1.04e-01	48	100	2.85e+01	3.16e-04
5000	304051	10	6	1.54e-02	46	10	1.12e+01	2.55e-04
	1249477	50	5	6.86e-02	55	50	8.20e+01	2.68e-04
	1984850	100	4	1.19e-01	53	100	2.01e+02	2.90e-04
10000	609765	10	6	1.51e-02	51	10	2.48e+01	2.05e-04
	2502872	50	5	6.64e-02	66	50	2.18e+02	1.96e-04
	3988660	100	4	1.14e-01	55	100	4.69e+02	3.70e-04
20000	1220822	10	6	1.49e-02	56	10	5.95e+01	2.02e-04
30000	1831551	10	6	1.49e-02	59	10	9.66e+01	3.00e-04
50000	3049294	10	6	1.48e-02	63	10	1.92e+02	1.59e-04
100000	6099941	10	6	1.47e-02	65	10	5.00e+02	1.68e-04

Table 6: Numerical results on semidefinite random matrix completion problems with noise. The noise factor nf is set to 0.1.

Unknown M					**Results**			
n	p	r	p/d_r	μ	iter	#sv	time	error
1000 /0.10	60497	10	6	1.59e-02	33	10	2.66e+00	2.85e-02
	195544	50	4	6.34e-02	39	50	1.01e+01	3.63e-02
	285424	100	3	1.04e-01	44	100	2.43e+01	4.36e-02
5000 /0.10	304051	10	6	1.54e-02	41	10	1.61e+01	2.87e-02

Table 6: Numerical results on semidefinite random matrix completion problems with noise. The noise factor nf is set to 0.1.

Unknown M					**Results**			
n	p	r	p/d_r	μ	iter	#sv	time	error
	1249477	50	5	6.86e-02	49	50	8.30e+01	3.22e-02
	1984850	100	4	1.19e-01	49	100	1.78e+02	3.79e-02
10000 /0.10	609765	10	6	1.51e-02	45	10	3.14e+01	2.87e-02
	2502872	50	5	6.64e-02	60	50	2.33e+02	3.25e-02
	3988660	100	4	1.14e-01	52	100	4.59e+02	3.84e-02
20000 /0.10	1220822	10	6	1.50e-02	50	10	7.98e+01	2.89e-02
30000 /0.10	1831551	10	6	1.49e-02	52	10	1.44e+02	2.90e-02
50000 /0.10	3049294	10	6	1.48e-02	56	10	3.50e+02	2.90e-02
100000 /0.10	6099941	10	6	1.47e-02	61	10	1.02e+03	2.89e-02

4.4 Numerical experiments on real matrix completion problems

In this section, we consider matrix completion problems based on some real data sets, namely, the Jester joke data set [23] and the MovieLens data set [25]. The Jester joke data set contains 4.1 million ratings for 100 jokes from 73421 users and is available on the website `http://www.ieor.berkeley.edu/~goldberg/jester-data/`. The whole data is stored in three excel files with the following characteristics.

(1) `jester-1`: 24983 users who have rated 36 or more jokes;
(2) `jester-2`: 23500 users who have rated 36 or more jokes;
(3) `jester-3`: 24938 users who have rated between 15 and 35 jokes.

We let `jester-all` be the data set obtained by combining all the above data sets.

For each data set, we let M be the original incomplete data matrix such that the i-th row of M corresponds to the ratings given by the i-th user on the jokes. For convenience, let Γ be the set of indices for which M_{ij} is given. We tested the jester joke data sets as follows. For each user, we randomly choose 10 ratings. Thus we select a subset Ω randomly from Γ, and hence a vector M_Ω which we take as the vector b in the problem (7).

Since some of the entries in M are missing, we cannot compute the relative error of the estimated matrix X as we did for the randomly generated matrices in section 4.2. Instead, we computed the Normalized Mean Absolute Error (NMAE) as in [23, 30]. The Mean Absolute Error (MAE) is defined as

$$\text{MAE} = \frac{1}{|\Gamma \backslash \Omega|} \sum_{(i,j) \in \Gamma \backslash \Omega} |M_{ij} - X_{ij}|, \tag{32}$$

where M_{ij} and X_{ij} are the original and computed ratings of joke j by user i, respectively. The normalized MAE is defined as

$$\text{NMAE} = \frac{\text{MAE}}{r_{\max} - r_{\min}}, \tag{33}$$

where $r_{\min}$ and $r_{\max}$ are lower and upper bounds for the ratings. For the jester joke data sets, all ratings are scaled to the range $[-10, +10]$, and we have $r_{\min} = -10$ and $r_{\max} = 10$.

The MovieLens data has 3 data sets (available from `http://www.grouplens.org/`) with the following characteristics.

(1) `movie-100K`: 100,000 ratings for 1682 movies by 943 users;
(2) `movie-1M`: 1 million ratings for 3900 movies by 6040 users;
(3) `movie-10M`: 10 million ratings for 10681 movies by 71567 users.

Here we have $r_{\min} = 1$ and $r_{\max} = 5$. In all the above data sets, each user has rated at least 20 movies. We note that some of the movies in the data sets are not rated by any of the users, and in our numerical experiments, those movies are removed.

For the MovieLens data sets, the matrices M are very sparse. In our experiments, we randomly select about 50% of the ratings given by each user to form the data vector b, i.e., $|\Omega|/|\Gamma| \approx 50\%$.

Table 7 reports the results for the APGL algorithm on the real matrix completion problems arising from the jester joke and MovieLens data sets. Here, we set the maximum number of iterations allowed in the APGL algorithm to 100. In the table, we report the regularization parameter μ, the number of iterations, the CPU time (in seconds), and the NMAE value. We also report #sv, $\sigma_{\max}$ and $\sigma_{\min}$, which are the rank, the largest and smallest positive singular values of the final solution matrix, respectively. From Table 7, observe that we can solve the matrix completion problem with dimension 73421×100 for the `jester-all` data set within 4 minutes with a relatively low NMAE. We can also solve the matrix completion problem with dimension 71567×10674 arising from the `moive-10M` data set in less than 4 minutes with an NMAE value of 1.64×10^{-1}.

Table 7: Numerical results on real data sets. In the table, $N := |\Gamma|$ denotes the total number of known ratings in M.

	m/n	N, $\lvert\Omega\rvert/N$	μ	iter	time	NMAE	#sv	$\sigma_{\max}/\sigma_{\min}$
jester-1	24983/ 100	1.81e+06, 13.80%	5.76e-01	50	7.15e+01	1.89e-01	79	3.85e+03/ 1.13e+01
jester-2	23500/ 100	1.71e+06, 13.75%	5.66e-01	50	6.86e+01	1.88e-01	79	3.84e+03/ 9.74e+00
jester-3	24938/ 100	6.17e+05, 40.42%	8.30e-01	47	6.24e+01	1.94e-01	78	2.79e+03/ 6.46e+00
jester-all	73421/ 100	4.14e+06, 17.75%	1.06e+00	51	2.18e+02	1.91e-01	79	5.95e+03/ 2.22e+01
moive-100K	943/ 1682	1.00e+05, 49.92%	3.21e-01	100	7.39e+00	2.05e-01	5	3.06e+03/ 6.70e+02
moive-1M	6040/ 3706	1.00e+06, 49.86%	9.47e-01	89	2.45e+01	1.76e-01	5	1.19e+04/ 1.76e+03
moive-10M	71567/ 10674	9.91e+06, 49.84%	2.66e+00	100	2.02e+02	1.64e-01	5	4.23e+04/ 9.37e+03

4.5 Numerical experiments on dimension reduction in multivariate linear regression

Recently Lu et al. [29] considered a nuclear norm regularized linear least squares problem (7) arising from simultaneous coefficient estimation and dimension reduction in multivariate linear regression (MLR). For this problem, the linear map $\mathcal{A}$ is given by $\mathcal{A}(X) = \mathrm{vec}(\Lambda X)$ for $X \in \Re^{m\times n}$ with $m > n$, and $\Lambda \in \Re^{m\times m}$ is a given positive definite diagonal matrix. Here, $\mathrm{vec}(M)$ denotes the vector obtained from the matrix M by stacking its columns sequentially. In [29], the authors proposed a variant of the Nesterov's smooth method [43] for solving smooth saddle point reformulations of the MLR problem.

In this section, we evaluate the performance of the APGL algorithm on randomly generated MLR problems. For a given m, set $n = m/2$ and $r = 50$, we generate the matrix Λ and vector b as follows:

$$\texttt{Λ} = \texttt{diag}(\texttt{rand}(\texttt{m},1)); \ \texttt{M} = \texttt{randn}(\texttt{m},\texttt{r}) * \texttt{randn}(\texttt{r},\texttt{n}); \ \texttt{R} = \texttt{randn}(\texttt{m},\texttt{n});$$

$$\texttt{b} = \texttt{vec}(\texttt{ΛM} + 0.01\|\texttt{ΛM}\|_{\texttt{F}}/\|\texttt{R}\|_{\texttt{F}}\ \texttt{R});$$

Table 8 reports the results for randomly generated MLR problems. For these problems, we set $\mu = 10^{-3}\|\mathcal{A}^*(b)\|_2$. In the table, we report the matrix dimension m/n, the mean value of the regularization parameter μ, the mean value of the number of iterations, the number (#sv) of positive singular values of the computed solution matrix, the final objective value attained for (7), the CPU time (in seconds), and the relative error (31) of 5 runs. In the table, we also report the results obtained by the codes created for the algorithm in [29]. Note that we stop the algorithm in [29] when the relative gap computed in the codes is less than 0.02. (This stopping criterion is much less stringent than the default.)

It is clear from Table 8 that the APGL algorithm substantially outperformed the algorithm in [29] in terms of the CPU time taken to solve the problems. The APGL algorithm also obtained smaller objective values for all the instances. But interestingly, the relative errors (31) in the solutions obtained by the algorithm in [29] are slightly better than those obtained by the APGL algorithm. We note the number of iterations taken by the algorithm in [29] can vary widely. For example, for the 5 random instances with dimension 200/100, the number of iterations vary from 144 to 2733.

Table 8: Numerical results for randomly generated multivariate linear regression problems.

Unknown M			**APGL**					**[29]**				
m/n	p	μ	iter	#sv	obj	time	error	iter	#sv	obj	time	error
200/100	20000	1.37e-01	91	50	8.689e+02	1.29e+01	1.66e-01	1066	100	8.698e+02	3.47e+01	1.60e-01
500/250	125000	2.67e-01	100	50	4.527e+03	2.51e+01	1.53e-01	1629	250	4.534e+03	6.29e+02	1.48e-01
1000/500	500000	4.74e-01	100	50	1.651e+04	8.39e+01	1.49e-01	2625	500	1.663e+04	7.57e+03	1.43e-01

4.6 Numerical experiments on regularized kernel estimation problems

In this section, we consider a regularized semidefinite linear least squares problem arising from regularized kernel estimation (RKE) [28]. In RKE, we are given a set of n objects and dissimilarity measures d_{ij} for certain object pairs $(i,j) \in \Omega$. Our goal is to estimate a positive semidefinite kernel matrix $X \in \mathcal{S}^n_+$ such that the fitted squared distances between objects induced by X satisfy

$$X_{ii} + X_{jj} - 2X_{ij} = \langle A_{ij},\, X\rangle \;\approx\; d_{ij}^2 \quad \forall\, (i,j) \in \Omega,$$

where $A_{ij} = (e_i - e_j)^T(e_i - e_j)$. Formally, one version of the RKE problem solves the following semidefinite linear least squares problem:

$$\min\Big\{ \sum_{(i,j)\in\Omega} W_{ij}(\langle A_{ij},\, X\rangle - d_{ij}^2)^2 + \mu\langle I,\, X\rangle \;:\; \langle E,\, X\rangle = 0,\; X \succeq 0\Big\}, \tag{34}$$

where $W_{ij} > 0$, $(i,j) \in \Omega$, are given weights, and μ is a positive regularization parameter. The linear equality in the above problem is a normalization constraint, and E is the matrix of all ones normalized to have unit Frobenius norm.

In this section, we apply the APGL algorithm to the problem (34) where the data d_{ij}, $(i,j) \in \Omega$, are protein sequences dissimilarities considered in [28]. The data are normalized to be in the interval $[0,1]$, and $\Omega = \{(i,j) \;:\; 1 \le i < j, 1 \le j \le 630\}$. In order to apply the APGL algorithm, we need to transform (34) into the form (10). In our case, we approximately enforce the linear equality constraint, $\langle E,\, X\rangle = 0$, in (34) by penalizing its violation in the objective function.

In [28], due to the computational limitation of the interior-point method used to solve (34), a subset of 280 globin proteins were selected from the entire set of 630 proteins. And for each of the selected proteins, 55 dissimilarities were randomly selected out of the total of 280. Here we are able to consider the entire set of $n = 630$ proteins and the dissimilarities among all the pairs of proteins. In our numerical experiments, we set $W_{ij} = 1$ for all $(i,j) \in \Omega$ and $\mu = 10^{-3}\|\mathcal{A}^*(b)\|_2$, where $\mathcal{A}$ is the linear map that maps a matrix X to the vector $[(\langle A_{ij},\, X\rangle :\; (i,j) \in \Omega)\,;\, \langle E,\, X\rangle]$. We set the maximum number of iterations allowed in the APGL algorithm to 100.

As mentioned in [28], the purpose of RKE is to perform clustering of proteins by representing each protein seqeunce as a point in the space $\Re^h$ for some appropriate dimension h so that clusters can subsequently be identified by checking the Euclidean distances between pairs of points. For visualization purpose, it is desirable to choose $h = 3$ by projecting the solution $\bar{X}$ in (34) to a 3D subspace. More specifically, consider the eigenvalue decomposition: $\bar{X} = Q\mathrm{diag}(\lambda)Q^T$. Let $Y_h = \mathrm{diag}(\sqrt{\lambda_1}, \ldots, \sqrt{\lambda_h})[Q_1, \ldots, Q_h]^T \in \Re^{h\times n}$, where $\lambda_1, \ldots, \lambda_h$ are the largest h eigenvalues of $\bar{X}$, and $Q_1, \ldots Q_h$ are their corresponding eigenvectors. Then $Y_h^T Y_h$ is the best rank-h approximation of $\bar{X}$ and each column of Y_h is a

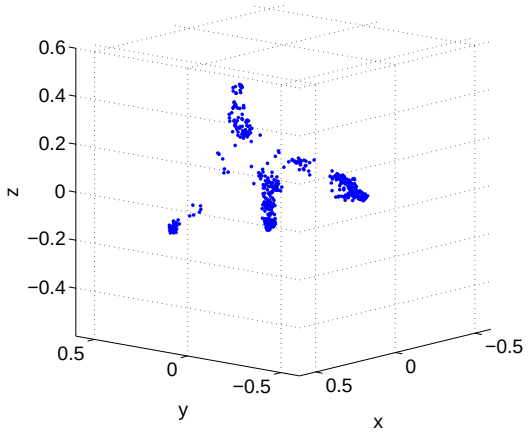

Figure 1: 3D representation of the sequence space for 630 proteins.

point in $\Re^h$ representing a protein sequence. For the choice $h = 3$, Figure 1 displays the 3D representation of the set of 630 proteins. It shows the first three coordinates of the protein sequence space, corresponding to three largest eigenvalues. The figure shows that there are at least 4 clusters in the set of 630 proteins. This finding is consistent with the observations reported in [28]. The numerical results for computing an approximate solution $\bar{X}$ of (34) using our APGL algorithm are reported in Table 9. For our computed solution $\bar{X}$, we have $\langle \bar{X}, E \rangle = 1.11 \times 10^{-12}$ and $\langle \bar{X}, I \rangle = 1.85 \times 10^2$.

Table 9: Numerical results on an RKE problem arising from protein clustering.

	n	p	μ	iter	time	#sv
RKE	630	198136	5.07e-01	100	7.36e+01	138

5 Conclusions

In this paper we have proposed an accelerated proximal gradient algorithm for solving the convex nonsmooth minimization problem on a set of real matrices, in particular, the nuclear norm regularized linear least squares problem arising in the applications such as the matrix completion, and presented its iteration complexity. This accelerated proximal gradient algorithm with a fast method, such as PROPACK, for computing partial singular value decomposition is simple and suitable for solving large-scale matrix completion problems

of the form (7) when the solution matrix has low-rank. Three techniques, linesearch-like, continuation, and truncation techniques, have been developed to accelerate the convergence of the original APG algorithm. From the numerical tests in section 4.1, we see that these techniques can accelerate the convergence of proximal gradient algorithms when applied to solve (7). Our numerical results suggest that our APGL algorithm is a promising algorithm for large-scale matrix completion problems, as well as other nuclear norm regularized linear least squares problems such as those arising from simultaneous coefficient estimation and dimension reduction in multivariate linear regression [45].

References

[1] J. Abernethy, F. Bach, T. Evgeniou and J.-P. Vert, Low-rank matrix factorization with attributes, Technical Report N24/06/MM, Ecole des Mines de Paris, 2006.

[2] ACM SICKDD and Netflix, Proceedings of KDD Cup and Workshop, 2007, Proceedings available online at `http://www.cs.uic.edu/~liub/KDD-cup-2007/proceedings.html`.

[3] Y. Amit, M. Fink, M. Srebro and S. Ullman, Uncovering shated structures in multiclass classification, in Proceedings of the Twenty-fourth International Conference on Machine Learning, 2007.

[4] A. Beck and M. Teboulle, A fast iterative shrinkage-thresholding algorithm for linear inverse problems, October 2008, to appear SIAM J. Imaging Sciences.

[5] D.P. Bertsekas, Nonlinear Programming, 2nd edition, Athena Scientific, Belmont, 1999.

[6] J.-F. Cai, E.J. Candés and Z. Shen, A singular value thresholding algorithm for matrix completion, September 2008.

[7] E.J. Candés and Y. Plan, Matrix completion with noise, preprint, Caltech, 2009.

[8] E.J. Candés and J. Romberg, Quantitative robust uncertainty principles and optimally sparse decompositions, Found. Comput. Math. 6 (2006), 227–254.

[9] E.J. Candés, J. Romberg and T. Tao, Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information, IEEE Trans. Info. Theory 52 (2006), 489–509.

[10] E.J. Candés and T. Tao, Nearly optimal signal recovery from random projections: Universal encoding strategies, IEEE Trans. Info. Theory 52 (2006), 5406–5425.

[11] E.J. Candés and B. Recht, Exact matrix completion via convex optimization, May 2008.

[12] S. Chen, D. Donoho and M. Saunders, Atomic decomposition by basis pursuit, SIAM J. Sci. Comput. 20 (1999), 33–61.

[13] I. Daubechies, M. De Friese and C. De Mol, An iterative thresholding algorithm for linear inverse problems with a sparsity constraint, Comm. on Pure and Applied Math. 57 (2004), 1413–1457.

[14] D. Donoho, Compressed sensing, IEEE Trans. Info. Theory 52 (2006), 1289–1306.

[15] D. Donoho and J. Tanner, Neighborliness of randomly projected simplices in high dimensions, Proc. Natl. Acad. Sci. USA 102 (2005), 9452–9457.

[16] P. Drineas, R. Kannan and M.W. Mahoney, Fast monte carlo algorithms for matrices ii: Computing low-rank approximations to a matrix, SIAM J. Comput. 36 (2006), 132–157.

[17] L. El Ghaoui and P. Gahinet, Rank minimization under LMI constraints: A framework for output feedback problems, In Proceedings of the European Control Conference, 1993.

[18] M. Fazel, H. Hindi and S. Boyd, A rank minimization heuristic with application to minimimum order system approximation, In Proceedings of the American Control Conference, 2001.

[19] M. Fazel, Matrix rank minimization with applications, PhD thesis, Stanford University, 2002.

[20] M. Figueiredo and R. Nowak, An EM algorithm for wavelet-bassed image restoration, IEEE Trans. Image Proc. 12 (2003), 906–916.

[21] M. Figueiredo and R. Nowak, A bound optimization approach to wavelet-based image deconvolution, IEEE Intern. Conf. on Image Processing-ICIP'05, 2005.

[22] M. Fukushima and H. Mine, A generalized proximal point algorithm for certain non-convex minimization problems, Int. J. Systems Sci. 12 (1981), 989–1000.

[23] K. Goldberg, T. Roeder, D. Gupta and C. Perkins, Eigentaste: A constant time collaborative filtering algorithm. Information Retrieval 4 (2001), 133–151.

[24] E.T. Hale, W. Yin and Y. Zhang, Fixed-point continuation for l1-minimization: Methodology and convergence, SIAM Journal on Optimization, 19 (2008), 1107–1130.

[25] J. Herlocker, J. Konstan, A. Borchers and J. Riedl, An algorithmic framework for performing collaborative filtering, In proceedings of the 1999 Conference on Research and Development in Information Retrieval, 1999.

[26] S. Ji and J. Ye, An accelerated gradient method for trace norm minimization, in Proceedings of the Twenty-Sixth International Conference on Machine Learning, 2009, pp. 457–464.

[27] R.M. Larsen, PROPACK - Software for large and sparse SVD calculations, Available from `http://sun.stanford.edu/~rmunk/PROPACK/`.

[28] F. Lu, S. Keles, S.J. Wright and G. Wahba, A framework for kernel regularization with application to protein clustering, Proceedings of the National Academy of Sciences 102 (2005), 12332–12337.

[29] Z. Lu, R.D.C. Monteiro and M. Yuan, Convex optimization methods for dimension reduction and coefficient estimation in multivariate linear regression, January 2008 (revised: March 2009), submitted to Math. Prog.

[30] S. Ma, D. Goldfarb and L. Chen, Fixed point and Bregman iterative methods for matrix rank minimization, October 2008.

[31] M. Mesbahi and G.P. Papavassilopoulos, On the rank minimization problem over a positive semidefinite linear matrix inequality, IEEE Transactions on Automatic Control 42 (1997), 239–243.

[32] A. Nemirovski and D. Yudin, Problem Complexity and Method Efficiency in Optimization, Wiley, New York, 1983.

[33] Y. Nesterov, A method of solving a convex programming problem with convergence rate $O(1/k^2)$, Soviet Mathematics Doklady 27 (1983), 372–376.

[34] Y. Nesterov, On an approach to the construction of optimal methods of minimization of smooth convex functions, Èkonom. i. Mat. Metody 24 (1988), 509–517.

[35] Y. Nesterov, Introductory Lectures on Convex Optimization, Kluwer Academic Publisher, Dordrecht, The Netherlands, 2004.

[36] Y. Nesterov, Smooth minimization of nonsmooth functions, Math. Prog. 103 (2005), 127–152.

[37] T.-K. Pong, P. Tseng, S. Ji and J. Ye, Trace norm regularization: Reformulations, Algorithms, and Multi-task Learning, Preprint, June 2009.

[38] B. Recht, M. Fazel and P. Parrilo, Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization, June 2007, to appear in SIAM Review.

[39] R.T. Rockafellar, Convex Analysis, Princeton University Press, Princeton, 1970.

[40] R. Tibshirani, Regression shrinkage and selection via the lasso, J. Royal Statist. Soc. B. 58 (1996), 267–288.

[41] K.C. Toh, M.J. Todd and R.H. Tütüncü, SDPT3 - a MATLAB software package for semedefinite programming, Optimization Methods and Software, 11 (1999), 545–581. Codes available at `http://www.math.nus.edu.sg/~mattohkc/sdpt3.html`.

[42] M.W. Trosset, Distance matrix completion by numerical optimization, Comput. Optim. Appl. 17 (2000), 11–22.

[43] P. Tseng, On accelerated proximal gradient methods for convex-concave optimization, May 2008, submitted to SIAM J. Optim.

[44] P. Tseng and S. Yun, A coordinate gradient descent method for nonsmooth separable minimization, Math. Prog. B. 117 (2009), 387–423.

[45] M. Yuan, A. Ekici, Z. Lu and R.D.C. Monteiro, Dimension reduction and coefficient estimation in multivariate linear regression, Journal of the Royal Statistical Society: Series B, 69 (2007), 329-346.

[46] S. Yun and K.C. Toh, A coordinate gradient descent method for ℓ_1-regularized convex minimization, to appear in Computational Optimization and Applications.

[47] S.J. Wright, R. Nowak and M. Figueiredo, Sparse reconstruction by separable approximation, IEEE Transactions on Signal Processing, 57 (2009) 2479–2493.