

Augmented Lagrangian alternating direction method for matrix separation based on low-rank factorization

Yuan Shen*

Zaiwen Wen[†]

Yin Zhang[‡]

January 11, 2011

Abstract

The matrix separation problem aims to separate a low-rank matrix and a sparse matrix from their sum. This problem has recently attracted considerable research attention due to its wide range of potential applications. Nuclear-norm minimization models have been proposed for matrix separation and proved to yield exact separations under suitable conditions. These models, however, typically require the calculation of a full or partial singular value decomposition (SVD) at every iteration that can become increasingly costly as matrix dimensions and rank grow. To improve scalability, in this paper we propose and investigate an alternative approach based on solving a non-convex, low-rank factorization model by an augmented Lagrangian alternating direction method. Numerical studies indicate that the effectiveness of the proposed model is limited to problems where the sparse matrix does not dominate the low-rank one in magnitude, though this limitation can be alleviated by certain data pre-processing techniques. On the other hand, extensive numerical results show that, within its applicability range, the proposed method in general has a much faster solution speed than nuclear-norm minimization algorithms, and often provides better recoverability.

Keywords. matrix separation, alternating direction method, augmented Lagrangian function.

1 Introduction

Recently, the low-rank and sparse matrix separation problem has attracted increasing research interest with potential applications in various areas such as statistics and system identifications [17, 18, 34]. The problem is to separate a low-rank matrix L^* and a sparse matrix S^* from their given sum $D \in \mathbb{R}^{m \times n}$, i.e.,

$$\text{Find a low-rank } L^* \in \mathbb{R}^{m \times n} \text{ and a sparse } S^* \in \mathbb{R}^{m \times n} \text{ such that } L^* + S^* = D. \quad (1)$$

We will call this problem *sparse matrix separation* (SMS) for brevity, omitting mentioning low-rankness since it is the assumed property (just like in *matrix completion* where low-rankness is omitted). It has been shown in [3, 9, 10, 21, 44] that, under some suitable conditions, the solution to problem (1) can be found by solving the convex optimization problem:

$$\min_{L, S \in \mathbb{R}^{m \times n}} \|L\|_* + \mu \|S\|_1 \quad \text{s.t.} \quad L + S = D, \quad (2)$$

*Department of Mathematics, Nanjing University, Nanjing, 210093, P. R. China.

[†]Department of Mathematics and Institute of Natural Sciences, Shanghai Jiao Tong University, Shanghai, 200240, P. R. China.

[‡]Department of Computational and Applied Mathematics, Rice University, Houston, Texas, 77005, U.S.A.

where $\mu > 0$ is a proper weighting factor, $\|L\|_*$ is the nuclear norm of L (the sum of its singular values) and $\|S\|_1$ is the sum of the absolute values of all entries of S (not the matrix 1-norm). Model (2) is called robust principal component analysis in [33, 44] or principal component pursuit in [21]. As is now well known, ℓ_1 -norm minimization has been used to recover sparse signals in compressive sensing (see, for example, [5, 6, 7, 13]), and nuclear-norm minimization has been used to recover a low-rank matrix from a subset of its entries in matrix completion (see, for example, [2, 42]).

Several algorithms have been developed to solve the convex model (2). One type of algorithms is based on a scheme called iterative shrinkage which has been successfully applied to ℓ_1 minimization ([12, 16, 19, 25], for example), and to matrix completion [2, 35]. Extensions of this basic shrinkage scheme have been devised to solve model (2) in [43, 44]. To accelerate the convergence of the basic shrinkage scheme, a projection gradient algorithm was presented in [33], utilizing Nesterov's acceleration approach [36, 39]. Also in [33], a gradient projection-like method was proposed to solve a dual problem of model (2).

Another type of algorithms is based on the classic augmented Lagrangian alternating direction method (ALADM or simply ADM) that minimizes the augmented Lagrangian function of model (2) with respect to one variable, either L or S , at a time while fixing the other at its latest value, and then updates the Lagrangian multiplier after each round of such alternating minimization. This ADM framework has been implemented by Yuan and Yang [46] in a code called LRSD (Low Rank and Sparse matrix Decomposition), and by Lin, Chen, Wu and Ma [32] in a code called IALM (Inexact Augmented Lagrangian Method); though due to implementation differences, LRSD and IALM often behave quite differently. In [32], an exact augmented Lagrangian method (EALM) is also implemented that runs multiple rounds of alternating minimization before the Lagrangian multiplier is updated.

The most expensive computational task required by nuclear-norm minimization algorithms is to perform singular value decomposition (SVD) at each iteration, which becomes increasingly costly as the matrix dimensions grow. Instead of doing a full SVD at each iteration, one may perform a partial SVD by only calculating the dominated singular values and vectors. When the involved ranks are relatively low, partial SVD schemes can be much faster than a full SVD one, though they do require the ability to reliably estimate the rank of the involved matrix. In the two ADM codes LRSD and IALM mentioned above, the former uses the full SVD while the latter uses a partial SVD scheme.

Even with partial SVD implementations, the scalability of nuclear-norm minimization algorithms are still limited by the computational complexity of SVD as matrix sizes and ranks both increases. To further improve the scalability of solving large-scale matrix separation problems, in this paper, we propose and study an alternative factor model and an algorithm for solving the model. As in [42, 49], we explicitly apply a low-rank matrix factorization form to L rather than minimizing the nuclear norm of L as in (2), avoiding SVD computation all together. As simple as the motivation is, a number of issues needs to be carefully addressed in order to make this approach work reliably and efficiently.

In practical applications, some entries of the data matrix D may be corrupted or missing (see a discussion in [3]). To accommodate this possibility, we consider a more general model than (1) in which only a subset of the entries in D are given. Let Ω be an index subset,

$$\Omega \subseteq \{(i, j) : 1 \leq i \leq m, 1 \leq j \leq n\}.$$

Let $\mathcal{P}_\Omega(M)$ be the projection of a matrix $M \in \mathbb{R}^{m \times n}$ onto the subspace of matrices whose nonzeros entries are restricted to Ω . That is, $[\mathcal{P}_\Omega(M)]_{ij} = 0, \forall (i, j) \notin \Omega$. Similarly, $\mathcal{P}_{\Omega^c}(M)$ is the projection to the complement subspace.

Given $\mathcal{P}_\Omega(D)$ and assuming that the rank of L does not exceed a prescribed estimate $k < \max(m, n)$, we first consider the model:

$$\min_{L, S \in \mathbb{R}^{m \times n}} \|\mathcal{P}_\Omega(S)\|_1 \quad \text{s.t.} \quad \mathcal{P}_\Omega(L + S) = \mathcal{P}_\Omega(D), \quad \text{rank}(L) \leq k. \quad (3)$$

Obviously, the low-rank matrix L can be expressed, non-uniquely, as a matrix product $L = UV$ where $U \in \mathbb{R}^{m \times k}$ and $V \in \mathbb{R}^{k \times n}$. For clarity, we will use the letter Z for the variable L . Replacing the constraint $\text{rank}(Z) \leq k$ by $Z = UV$, substituting $S = D - Z$ into (3) and simplifying, we arrive at a new model with three variables,

$$\min_{U, V, Z} \|\mathcal{P}_\Omega(Z - D)\|_1 \quad \text{s.t.} \quad UV - Z = 0, \quad (4)$$

where $U \in \mathbb{R}^{m \times k}$, $V \in \mathbb{R}^{k \times n}$ for some given but adjustable rank estimate $k > 0$, and $Z \in \mathbb{R}^{m \times n}$. In particular, problem (4) is called a partial-data model if Ω is a proper subset of the full index set; otherwise, it is called a full-data model.¹

The motivation for proposing the low-rank factorization model (4) is that it is hopefully much faster to solve this model than model (2). However, two potential drawbacks are apparent: (a) the non-convexity in the model may prevent one from getting a global solution, and (b) the approach requires an initial rank estimate k . In this paper, we present convincing evidences to show that (a) on a wide range of problems tested, the low-rank factorization model (4) is empirically as reliable as the nuclear norm minimization model (2); and (b) at least for “well-conditioned” low-rank matrices the initial rank estimate need not to be close to the exact rank.

Recently, the idea of replacing the rank constraint by the product of two low-rank matrices has been applied to matrix compressed sensing problem [24] and matrix completion problem [42]. In these works, the variables U and V appear only in the objective functions which are minimized either by an Gauss-Seidel-like scheme [24] or by an SOR-like scheme [42]. In the current paper, we devise an augmented Lagrangian alternating direction method (ALADM or ADM) to solve model (4). The approach of ADM has been widely used in solving convex optimization and related problems, including problems arising from partial differential equations (PDEs) [20, 23], variational inequality problems [26, 27], conic programming [14, 41], various nonlinear convex programming [1, 11, 28, 29, 38], zero-finding for maximal monotone operators [15], and ℓ_1 minimization arising from compressive sensing [40, 45, 48]. Recently, numerical results in [49] on nonnegative matrix factorization show that ADM can also perform well for solving certain non-convex models.

Our main contribution in this work is the development of a practically efficient solution method that offers much enhanced scalability in solving large-scale matrix separation problems, providing accelerations up to multiple orders of magnitude on some difficult instances. We present comprehensive numerical evidence to show the effectiveness of the proposed approach, and also identify and discuss weaknesses and limitations of the proposed approach.

This paper is organized as follows. An alternating direction method (ADM) for solving model (4) is introduced in Section 2.1. A preliminary convergence result for our algorithm is presented in Section 2.2 indicating that whenever the algorithm converges, it must converge to a stationary point. Section 3 describes the set-up for our numerical experiments. Extensive numerical results are presented in Section 4 comparing our algorithm with two state-of-the-art nuclear-norm minimization algorithms. We also discuss limitations of our approach in Section 4.

¹After completing the first version of this paper, we have found a very recent work [37] that extended the nuclear-norm minimization model (2) to include partial-data situations, and studied an augmented Lagrangian based algorithm.

2 Augmented Lagrangian Alternating Direction Algorithm

2.1 The algorithmic framework

We first present an augmented Lagrangian alternating direction method for solving (4). The augmented Lagrangian function of (4) is defined as

$$\mathcal{L}_\beta(U, V, Z, \Lambda) = \|\mathcal{P}_\Omega(Z - D)\|_1 + \langle \Lambda, UV - Z \rangle + \frac{\beta}{2} \|UV - Z\|_F^2, \quad (5)$$

where $\beta > 0$ is a penalty parameter and $\Lambda \in \mathbb{R}^{m \times n}$ is the Lagrange multiplier corresponding to the constraint $UV - Z = 0$, and $\langle X, Y \rangle$ denotes the usual inner product between matrices X and Y of equal sizes, i.e., $\langle X, Y \rangle = \sum_{i,j} X_{i,j} Y_{i,j}$. It is well-known that, starting from $\Lambda^0 = 0$, the classic augmented Lagrangian method solves

$$\min_{U, V, Z} \mathcal{L}_\beta(U, V, Z, \Lambda^j), \quad (6)$$

at the j -th iteration for $(U^{j+1}, V^{j+1}, Z^{j+1})$, then updates the multiplier Λ by the formula

$$\Lambda^{j+1} = \Lambda^j + \beta(U^{j+1}V^{j+1} - Z^{j+1}).$$

Since solving (6) for U , V and Z simultaneously can be difficult, following the idea in the classic alternating direction method for convex optimization [22, 23], we choose to minimize the augmented Lagrangian function with respect to each block variable U , V and Z one at a time while fixing the other two blocks at their latest values, and then update the Lagrange multiplier. Specifically, we follow the framework:

$$U^{j+1} = \operatorname{argmin}_{U \in \mathbb{R}^{m \times k}} \mathcal{L}_\beta(U, V^j, Z^j, \Lambda^j), \quad (7a)$$

$$V^{j+1} = \operatorname{argmin}_{V \in \mathbb{R}^{k \times n}} \mathcal{L}_\beta(U^{j+1}, V, Z^j, \Lambda^j), \quad (7b)$$

$$Z^{j+1} = \operatorname{argmin}_{Z \in \mathbb{R}^{m \times n}} \mathcal{L}_\beta(U^{j+1}, V^{j+1}, Z, \Lambda^j), \quad (7c)$$

$$\Lambda^{j+1} = \Lambda^j + \gamma\beta(U^{j+1}V^{j+1} - Z^{j+1}). \quad (7d)$$

where $\gamma > 0$ is a step-length (or relaxation when $\gamma > 1$) parameter.

For simplicity, we now temporally omit the superscripts in U^j and U^{j+1} , and instead denote the two by U and U_+ , respectively. Similar notation is used for the variables V , Z and Λ . The subproblems (7a) and (7b) are simple least squares problems, whose solutions are:

$$B = Z - \Lambda/\beta, \quad U_+ = BV^\dagger, \quad V_+ = U_+^\dagger B. \quad (8)$$

where $X^\dagger$ denotes the pseudo-inverse of the matrix X . The computational cost for solving the two linear least squares problems in (8) is relatively inexpensive whenever k is relatively small. In fact, only one least squares problem is necessary as pointed out in [42]. It follows from (8) that

$$U_+V_+ = U_+U_+^\dagger B = \mathcal{P}_{U_+}(B),$$

where $\mathcal{P}_{U_+}(B)$ is an orthogonal projection of B onto $\mathcal{R}(U_+)$, the range space of U_+ . It can be easily verified that $\mathcal{R}(U_+) = \mathcal{R}(BV^\top)$ and hence

$$U_+V_+ = \mathcal{P}_{BV^\top}(B) = QQ^\top B$$

where Q is an orthonormal basis for $\mathcal{R}(BV^\top)$, which can be computed, say, by a QR factorization. Hence, (8) can be replaced by

$$B = Z - \Lambda/\beta, \quad U_+ = \operatorname{orth}(BV^\top), \quad V_+ = U_+^\top B, \quad (9)$$

so that U_+V_+ remains the same. The solution Z_+ of the subproblem (7c) can also be explicitly computed from the well-known shrinkage (or soft-thresholding) formula, leading to

$$\mathcal{P}_\Omega(Z_+) = \mathcal{P}_\Omega\left(\mathcal{S}\left(U_+V_+ - D + \frac{\Lambda}{\beta}, \frac{1}{\beta}\right) + D\right) \quad \text{and} \quad \mathcal{P}_{\Omega^c}(Z_+) = \mathcal{P}_{\Omega^c}\left(U_+V_+ + \frac{\Lambda}{\beta}\right), \quad (10)$$

where $\mathcal{S}(x, \tau) := \text{sign}(x) \max(|x| - \tau, 0)$ for a scalar variable x and is applied component-wise to vector- or matrix-valued x . Clearly, when Ω is the full index set, (10) reduces to

$$Z_+ = \mathcal{S}\left(U_+V_+ - D + \frac{\Lambda}{\beta}, \frac{1}{\beta}\right) + D. \quad (11)$$

Summarizing the above description, we arrive at an augmented Lagrangian alternating direction method (ALADM) given in Algorithm 1, whose implementation is called **LMaFit** (Low-rank Matrix Fitting) [47]. Note that the code in [42] for matrix completion is also called **LMaFit** since they both employ the matrix factorization strategy.

Algorithm 1: LMaFit for sparse matrix separation

- 1 Input Ω , data $\mathcal{P}_\Omega(D)$, initial rank estimate k , and parameters β and γ .
 - 2 Set $j = 0$. Initialize $V^0 \in \mathbb{R}^{k \times n}$, and $Z^0, \Lambda^0 \in \mathbb{R}^{m \times n}$.
 - 3 **while** *not converge* **do**
 - 4 Compute U^{j+1} and V^{j+1} by (9), Z^{j+1} by (10), and Λ^{j+1} by (7d).
 - 5 Increment j , and possibly re-estimate k and adjust sizes of the iterates.
-

Remark 2.1 *It is important to note that although the augmented Lagrangian function (5) is not jointly convex in the pair (U, V) , it is convex with respect to either U or V while fixing the other. This property allows the ADM scheme to be well defined.*

2.2 Convergence issue

There is no established convergence theory, to the best of our knowledge, for ALADM algorithms applied to non-convex problems or even to convex problems with more than two blocks of variables as we have in Algorithm 1. On the other hand, empirical evidence suggests that Algorithm 1 have very strong convergence behavior. In this subsection, we give a weak convergence results for Algorithm 1 that under mild conditions any limit point of the iteration sequence generated by Algorithm 1 is a KKT point. Although far from being satisfactory, this result nevertheless provides an assurance for the behavior of the algorithm. Further theoretical studies in this direction are certainly desirable.

For simplicity, we will consider the full-data model (4) where Ω is the full index set. A similar result can be derived for the partial-data model as well. It is straightforward to derive the KKT conditions for (4):

$$\Lambda V^\top = 0, \quad U^\top \Lambda = 0, \quad UV - Z = 0, \quad \Lambda \in \partial_Z(\|Z - D\|_1),$$

where, for any $\beta > 0$, the last relation is equivalent to

$$Z - D + \frac{\Lambda}{\beta} \in \frac{1}{\beta} \partial_Z(\|Z - D\|_1) + Z - D \triangleq \mathcal{Q}_\beta(Z - D) \quad (12)$$

with the scalar function $\mathcal{Q}_\beta(t) \triangleq \frac{1}{\beta} \partial|t| + t$ applied element-wise to $Z - D$. It is easy to verify that $\mathcal{Q}_\beta(t)$ is monotone so that $\mathcal{Q}_\beta^{-1}(t) \triangleq \mathcal{S}(t, \frac{1}{\beta})$. Applying $\mathcal{Q}_\beta^{-1}(\cdot)$ to both sides of (12) and invoking the equation $Z = UV$, we arrive at

$$Z - D = \mathcal{Q}_\beta^{-1}(Z - D + \Lambda/\beta) \equiv \mathcal{S}(Z - D + \Lambda/\beta, 1/\beta) \equiv \mathcal{S}(UV - D + \Lambda/\beta, 1/\beta).$$

Therefore, the KKT conditions for (4) can be written as: for $\beta > 0$,

$$\Lambda V^\top = 0, \quad U^\top \Lambda = 0, \quad UV - Z = 0, \quad (13a)$$

$$Z - D = \mathcal{S}(UV - D + \Lambda/\beta, 1/\beta). \quad (13b)$$

Proposition 2.2 *Let $X \triangleq (U, V, Z, \Lambda)$ and $\{X^j\}_{j=1}^\infty$ be generated by Algorithm 1. Assume that $\{X^j\}_{j=1}^\infty$ is bounded and $\lim_{j \rightarrow \infty} (X^{j+1} - X^j) = 0$. Then any accumulation point of $\{X^j\}_{j=1}^\infty$ satisfies the KKT conditions (13). In particular, whenever $\{X^j\}_{j=1}^\infty$ converges, it converges to a KKT point of (4).*

Proof: It follows from (8), (10), and the identities $V^\dagger V V^\top \equiv V^\top$ and $U_+^\top U_+ U_+^\dagger \equiv U_+^\top$ that

$$\begin{aligned} (U^{j+1} - U^j) V^j (V^j)^\top &= \left(Z^j - \frac{\Lambda^j}{\beta} - U^j V^j \right) (V^j)^\top, \\ (U^{j+1})^\top U^{j+1} (V^{j+1} - V^j) &= (U^{j+1})^\top \left(Z^j - \frac{\Lambda^j}{\beta} - U^{j+1} V^j \right), \\ Z^{j+1} - Z^j &= \mathcal{S} \left(U^{j+1} V^{j+1} - D + \frac{\Lambda^j}{\beta}, \frac{1}{\beta} \right) + D - Z^j, \\ \Lambda^{j+1} - \Lambda^j &= \beta (U^{j+1} V^{j+1} - Z^{j+1}). \end{aligned} \quad (14)$$

Since $\{X^j\}_{j=1}^\infty$ is bounded by our assumption, the sequences $\{V^j (V^j)^\top\}_{j=1}^\infty$ and $\{(U^{j+1})^\top U^{j+1}\}_{j=1}^\infty$ are bounded. Hence $\lim_{j \rightarrow \infty} (X^{j+1} - X^j) = 0$ implies that both sides of (14) all tend to zero as j goes to infinity. Consequently,

$$U^j V^j - Z^j \rightarrow 0, \quad \Lambda^j (V^j)^\top \rightarrow 0, \quad (U^j)^\top \Lambda^j \rightarrow 0, \quad (15a)$$

$$\mathcal{S} \left(U^j V^j - D + \frac{\Lambda^j}{\beta}, \frac{1}{\beta} \right) + D - Z^j \rightarrow 0, \quad (15b)$$

where the first limit in (15a) is used to derive other limits. That is, the sequence $\{X^j\}_{j=1}^\infty$ asymptotically satisfies the KKT conditions (13), from which the conclusions of the proposition follow readily. This completes the proof.

2.3 A rank estimation technique

A good estimation for the rank of L^* , denoted by k^* , is essential to the success of the matrix separation model (4). Here we describe a rank estimation strategy, first proposed in [42], that utilizes the rank-revealing feature of QR factorization. Suppose that k^* is relatively small but not known exactly. We start from a relatively large estimate k so that $k \geq k^*$ and monitor the diagonal of the upper-triangular matrix in the QR factorization of $BV^\top$ in (9). Let $QR = (BV^\top)E$ be the economy-size QR factorization of U with a permutation matrix E so that the diagonal of R is non-increasing in magnitude. We then examine the diagonal elements of R , trying to detect a large drop in magnitude to near zero which gives a rank estimate. It should be clear that if the iterates converge to a well-conditioned low-rank solution, then rank-revealing QR factorization will eventually give a correct answer provided that a proper thresholding value is used in

detecting the drops in magnitude. Specifically, from the diagonal of R we compute two vectors $d \in \mathbb{R}^k$ and $r \in \mathbb{R}^{k-1}$:

$$d_i = |R_{ii}| \quad \text{and} \quad r_i = d_i/d_{i+1}, \quad (16)$$

where we set $0 = 0/0$, and then examine the quantity

$$\tau = \frac{(k-1)r(p)}{\sum_{i \neq p} r_i}, \quad (17)$$

where $r(p)$ is the maximum element of the vector r (with the largest index p if the maximum value is not unique). The value of τ represents how many times the maximum drop (occurring at the p -th element of d) is larger than the average of the rest of drops. In the current implementation, we reset the rank estimate k to p once $\tau > 2$, and this adjustment is done only once. This simple heuristics seemed to work sufficiently well on randomly generated problems in our tests where the correct drops could usually be detected after a very few iterations.

Rank estimation is an essential ingredient in algorithms designed to avoid full-matrix operations, including those using only partial SVDs. On the other hand, the effectiveness of rank estimation depends heavily on the existence of a clear-cut numerical rank, as well as on other properties of the matrices involved. In general, there exist no foolproof techniques for rank estimation. The above QR-based strategy is designed for well-conditioned, random-like, low-rank matrices that are not combined with sparse matrices of dominant magnitude in matrix separation problems.

3 Setup of Numerical Experiments

In this section, we describe the setup of our numerical experiments, including the solvers and rules used for comparison and the generation of random test problems. We denote the rank of the low-rank matrix L^* by k^* , the number of nonzero elements of sparse matrix S^* by $\|S^*\|_0$, and the “density” (or “sparsity”) of S^* by $\|S^*\|_0/mn$.

3.1 Solvers compared

We compared the performance of our solver **LMaFit** with either or both of the two recent solvers: LRSD (Low Rank and Sparse matrix Decomposition) by Yuan and Yang [46] and IALM (Inexact Augmented Lagrangian Method) by Lin, Chen, Wu and Ma [32] on various test problems. In fact, two other codes, the dual algorithms in [33] and the exact augmented Lagrangian method in [32], were also tested. Since our preliminary numerical results indicated that they generally were not as competitive as LRSD and IALM, we will only present comparison results involving LRSD and IALM. Most of our test problems are generated randomly, but some simple video background extraction problems are also used for the purpose of demonstration. All these tested codes were written in Matlab (though with some tasks done in C or Fortran). All numerical experiments were performed on a notebook computer with an Intel Core 2 Duo P8700 2.53GHz processor and 4G memory running Windows Vista basic edition and Matlab Release 7.9.0.

Although both LRSD and IALM are essentially the same algorithm, the two codes can behave quite differently due to differences in implementation. LRSD uses the subroutine “DGESVD” in the LAPACK library [?] to calculate full SVD through Matlab’s external interface. IALM uses a partial SVD implemented in the SVD package PROPACK [30]. The partial SVD scheme begins with a small rank estimate and increases it during iterations. This strategy can be significantly faster than full SVD when the rank involved is relatively

small so that only a small number of singular values and vectors are calculated at each iteration. On the other hand, the speed advantage diminishes as the rank increases, and at some point completely vanishes. LRSD and IALM also differ in the choices of the penalty parameter β in the augmented Lagrangian function. While the value of β is fixed in LRSD, IALM uses a sequence of β values, starting from a small value and increasing it after each iteration. By updating β values properly, the performance of IALM can be better than LRSD, especially when highly accurate solutions are required.

In our experiments, unless otherwise specified we used a set of default values for parameters required by the solvers. The weight μ in model (2) was set to $\mu = 1/\sqrt{m}$ for LRSD and IALM as is suggested in [3, 44]. All the other parameters in the codes LRSD and IALM were set to their default values. For **LMaFit**, the default value for the penalty parameter β was set to the reciprocal of the average of D -elements in absolute value, i.e., $\beta = 1/\text{mean}(|D|)$. Unless otherwise specified, in **LMaFit** the initial rank estimate k was set to $0.25m$ as the default. In some experiments, we employ a continuation strategy in **LMaFit**, similar to that in IALM, for the penalty parameter in which an increasing sequence of penalty parameter values is used, beginning with the initial value $\beta = 0.01/\text{mean}(|D|)$, multiplying β by a factor of $\rho = 1.5$ after each iteration until $\beta = 10^5/\text{mean}(|D|)$.

3.2 Comparison quantities and stopping rules

In many of our numerical experiments, we compared the performance of the involved algorithms by examining three quantities: the number of iterations denoted by “**iter**”, the CPU time in seconds denoted by “**CPU**”, and the relative error, denoted by “**relerr**” (or “**err**”), between the recovered low-rank matrix L and the exact one L^* , i.e.,

$$\text{relerr} := \frac{\|L - L^*\|_F^2}{\|L^*\|_F^2}. \quad (18)$$

For random problems, the reported **iter**, **CPU** and **relerr** (or **err**) are the mean values of corresponding results obtained on a number randomly generated matrices specified by the parameter **test_num**, which was set to 50 unless specified otherwise. Even though the relative error is not measurable in practice due to unavailability of L^* , it is one of the most informative quantities in evaluating algorithm’s performance.

The original stopping rule for the code LRSD is when the relative change

$$\text{relchg} := \frac{\|L^j - L^{j-1}\|_F^2}{\|L^{j-1}\|_F^2} \quad (19)$$

is below some prescribed tolerance. On the other hand, IALM is originally stopped when the relative error in separation, $\|L^j + S^j - D\|_F^2 / \|D\|_F^2$, becomes smaller than a given tolerance. However, to maintain consistency in comparison, we selectively used the following three stopping rules for all three codes compared:

$$(1) \text{ relerr} < \text{tol1}, \quad (2) \text{ relchg} < \text{tol2}, \quad (3) \text{ iter} < \text{maxit}$$

where **tol1** and **tol2** are prescribed tolerance values, and **maxit** is a prescribed maximum iteration number with the default value set to 100. We used rules (2) and (3) for comparing recoverability in which case rule (2) allows a code to stop even when it is unable to reach a meaningful accuracy. On the other hand, when comparing running speed, we used rules (1) and (3) because solution speed can be fairly compared only when the codes reach a similar accuracy. In some difficult cases, we do make an exception for IALM when it had trouble to reduce the relative error to a prescribed tolerance as other two codes could. In those type of tests, we also stopped IALM once $\|L + S - D\|_F / \|D\|_F < \text{tol3}$ where two default values were used for

the tolerance: $\text{tol3} = 10^{-3}$ in tests with noisy data and $\text{tol3} = 10^{-8}$ in tests with noiseless data. This rule was used on IALM only, and we call it rule (4).

3.3 Random problem generation

A random data matrix $D \in \mathbb{R}^{m \times n}$ is created from the following model

$$D = L^* + S^* + N,$$

where L^* is low-rank, S^* is sparse, and N represents additive white noise. First, random matrices $U \in \mathbb{R}^{m \times k^*}$ and $V \in \mathbb{R}^{k^* \times n}$ with i.i.d. standard Gaussian entries are generated, and then the rank- k^* matrix $L^* = UV$ is assembled. The sparse matrix $S^* \in \mathbb{R}^{m \times n}$ is constructed as

$$S^* = \sigma G, \tag{20}$$

where σ is a scalar and G is sparse and random. The nonzero entries of G are i.i.d. standard Gaussian whose locations are uniformly sampled at random. Moreover, another standard Gaussian noise matrix N is generated, and then scaled so that for some prescribed value $\delta \geq 0$, $\|N\|_F / \|L^*\|_F = \delta$.

Unless specified otherwise, our test problems were constructed using the following parameter settings: $m = n$, $\sigma = 0.01m$ and $\delta = \|N\|_F / \|L^*\|_F = 0.01$. Note that the parameter σ in (20) basically determines the magnitude of the sparse matrix S^* . We choose $\sigma = 0.01m$ because it ensures that L^* and S^* roughly have the same order of magnitudes, which seems to represent one of the most common scenarios in applications. To see this, in Figures 1 we present the average and maximum of the ratios between entries of L^* and S^* in absolute value, calculated from 100 random generated problems. From these figures, we can see that the entries of L^* and S^* are roughly in the same order.

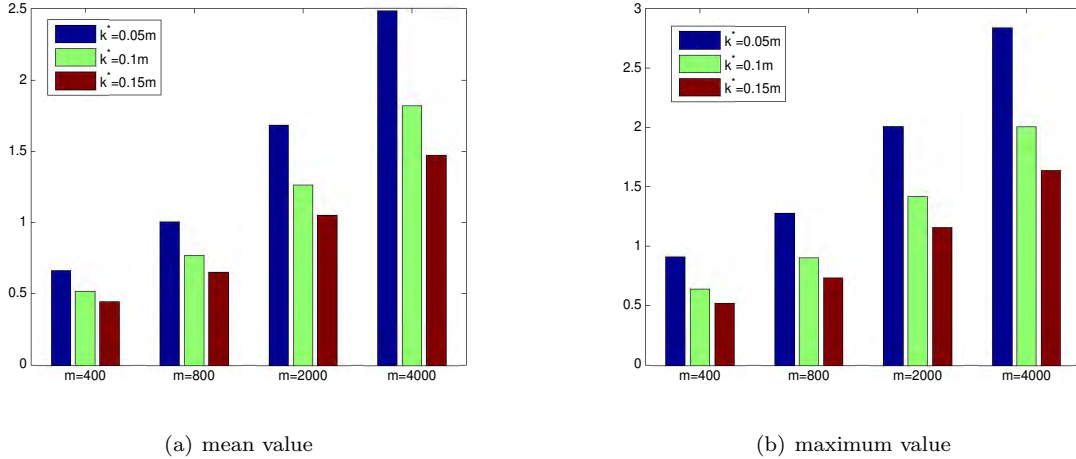


Figure 1: The mean (left) and maximum (right) ratios between entries of L^* and S^* in absolute value.

4 Numerical Results

We start by presenting a brief comparison that reveals major advantages and disadvantages of the proposed model compared to nuclear-norm minimization. We then provide further supporting evidence from compe-

hensive experiments on three types of test problems: (a) *fully random* test problems where both the low-rank and the sparse matrices are randomly generated; (b) *semi-random* test problems where only one of the two is random while the other is deterministic; and (c) *deterministic* test problems from video background extraction. In addition, we also demonstrate the use of model (4) to solve matrix completion problems where available partial data are corrupted by impulsive noise.

In our numerical experiments, we conducted numerical comparison with either IALM, LRSD, or both whenever appropriate. In those cases where only one of the two codes was used, we always chose the one with better or equal performance unless such a choice demanded excessive computer resource (as was for LRSD in a few cases).

4.1 A brief comparison

To get quick ideas on the performance of our algorithm in comparison to the nuclear-norm minimization code IALM [32], we ran **LMaFit** and IALM in two simple experiments where problems were generated by the procedure described in Section 3.3 without noise added. In each experiment, we changed only one parameter while fixing all others to observe the corresponding change in performance. More specifically, the parameter `test_num` (number of runs on random problems) was set to 5, the stopping rules (2) and (3) with `tol2` = 10^{-4} were applied to **LMaFit**, and stopping rules (3) and (4) with `tol3` = 10^{-4} were applied to IALM. The maximum iteration number was set to `maxit` = 200. In both experiments we set $m = n = 100$.

In the first experiment, we fixed the parameter pair $(k^*, \|S^*\|_0)$ to either $(5, 0.1mn)$ or $(10, 0.05mn)$, and varied the parameter σ in (20) from 1 to 15 to increase the magnitude of the sparse matrix S^* . The computational results are presented in Figures 2 (a) and (b). We can see that **LMaFit** failed to recover the solutions once σ is larger than 7 in Figures 2 (a) and 10 in Figures 2 (b). Incidentally, at these two threshold values, the maximum absolute value of the S^* -elements is slightly larger than twice of that of L^* -elements. This experiment suggests that **LMaFit** encounter difficulty when the sparse matrix dominates the low-rank one in magnitude. On the other hand, IALM seemed to have not shown such a sensitivity to the increase of σ , at least within the tested range, even though it did show some instability in Figure 2 (b) in terms of obtained accuracy.

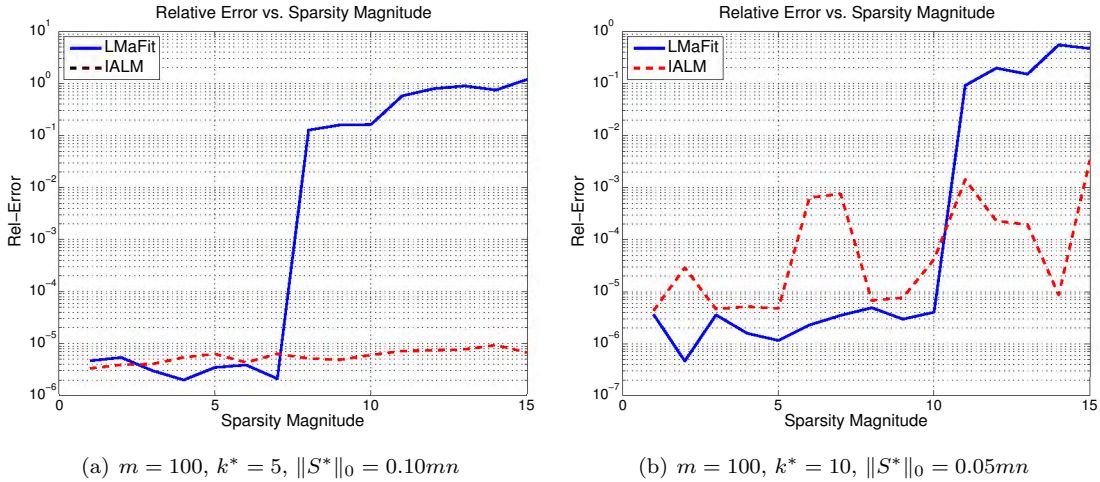


Figure 2: Relative error vs. sparse matrix magnitude.

In the second experiment we set $k^* = 5$, $\sigma = 0.01m = 1$ and varied the sparsity level $\|S^*\|_0/m^2$ from 1% to 40%. The relative errors and CPU seconds are presented in Figures 3. Results in Figures 3 (a)-(b) were generated from noiseless data with small tolerance, from which we see that **LMaFit** could recover accurate solutions for sparsity up to 24% while the relative error obtained by IALM started to deteriorate once sparsity is greater than 7%. In addition, the speed of **LMaFit** was about an order of magnitude faster than that of IALM in the range of interest where IALM still could maintain adequate accuracy. We did not run LRSD for this set of experiments since it would have required considerably more computing time. Results in Figure 3 (c)-(d) were for noisy data with $\delta = 0.01$. In these tests, we see that **LMaFit** clearly outperformed both IALM and LRSD in terms of both CPU time and relative error.

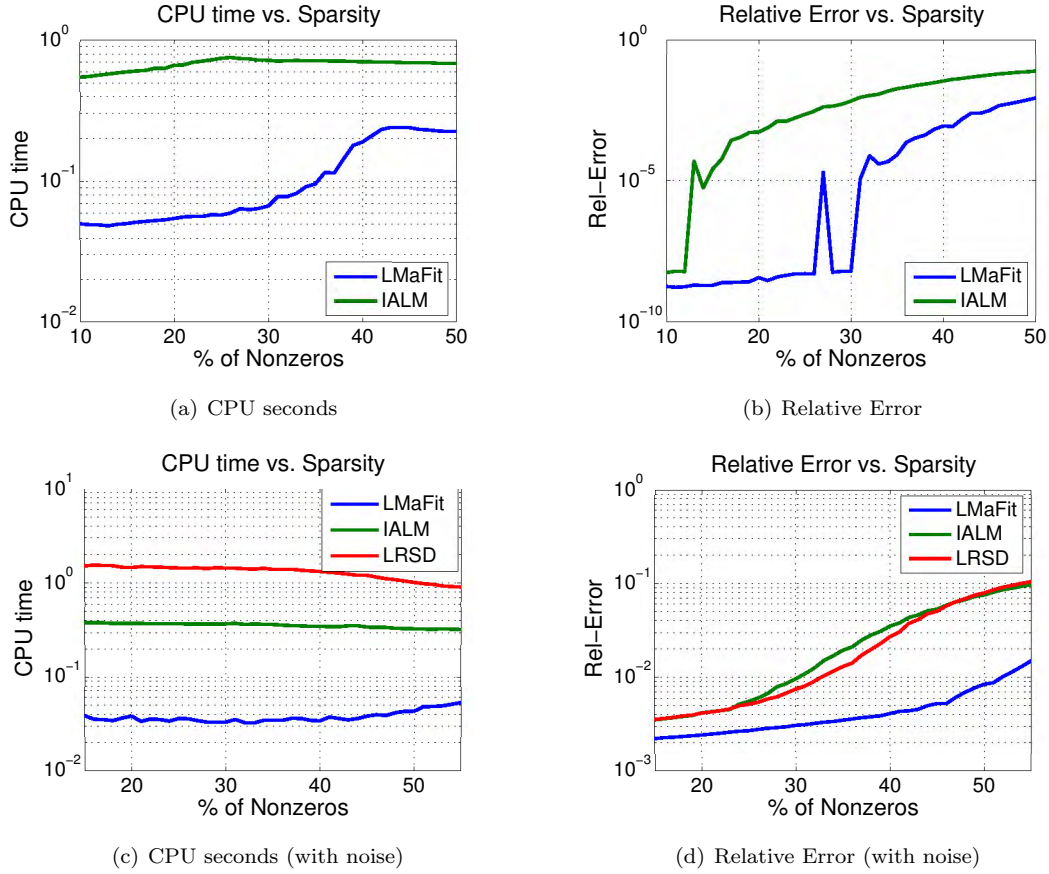


Figure 3: CPU time and relative error vs. sparsity for fixed $m = 100$ and $k^* = 5$

These two simple experiments suggest a few major properties about the proposed approach.

Disadvantage: The proposed approach appears to be unable to solve problems when the sparse matrix dominates the low-rank matrix in some measure of magnitude. The reason can be two-fold: (1) a large sparse matrix makes rank estimation difficult or impossible; (2) a large sparse matrix may force a part of it being shifted to the low-rank side in order to further reduce the objective.

Advantage I: When the sparse matrix is not the dominant part, the proposed approach exhibits better recoverability, being able to solve problems on which the nuclear-norm minimization approach fails. The reason for this phenomenon remains unclear, though should be model-related.

Advantage II: Thanks to its SVD-free feature, the proposed approach possesses a solution speed much faster than the nuclear-norm minimization approach, up to one to two orders of magnitude.

In the remainder of this section, we will present experimental results on different types of data matrices to provide more comprehensive evidence in support of the above observed advantages for our approach.

4.2 Results on fully random problems

In this subsection, we present numerical results using fully random data in which both the sparse and the low-rank matrices are randomly generated.

4.2.1 Recoverability

We first investigate the recoverability of **LMaFit** and **IALM** with respect to rank and sparsity. Although the recoverability of **LRSD** seems slightly better than that of **IALM**, we do not report the results of **LRSD** since it demanded too much computing time due to its use of the expensive full SVD. The test problems were generated with $m = 200$ and $\sigma = 0.01m = 2$. In order to observe the solvers' ability to achieve a high accuracy, we did not add Gaussian noise (*i.e.*, $\delta = 0$). We varied rank k^* from 2 and 120 with an increment 2 and sparsity from 1% to 60% with an increment 1%. For each pair of values of rank and sparsity, we ran 10 random problems with $\text{tol2} = 5 \times 10^{-8}$ for **LMaFit** and $\text{tol3} = 5 \times 10^{-8}$ for **IALM**. An initial rank estimate $k = 160$ was used for **LMaFit**. After 10 random runs, the geometric mean values of the relative errors are computed and reported in Figure 4 in terms of the digits of accuracy. For example, an error of 10^{-8} corresponds to 8 digits of accuracy. A relative error worse than 10^{-2} is shown in black, and a relative error better than 10^{-8} is shown in white; otherwise, it is shown in gray-scale corresponding to its actual value (between 2 and 8).

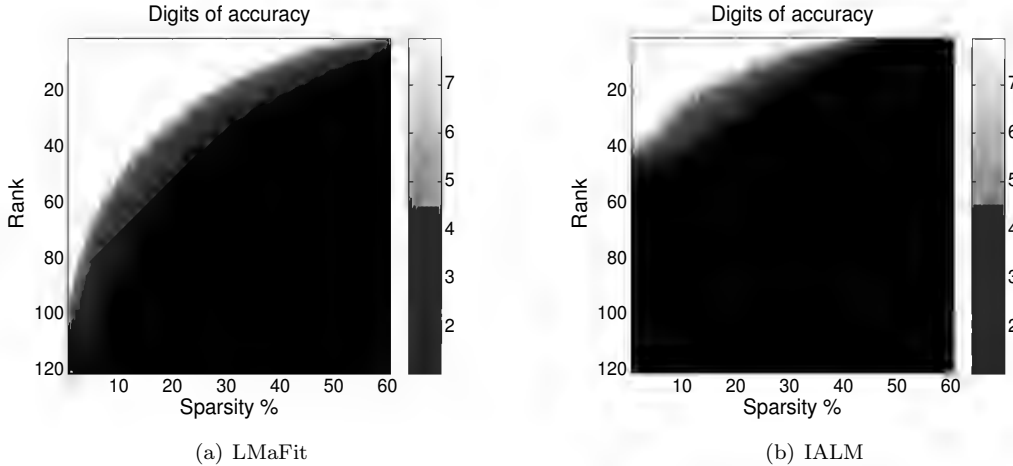


Figure 4: Recoverability phase plots for **LMaFit** and **IALM**

It is evident from the “recoverability phase plots” in Figure 4 that the recoverability of **LMaFit** is superior to that of **IALM**, which appears to suggest that model (4) may be better than the nuclear-norm minimization model (2) at least for some classes of problems.

4.2.2 Solution speed

To examine the convergence behavior of the codes, *in terms of iteration numbers rather than running time*, we ran the three codes on two noiseless random problems of sizes $m = n = 400$ for 200 iterations and record the progress of relative errors at all iterations. The random problems were generated with the default parameter values as is described in Section 3.3. The first problem has a relatively low rank at $k^* = 0.05m$ at a relatively low density level with $\|S^*\|_0 = 0.05m^2$; the second problem is a more difficult one with $k^* = 0.15m$ and $\|S^*\|_0 = 0.15m^2$. The results on these two problems are given in the first row of Figures 5 (a)-(b), respectively where the relative errors are plotted against the iteration number. As was mentioned in Section 3.1, a strategy of increasing β was used by both IALM and **LMaFit** that helped them to reach a high accuracy. Not using this “continuation” strategy on β , LRSD did not yield a comparable accuracy on this noiseless problem. Hence, we choose not to include the LRSD results in Figures 5 (a)-(b). From Figures 5 (a), we see that the convergence speeds of **LMaFit** and IALM are comparable on the first and easier problem. On the second and harder problem, however, as can be seen from Figures 5 (b), IALM could not reach as a high accuracy as **LMaFit** could, likely caused by over-aggressiveness in its rank-increasing scheme.

Adding Gaussian noise to the data matrix D , we then repeated the same experiment, but this time ran the three codes for 100 iterations only. The results are presented in Figures 5 (c)-(d) (in the second row). As can be seen, IALM and **LMaFit** converged slightly faster than LRSD in this case of noisy data where only a low accuracy was achievable.

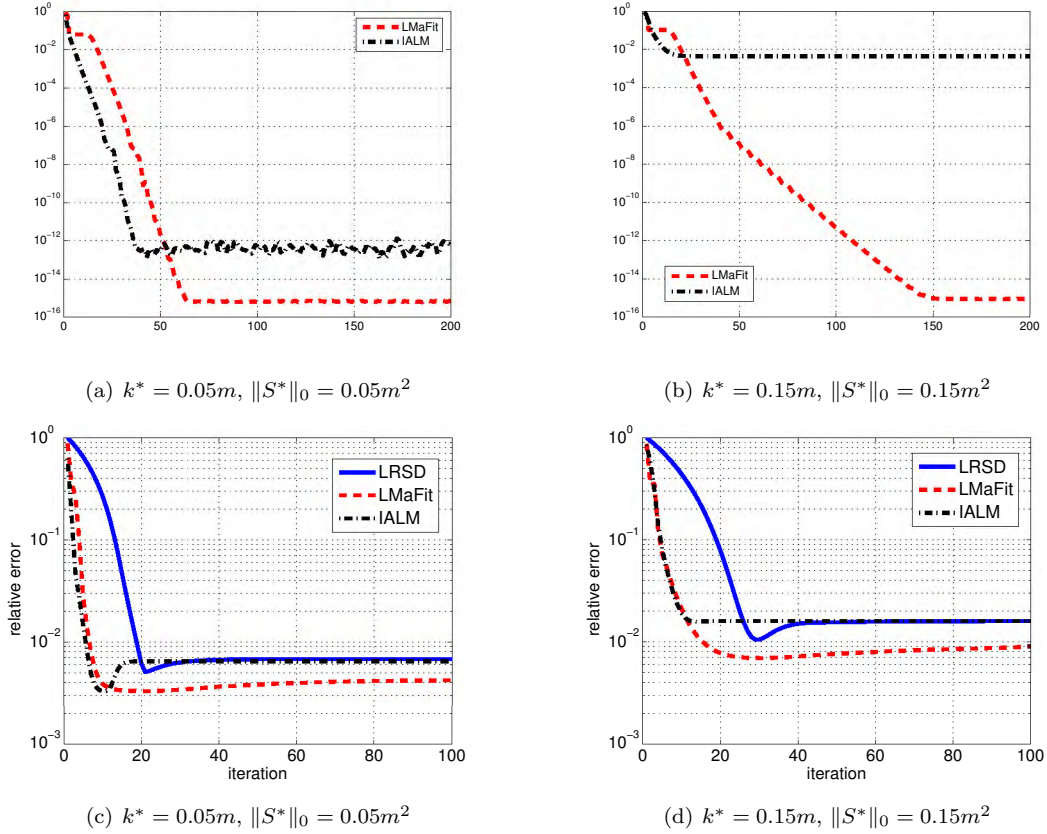


Figure 5: Iteration progress of relative error.

We emphasize again that so far convergence speed has been measured in terms of iteration number rather than running time. Next we present results on computing time on a set of experiments where the matrix dimension m ranges from 200 to 2000 with increment 100, and the stopping rules (1) and (3) were used. Figures 6 (a)-(b) give results of CPU time versus the dimension on two problems with $(k^*, \|S^*\|_0)$ set to $(0.05m, 0.05m^2)$ and $(0.15m, 0.15m^2)$, respectively, representing an easier and a harder problem.

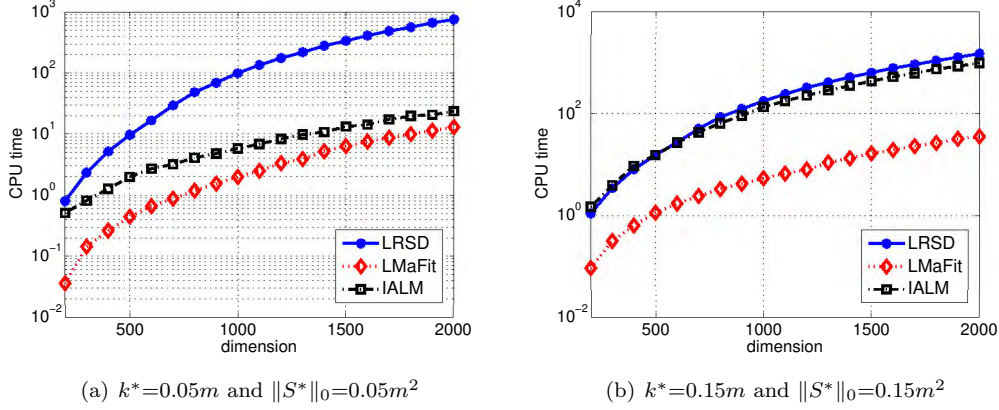


Figure 6: CPU time versus matrix dimension m .

By examining Figure 6 (a), we see that on this easier problem LRSD was much slower than both IALM and LMaFit, while IALM was slower than LMaFit, though the performance gap between the two appears to be narrowing as m increases. On the harder case depicted in Figure 6 (b), LRSD and IALM performed at very similar speeds, while LMaFit ran much faster than both for all matrix sizes with a speed advantage about one to two orders of magnitude.

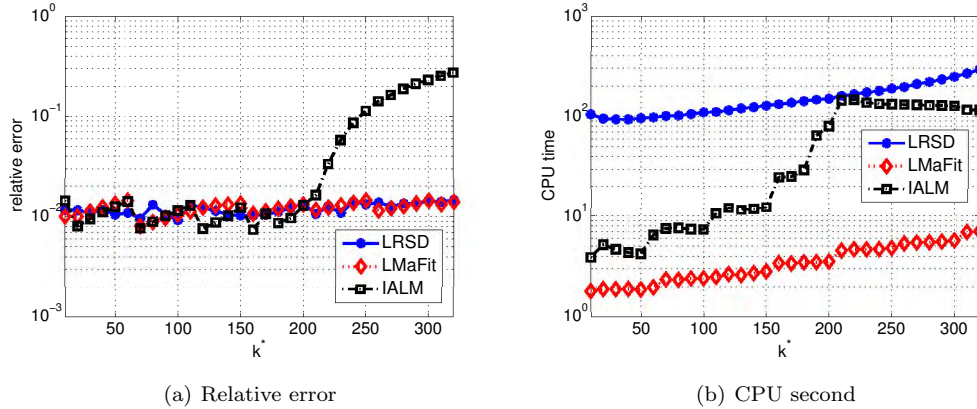


Figure 7: CPU time versus rank k^* ($m = 1000$ and $\|S^*\|_0 = 0.05m^2$).

In the next experiment, we take a look at the performance of the algorithms as rank k^* varies. We fixed the matrix dimension to $m = 1000$ and the sparsity level at $\|S^*\|_0 = 0.05m^2$, and then increased rank k^* from 10 to 320 with increment 10. The test results are depicted in Figure 7 where relative error and CPU time are plotted against rank k^* in plot (a) and (b), respectively. As is shown in the plot (a), the relative

error of IALM starts to deteriorate after $k^* > 200$ while those for both **LMaFit** and LRSD remain unchanged, indicating a lesser degree of recoverability for the tested version of IALM in this case. Plot (b) indicates that the performance gap between IALM and **LMaFit** in CPU time increases rapidly as k^* increases (up to the point when IALM starts to lose accuracy), while the gap between LRSD and **LMaFit** remains constant. This trend can be explained by the fact that as k^* increases, the cost of the partial SVD scheme quickly increases to a point where the partial SVD loses all its speed advantage over the full SVD.

To sum up, in Section 4.2 we have shown that, when the low-rank matrix is not dominated by the sparse one in magnitude, our code outperforms the two nuclear-norm minimization codes on fully random problems in terms of both recoverability and solution speed.

4.3 Results on semi-random problems

In practical applications, either the low-rank matrix or the sparse matrix in a given data matrix (sum of the two) may be more like deterministic than random. To test the performance of the codes in a more realistic setting, we consider some image separation problems in which one of the two matrices is an image. For such problems, since incoherent assumptions (see [3, 9, 44]) do not hold, there exists no theoretical guarantee of any kind for exact separations.

4.3.1 Deterministic sparse matrices

We first consider the case of recovering a deterministic sparse matrix (an image) from its sum with a low-rank random matrix. While the low-rank random matrices were constructed as before, with rank $k^* = 10$, by the procedure in section 3.3, the sparse matrices were from five images: “blobs”, “circles”, “phantom”, “text” and “rice” included in the Matlab Image Processing toolbox (“blob” was cropped slightly, and “phantom” was converted to black and white). As has been mentioned, our algorithm cannot handle the case where the magnitude of the sparse matrix dominant that of the low-rank one. To make the magnitude of the two roughly equal in our construction, we multiplied the sparse images by a scalar 0.1 before adding the low-rank matrices.

We ran **LMaFit** and IALM on these five semi-random problems without and with noise added. The stopping rules (2)-(3) with $\text{tol2} = 10^{-4}$ were used for **LMaFit**, and the rules (3)-(4) with $\text{tol3} = 10^{-4}$ were used for IALM. The recovered images are shown Figure 8 and the relative errors in Table 1, where the first two rows contain recovered sparse images without impulsive noise added, and the last two with impulsive noise of value 0.1 added to 10% pixels at random.

Table 1: Relative error in separating deterministic sparse matrices. The first 2 rows are for noiseless images, and the last two for noisy images with impulsive noise added to 10% pixels.

Solver \ Pattern	blob	circle	phantom	text	rice
LMaFit	6.579e-5	1.004e-4	8.528e-6	1.325e-4	1.113e-4
IALM	9.329e-1	9.741e-1	7.594e-1	7.830e-1	8.592e-1
LMaFit	9.348e-4	7.429e-4	6.516e-4	6.087e-4	6.134e-4
IALM	7.620e-1	8.725e-1	5.651e-1	5.505e-1	7.795e-1

As can be seen clearly from both Figure 8 and Table 1, **LMaFit** recovered the sparse images with much

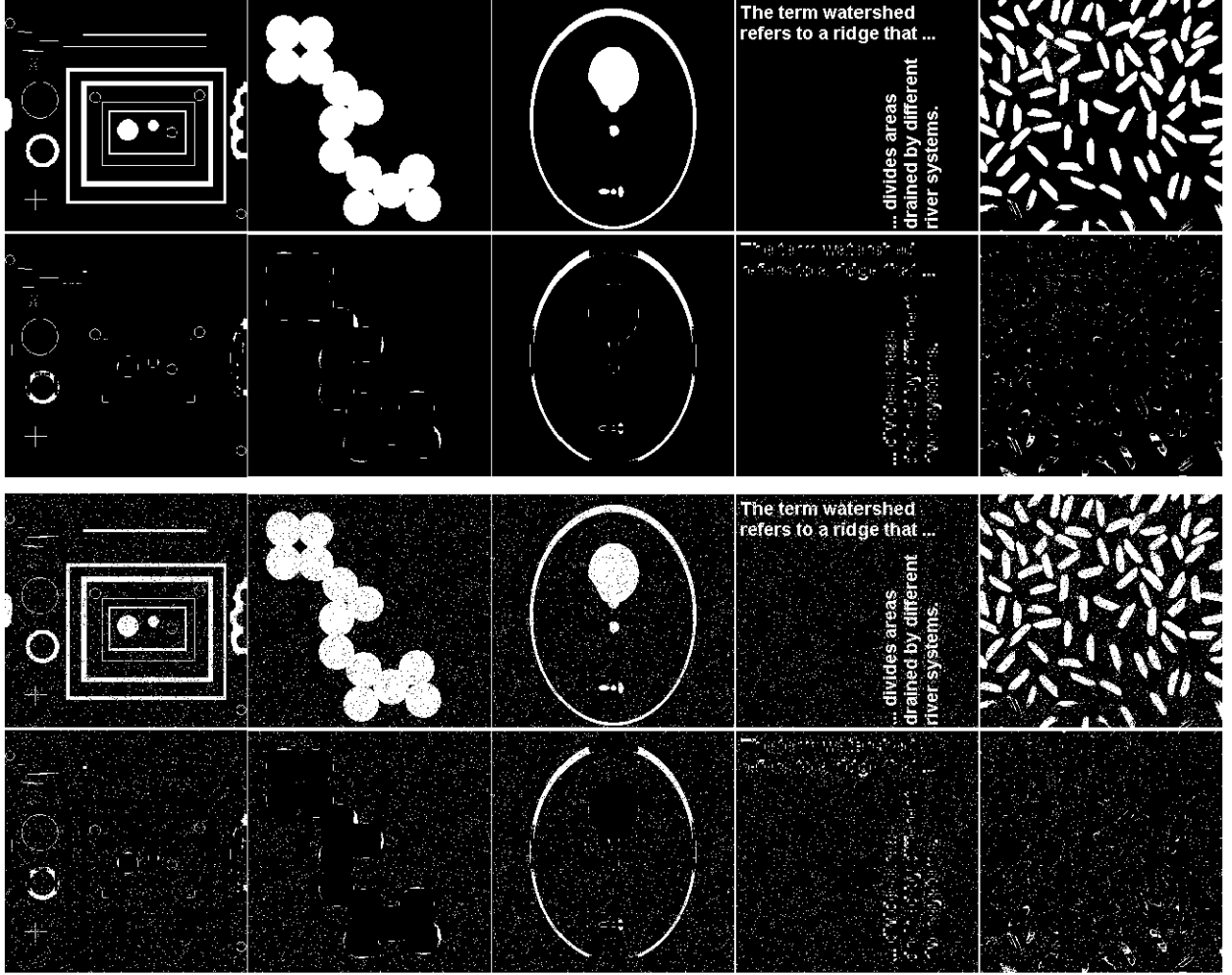


Figure 8: Recovered sparse images by LMaFit (the first and third rows) and IALM (the second and forth rows). In the last two rows, impulsive noise is added to 10% pixels at random in each sparse image.

higher accuracies than IALM did, confirming the recoverability advantage of our approach in the case of deterministic sparse matrices as long as their magnitude does not overwhelm that of low-rank matrices.

4.3.2 Deterministic low-rank matrices

Now we consider the opposite situation where the low-rank matrix is deterministic while the sparse matrix being random. We used two images for the low-rank matrices: “checkerboard” is exactly rank-2; “brickwall” is only approximately low-rank.

Our first test is on the 256×256 “checkerboard” image (see Figure 9) which is of rank 2. We corrupted the image by adding a sparse matrix whose nonzero entries take random values uniformly distributed in $[0, 1]$ and the locations of the nonzero entries were sampled uniformly at random. The sparsity level $\|S^*\|_0/m^2$ ranged from 0.05 to 0.8 with increment 0.05. We compared LMaFit with LRSD on this set of problems. In LMaFit, the initial rank was set to $k = 10$ and the penalty parameter set to $\beta = 10$. In LRSD, we used the parameter value $\mu = 0.25/\sqrt{m}$ instead of the default value $\mu = 1/\sqrt{m}$ since the former yielded a rank-2

matrix solution while the former did not. The stopping rules (1)-(3) with $\text{tol1} = 10^{-4}$ and $\text{tol2} = 5 \times 10^{-4}$ were used. All other parameters were set to their default values.

The results for this test set are depicted in Figure 9, where the first image corresponds to the sum of the checkerboard image and an impulsive noise image of $\|S^*\|_0/m^2 = 40\%$, the second image was recovered by LRSD and the third by LMaFit which clearly has a better quality. A more complete picture for this test can be seen from the last picture in Figure 9 where relative error corresponding to all tested sparsity levels are plotted. As can be seen, the recovery quality of LRSD started to deteriorate after $\|S^*\|_0/m^2 > 20\%$, while that of LMaFit remains good until $\|S^*\|_0/m^2 > 35\%$. If we say that a relative error of 10^{-2} is acceptable, then the acceptable level of impulsive noise is about 26% for LRSD and 44% for LMaFit.

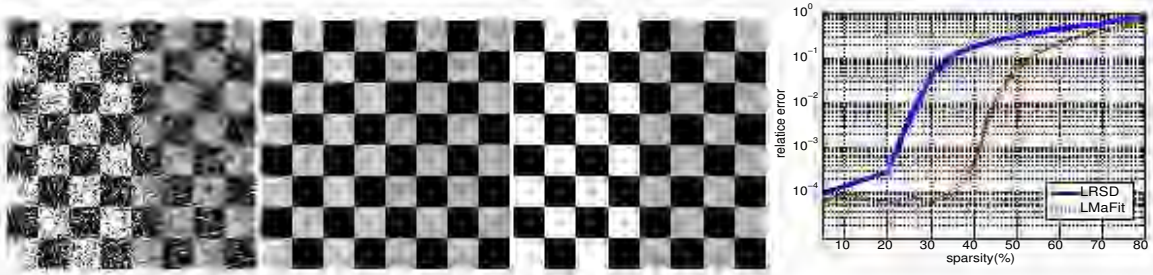


Figure 9: Recovery results for “checkerboard”. From left to right: (1) corrupted image by 40% impulsive noise, (2) recovered by LRSD, (3) recovered by LMaFit, (4) relative errors vs. noise level.

To construct a more challenging test, we replaced the image “checkerboard” by an image called “brick-wall”.² The matrix for this image is not really low-rank, has only a relatively small number of “dominant” singular values, and hence can be considered as an approximately low-rank matrix. This type of problems is much more difficult due to the lack of a clear-cut numerical rank and how to deal with them is of practical importance since they are perhaps more representative of real-world applications.

We compared LMaFit and IALM on this problem. For LMaFit, we fixed $k = 50$ without using rank estimation, and for IALM we used its default setting except stopping rules. The stopping rules for LMaFit were (2)-(3) with $\text{tol2} = 0.1$, for IALM were (3)-(4) with $\text{tol3} = 0.2$, while $\text{maxit} = 30$ for both. These settings were adopted after a few trial-and-error runs to ensure a sufficient and attainable accuracy.

The results for “brickwall” are presented in Figure 10 and Table 2. As can be seen Figure 10, the two codes recovered images that are visually comparable in quality. However, the computing time used by LMaFit was several times shorter than that used by IALM, as can be seen from Table 2.

Algorithm	iteration	CPU second	solution rank	relative error
LMaFit	10	2.527	50	0.2324
IALM	9	8.705	92	0.2110

Table 2: Numerical results for the “brickwall” image

Remark 4.1 We have shown that on random-like matrices with well-conditioned and a clear-cut numerical rank, rank estimation required by model (4) can be easily and reliably done. However, on more realistic,

²downloaded from http://upload.wikimedia.org/wikipedia/commons/1/14/Background_brick_wall.jpg.

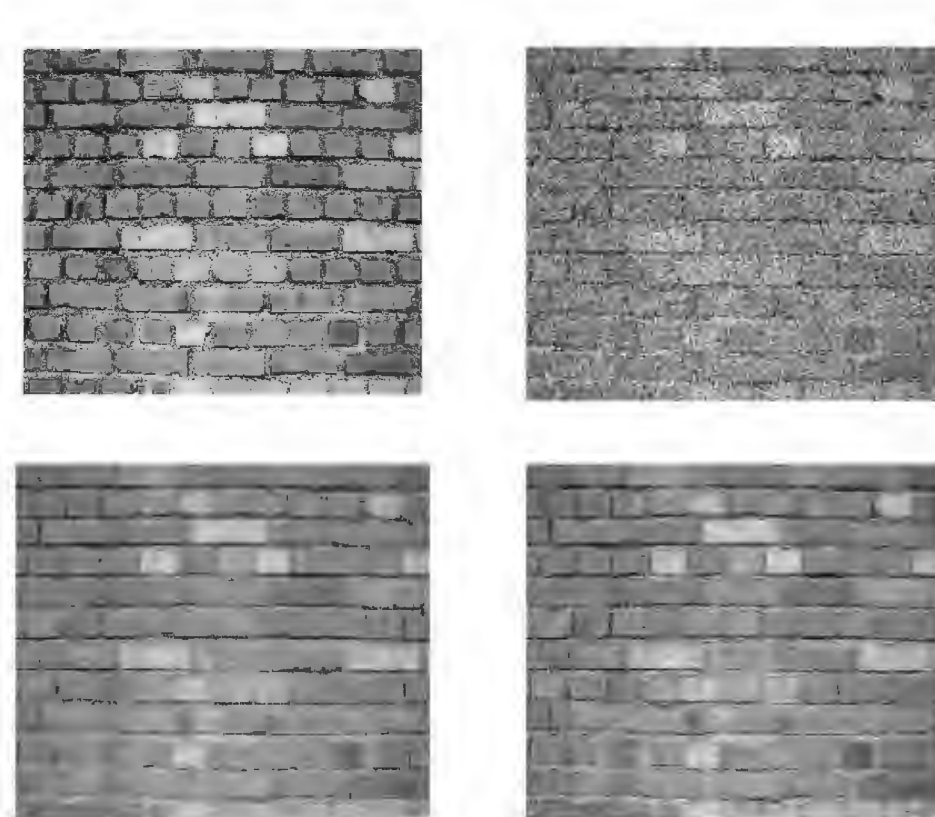


Figure 10: Results on “brickwall”: upper left: original image; upper right: corrupted image; lower left: recovered by IALM; lower right: recovered by LMaFit.

“approximately low-rank” matrices without a clear-cut numerical rank, rank estimation becomes extremely difficult or even impossible. Hence, trial-and-errors become necessary. In fact, the situation is similar for the nuclear-norm minimization model (2) which requires a proper choice of the parameter μ . Our computational experience indicates that for “approximately low-rank” matrices, the μ value can greatly affect the rank of returned solutions by either IALM or LRSD, making trial-and-errors necessary as is for the choice of k in LMaFit. In a sense, one may argue that choosing a good μ -value for model (2) is more difficult than guessing a rank estimate k for model (4) since in (2) the dependence of the solution rank on μ appears highly nonlinear.

4.4 Results on non-random problems

We now consider the video background extraction problem in [3], which aims to separate a video into two parts: a background and moving objects. Traditional algorithms for this problem often require a good knowledge about the background and can be quite complicated (see, for example [31]). For simplicity, we use videos from security cameras in which the background is relatively static (*i.e.*, the background is roughly the same in every frame) and hence can be viewed approximately as a low-rank matrix (typically rank 3 for color videos). The moving objects such as pedestrians and cars usually only occupy a small part of the picture, thus together can be regarded as a sparse matrix. As such, separation of the background and moving objects in a video can

be modeled as our a matrix separation problem. Our test procedure is similar to that given in [3], and the video clips were downloaded from http://perception.i2r.a-star.edu.sg/bk_model/bk_index.html.

Unlike the experiments in [3] where only gray-scale video clips were used, we tested both gray-scale and color videos. Since all the original videos are color, we converted them into gray-scale by using the Matlab command `rgb2gray`. We reshaped every frame of a video in every channel into a long column vector and then collected all the columns into a matrix. As such, the number of rows in a test matrix equals to the number of pixels and the number of columns equals to three times the number of frames in a video. For example, a color (3 channels) video with 300 frames of 320×240 resolution is converted into a 76800×900 matrix. We ran **LMaFit** and **IALM** on five video background extraction problems (we skipped LRSD since it would take excessive amounts of time due to full SVD calculations). Since all these video clips have more than 1000 frames, we took a part of each clip with 300 frames or less to reduce the storage and computation required.

For **IALM**, we set $\mu = 0.5/\sqrt{m}$ as suggested by [3]. For **LMaFit**, we fixed $k = 1$ for gray-scale videos and $k = 3$ for color ones. We stopped each code after 10 iterations because in our trials no further improvements were visually observable after that. A summary of the test problems and results is presented in Table 3. In Table 3, “Resolution” denotes the size of each frame in a video clip, and “No. frames” denotes the number of frames tested. Video names followed by (c) were tested as color videos, otherwise they were tested as gray-scale videos. The CPU time of **IALM** and **LMaFit** are reported in the last two columns of Table 3.

Video	Resolution	No. frames	IALM	LMaFit
hall	176×144	200	13.6969	4.7580
lobby	160×128	250	14.3209	4.8984
bootstrap	160×120	300	19.9213	5.4912
fountain(c)	160×128	150	27.9242	9.5785
bootstrap(c)	160×128	150	28.5170	8.5177

Table 3: Video separation problem statistics and CPU time used.

Both codes worked quite well on these video separation problems with indistinguishable quality by naked eyes, as can be visualized from Figure 11, where several frames of the original videos and computed separation results are presented for the video clips “hall” and “bootstrap(c)”. On these problems of extremely low ranks, the partial SVD technique used in **IALM** becomes quite effective. Even so, the CPU times required by **IALM** are still about three times of those required by **LMaFit**.

4.5 Matrix completion with corrupted data

In this section, we show that our model and algorithm can be applied to matrix completion problems with impulsive noise in available data. The standard matrix completion problem is to find a low-rank matrix from a subset of observed entries (see, for example, [2, 4, 8, 42]). Although, existing matrix completion models can tolerate small white noise, they are not equipped to deal with the more challenging problem where a part of the observed data is corrupted by large impulsive noise. Unlike model (2), model (4) is applicable to this case where the index set Ω consists of all known entries including the corrupted ones.

To construct random test problems, we first generate a low-rank matrix with rank k^* , then take a fraction of the entries uniformly at random as the “observed data” with a corresponding index set Ω . The quantity $\text{SR} = |\Omega|/mn$ is called the *sampling ratio*. For impulsive noise, we add a sparse matrix S^* , similarly

constructed as in (20), whose support is a subset of Ω . The quantity $\text{NR} = \|S^*\|_0/mn$ is called the *noise ratio*. In this experiments, test problems were generated with $m = n = 1000$ and $\sigma = 0.01m$. We tested two sets of problems corresponding to $\text{NR} = 0.1 \text{ SR}$ and $\text{NR} = 0.2 \text{ SR}$, respectively. In each test set, the sampling ratio took four values: $\text{SR} = 0.05, 0.1, 0.2$ and 0.4 . Then for each of the eight (SR, NR) pairs, we increased the rank k^* from 2 until the solver encountered a failure.

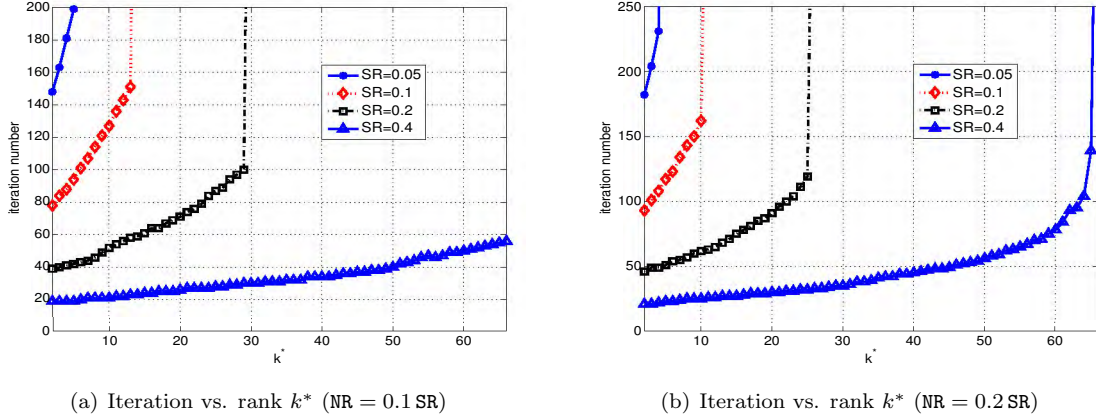


Figure 12: The performance of LMaFit on matrix completion problems with impulsive noise.

Figure 12 contains numerical results for running LMaFit under the above setting with default parameter values, except that the maximum number of iteration were set to `maxit` = 200 for the easier case $\text{NR} = 0.1 \text{ SR}$, and `maxit` = 250 for the harder case $\text{NR} = 0.2 \text{ SR}$. In Figure 12, the average number of iterations taken by LMaFit was plotted for each test instance after multiple runs, where an iteration number less than the maximum allowed indicates that LMaFit had “converged” with an accuracy less than the prescribed tolerance for all runs, and a near-vertical line indicates that LMaFit encountered a failed run from that point on. The results show that LMaFit is capable of solving these matrix completion problems with impulsive noise as long as a sufficient number of accurate sample entries is available.

4.6 Treatments of sparse matrices of excessive magnitude

We have shown that LMaFit is not able to solve problems in which the magnitude of the sparse matrix S^* dominates that of L^* . However, for these problems, it is often easy to detect those large elements of S^* and somehow fix the problem. In this section, we introduce two “fixes” or remedial techniques. One is to truncate the detected large entries of S^* , another is to exclude them from the index set Ω . The truncation technique depends on a properly chosen threshold value ν so that any element of D with $|D_{ij}| > \nu$ implies that $S_{ij}^* \neq 0$. In this case, we can replace the data matrix D by its projection $\mathcal{P}_{[-\nu, \nu]}(D)$ in model (4) and still obtain the correct low-rank solution, where the projection onto the interval $[-\nu, \nu]$ is done element-wise (in some cases it may be necessary to choose an unsymmetrical interval). Obviously, it is important to be able to find a proper threshold ν . For random matrices, we can sort the magnitudes of entries of D in descending order, locate a “turning point” and use the magnitude at this point for ν . An example for this idea is given in Figure 13 for a random data matrix of sizes $m = n = 400$. Instead of truncating large entries in D , we can also remove those large elements from D and solve a partial-data model for the index subset Ω consisting of the remaining indices, which is the second remedial technique tested.

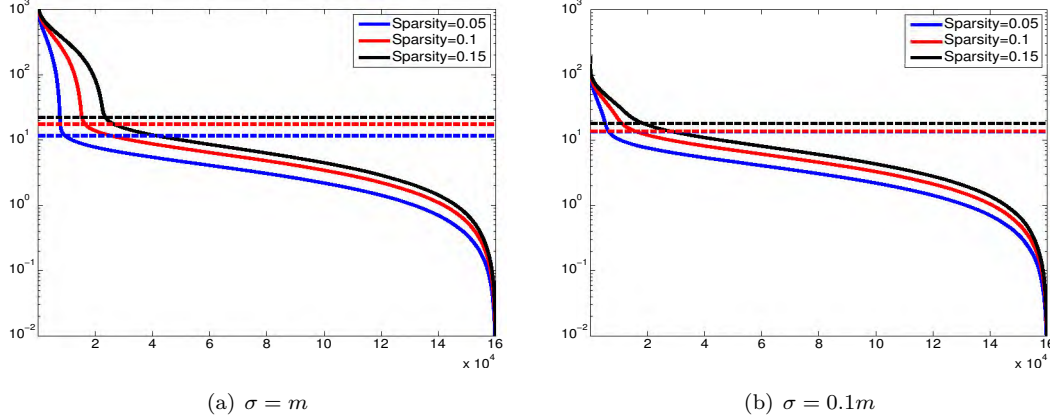


Figure 13: An example of choosing threshold in truncation. Each curve represents the sorted absolute values of all entries in D ; horizontal lines represent threshold values ν .

We tested the above two techniques in a set of experiments on fully random problems. We constructed test problems using $m = 400$, $\sigma = 400$, $k^* = 20$, and varying the “noise ratio” $\text{NR} = \|S^*\|_0/m^2$ from 0.01 to 0.80 with increment 0.01. For clarity, we will call LMaFit with the truncation technique “LMaFit w/trun”, and LMaFit applied to a partial-data model “PLMaFit”, and compared the two with the nuclear-norm minimization code LRSD since it exhibited a stronger recoverability than IALM in this set of tests. The test results are presented in Figure 14.

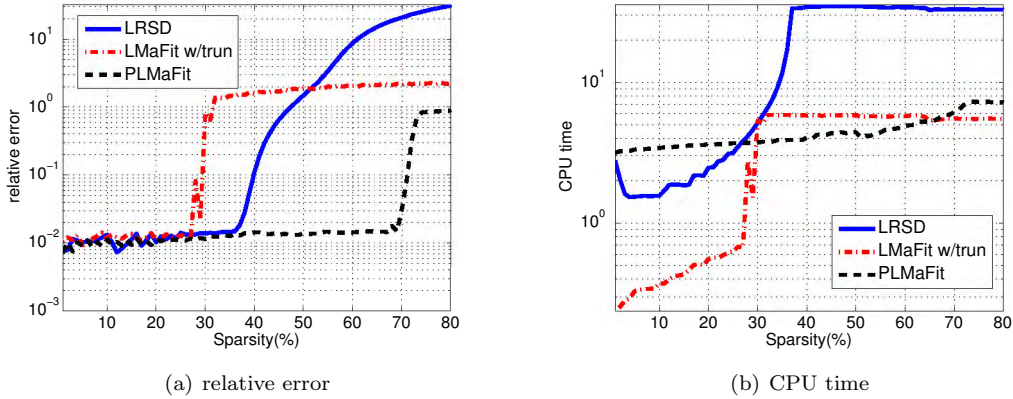


Figure 14: Results on random problems with large magnitude S^* ($\sigma = 400$)

Since $\sigma = 400$ used in this dataset makes S^* really large, LMaFit always failed without a remedial technique. On the other hand, the two remedial techniques enabled LMaFit to solve many of the tested problems at reasonable sparsity levels, as can be seen from Figure 14 (a). Although “LMaFit w/trun” is much faster than the other two algorithms, its ability is still limited in handling less sparse matrices, starting to fail when $\|S^*\|_0/m^2 > 0.30$, while LRSD started to fail after $\|S^*\|_0/m^2 > 0.36$, and PLSMaFit after $\|S^*\|_0/m^2 > 0.70$. The extra capacity of PLSMaFit is due to the fact that correctly removing detected large entries reduces the effective density level of the sparse matrices involved. The drawback of PLSMaFit is that solving the partial-data model is more costly than solving the full-data model.

Remark 4.2 *It should be emphasized that the above two remedial techniques have not completely removed*

the limitation of model (4), even though they do alleviate it to some extent. For example, there apparently exist borderline situations where an appropriate threshold value ν becomes exceedingly difficult or impossible to determine.

4.7 Summary of numerical experiments

Extensive numerical experiments were performed on three types of test matrices: fully random, semi-random and non-random. Computational results indicate that, when the magnitude of sparse matrices does not dominate that of the low-rank ones, our algorithm implemented in **LMaFit** generally outperforms the two state-of-the-art nuclear minimization algorithms in terms of both solution speed and recoverability. The speed advantage of **LMaFit** results from solving a model that does not require any SVD calculation. This speed advantage, already significant on very low-rank problems such as the video background extraction problems in Section 4.4, grows as matrix sizes and rank increase, reaching up to one or more orders of magnitude on tested random problems. Interestingly, **LMaFit** also demonstrates greater recoverability, able to solve harder problems with a higher rank or density level (relative to a given sample ratio) that nuclear-norm minimization algorithms have difficulty to solve.

For approximately low-rank matrices that do not have a clear-cut numerical rank, our approach can produce a similar solution quality compared with that of nuclear-norm minimization, while maintaining its speed advantage over the latter. For these approximately low-rank problems, both approaches generally require some trial-and-errors to find appropriate model parameters.

A main limitation of the proposed approach is its inability to solve problems where the sparse matrix dominates the low-rank one in magnitude. We have tested two remedial techniques that have shown promises to alleviate this difficulty at least for some classes of problems.

5 Conclusions

We have studied a practical procedure for the emerging problem of sparse matrix separation based on solving a low-rank matrix factorization model by an augmented Lagrangian alternating direction method.

In terms of recoverability, the proposed model complements well the nuclear-norm minimization model. On one hand, the factorization model is unable to solve “sparse-matrix-dominated” (measured by magnitude) problems. On other problems, it is able to extend the frontier of recoverable problems beyond those solvable by nuclear-norm minimization. This phenomenon seems particularly notable in situations, arguably more realistic in many applications, where incoherence assumptions do not hold as required by the current theory for recoverability. In addition, we have demonstrated that the proposed approach is capable of solving problems with partial data or corrupted data or both, an example being low-rank matrix completion problems based on a subset of observed entries that contain impulsive noise.

In terms of computational efficiency, the proposed approach has exhibited a significant speed advantage over nuclear-norm minimization, yielding a much improved scalability for solving large-scale problems in practice. The fast speed of the proposed algorithm, often up to one or more order of magnitude faster than nuclear-norm minimization codes compared in our numerical tests, is a direct result of solving the factorization model free of any SVD calculations.

In spite of non-convexity in the factorization model and a lack of theoretical guarantee, the proposed approach has numerically performed as reliably and stably as nuclear-norm minimization on all the tested

matrix classes that the approach is capable of solving. This phenomenon and other theoretical issues remain to be better understood and warrant further investigations.

Acknowledgment

The work of Yuan Shen has been supported by the Chinese Scholarship Council during his visit to Rice University. The work of Zaiwen Wen was supported in part by NSF DMS-0439872 through UCLA IPAM. The work of Yin Zhang has been supported in part by NSF Grant DMS-0811188 and ONR Grant N00014-08-1-1101.

References

- [1] Dimitri P. Bertsekas and John N. Tsitsiklis. *Parallel and distributed computation: numerical methods*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.
- [2] J. Cai, E. J. Candès, and Z. Shen. A singular value thresholding algorithm for matrix completion. *SIAM J. Optim.*, 20(4):1956–1982, 2010.
- [3] E. J. Candès, X. Li, Y. Ma, and J. Wright. Robust principal component analysis? *Arxiv preprint arXiv:0912.3599*, December 2009.
- [4] E. J. Candès and B. Recht. Exact matrix completion via convex optimization. *Foundations of Computational Mathematics*, 9(6):717–772, 2009.
- [5] E. J. Candès and J. Romberg. Quantitative robust uncertainty principles and optimally sparse decompositions. *Foundations of Computational Mathematics*, 6(2):227–254, 2006.
- [6] E. J. Candès, J. Romberg, and T. Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *IEEE Transactions on Information Theory*, 52:489–509, 2006.
- [7] E. J. Candès and T. Tao. Near optimal signal recovery from random projections: universal encoding strategies. *IEEE Transactions on Information Theory*, 52(1):5406–5425, 2006.
- [8] E. J. Candès and T. Tao. The power of convex relaxation: Near-optimal matrix completion. *IEEE Transactions on Information Theory*, 56(5):2053 – 2080, May 2010.
- [9] V. Chandrasekaran, S. Sanghavi, Pablo A. Parrilo, and Alan S. Willsky. Rank-sparsity incoherence for matrix decomposition. *Arxiv preprint arXiv:0906.2220*, 2009.
- [10] V. Chandrasekaran, S. Sanghavi, Pablo A. Parrilo, and Alan S. Willsky. Sparse and low-rank matrix decompositions. In *in IFAC Symposium on System Identification*, 2009.
- [11] G. Chen and M. Teboulle. A proximal-based decomposition method for convex minimization problems. *Math. Programming*, 64(1, Ser. A):81–101, 1994.
- [12] Patrick L. Combettes and Jean-Christophe Pesquet. Proximal thresholding algorithm for minimization over orthonormal bases. *SIAM Journal on Optimization*, 18(4):1351–1376, 2007.
- [13] D. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52:1289–1306, 2006.
- [14] J. Eckstein and Dimitri P. Bertsekas. An alternating direction method for linear programming. LIDS-P, 1967. Cambridge, MA, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology.

- [15] J. Eckstein and Dimitri P. Bertsekas. On the Douglas-Rachford splitting method and the proximal point algorithm for maximal monotone operators. *Math. Programming*, 55(3, Ser. A):293–318, 1992.
- [16] M. Elad. Why simple shrinkage is still relevant for redundant representations? *IEEE Transactions on Information Theory*, 52(12):5559–5569, 2006.
- [17] M. Fazel and J. Goodman. Approximations for partially coherent optical imaging systems. Technical report, Stanford University, 1998.
- [18] M. Fazel, H. Hindi, and S. Boyd. Log-det heuristic for matrix rank minimization with applications to hankel and euclidean distance matrices. In *Proceedings of the American Control Conference*, 2003.
- [19] M. Figueiredo and R. Nowak. An EM algorithm for wavelet-based image restoration. *IEEE Transactions on Image Processing*, 12:906–916, 2003.
- [20] M. Fortin and R. Glowinski. *Augmented Lagrangian methods*, volume 15 of *Studies in Mathematics and its Applications*. North-Holland Publishing Co., Amsterdam, 1983. Applications to the numerical solution of boundary value problems, Translated from the French by B. Hunt and D. C. Spicer.
- [21] A. Ganeshy, J. Wright, X. Li, E. J. Candes, and Y. Ma. Dense error correction for low-rank matrices via principal component pursuit. *Information Theory Proceedings (ISIT), 2010 IEEE International Symposium on*, pages 1513 – 1517, June 2010.
- [22] R. Glowinski. *Numerical methods for nonlinear variational problems*. Springer-Verlag, New York, Berlin, Heidelberg, Tokyo,, 1984.
- [23] R. Glowinski and P. Le Tallec. *Augmented Lagrangian and operator-splitting methods in nonlinear mechanics*, volume 9 of *SIAM Studies in Applied Mathematics*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1989.
- [24] J. P. Haldar and D. Hernando. Rank-constrained solutions to linear matrix equations using powerfactorization. *Signal Processing Letters, IEEE*, 16:584–587, 2009.
- [25] E. T. Hale, W. Yin, and Y. Zhang. Fixed-point continuation for l_1 -minimization: methodology and convergence. *SIAM J. Optim.*, 19(3):1107–1130, 2008.
- [26] B. He, L. Liao, D. Han, and H. Yang. A new inexact alternating directions method for monotone variational inequalities. *Math. Program.*, 92(1, Ser. A):103–118, 2002.
- [27] B. He, H. Yang, and S. Wang. Alternating direction method with self-adaptive penalty parameters for monotone variational inequalities. *J. Optim. Theory Appl.*, 106(2):337–356, 2000.
- [28] Krzysztof C. Kiwiel, Charles H. Rosa, and Andrzej Ruszczyński. Proximal decomposition via alternating linearization. *SIAM J. Optim.*, 9(3):668–689, 1999.
- [29] S. Kontogiorgis and Robert R. Meyer. A variable-penalty alternating directions method for convex optimization. *Math. Programming*, 83(1, Ser. A):29–53, 1998.
- [30] Rasmus M. Larsen. Propack: Software for large and sparse svd calculations. <http://soi.stanford.edu/~rmunk/PROPACK>.
- [31] L. Li, W. Huang, I. Gu, and Q. Tian. Statistical modeling of complex backgrounds for foreground object detection. *IEEE Transactions on Image Processing*, 13(11):1459–1472, 2004.
- [32] Z. Lin, M. Chen, L. Wu, and Y. Ma. The augmented lagrange multiplier method for exact recovery of a corrupted low-rank matrices. *Mathematical Programming*, *submitted*, 2009.
- [33] Z. Lin, A. Ganesh, J. Wright, L. Wu, M. Chen, and Y. Ma. Fast convex optimization algorithms

- for exact recovery of a corrupted low-rank matrix. 2009. http://yima.csl.uiuc.edu/psfile/rpca_algorithms.pdf.
- [34] H. Hindi M. Fazel and S. Boyd. A rank minimization heuristic with application to minimum order system approximation. In *Proceedings American Control Conference*, volume 6, pages 4734–4739, 2001.
 - [35] S. Ma, D. Goldfarb, and L. Chen. Fixed point and Bregman iterative methods for matrix rank minimization. Technical report, Department of IEOR, Columbia University, 2008.
 - [36] Y. Nesterov. Introductory lectures on convex optimization. 87:xviii+236, 2004. A basic course.
 - [37] M. Tao and X. Yuan. Recovering low-rank and sparse components of matrices from incomplete and noisy observations. *SIAM J. Optim.*, 21(1):57–81, 2011.
 - [38] P. Tseng. Alternating projection-proximal methods for convex programming and variational inequalities. *SIAM J. Optim.*, 7(4):951–965, 1997.
 - [39] P. Tseng. On accelerated proximal gradient methods for convex-concave optimization. *submitted to SIAM Journal on Optimization*, 2008.
 - [40] Y. Wang, J. Yang, W. Yin, and Y. Zhang. A new alternating minimization algorithm for total variation image reconstruction. *SIAM J. Imaging Sci.*, 1(3):248–272, 2008.
 - [41] Z. Wen, D. Goldfarb, and W. Yin. Alternating direction augmented lagrangian methods for semidefinite programming. Technical report, Dept of IEOR, Columbia University, 2009.
 - [42] Z. Wen, W Yin, and Y. Zhang. Solving a low-rank factorization model for matrix completion by a nonlinear successive over-relaxation algorithm. Technical report, 2010. http://www.optimization-online.org/DB_FILE/2010/03/2581.pdf.
 - [43] J. Wright, A. Ganesh, S. Rao, Y. Peng, and Y. Ma. Robust principal component analysis: Exact recovery of corrupted low-rank matrices by convex optimization. In *In Proceedings of Neural Information Processing Systems (NIPS)*, December 2009.
 - [44] J. Wright, A. Ganesh, S. Rao, Y. Peng, and Y. Ma. Robust principal component analysis: Exact recovery of corrupted low-rank matrices via convex optimization. *submitted to Journal of the ACM*, 2009.
 - [45] J. Yang, Y. Zhang, and W. Yin. An efficient tvl1 algorithm for deblurring multichannel images corrupted by impulsive noise. *SIAM Journal on Scientific Computing*, 31(4):2842–2865, June 2009.
 - [46] X. Yuan and J. Yang. Sparse and low-rank matrix decomposition via alternating direction methods. Technical report, Dept. of Mathematics, Hong Kong Baptist University, 2009.
 - [47] Y. Zhang. LMaFit: Low-rank matrix fitting, 2009. <http://www.caam.rice.edu/~optimization/L1/LMaFit/>.
 - [48] Y. Zhang. User’s guide for yall1: Your algorithms for l1 optimization. Technical report, Rice University, 2009. <http://www.caam.rice.edu/~zhang/reports/tr0917.pdf>.
 - [49] Y. Zhang. An alternating direction algorithm for nonnegative matrix factorization. Technical report, Rice University, 2010. <http://www.caam.rice.edu/~yzhang/reports/tr1003.pdf>.

Figure 11: Video separation. From left to right: original, separated results by IALM and LMaFit .



Above: frames from clip “Hall”. Below: frames from clip “bootstrap”

