Nonsmooth Low-Rank Matrix Recovery: Methodology, Theory and Algorithm

Wei Tu[1], Peng Liu[2], Yi Liu[3], Guodong Li[4], Bei Jiang[3], Linglong Kong[3(✉)], Hengshuai Yao[5], and Shangling Jui[6]

[1] Department of Public Health Sciences and Canadian Cancer Trials Group, Queen's University, Kingston, Canada
[2] School of Mathematics, Statistics and Actuarial Science, University of Kent, Canterbury, UK
[3] Department of Mathematical and Statistical Sciences, University of Alberta, Edmonton, Canada
lkong@ualberta.ca
[4] Department of Statistics and Actuarial Science, University of Hong Kong, Pok Fu Lam, Hong Kong
[5] Huawei Hisilicon, Edmonton, Canada
[6] Huawei Hisilicon, Shanghai, China

Abstract. Many interesting problems in statistics and machine learning can be written as $\min_x F(x) = f(x) + g(x)$, where x is the model parameter, f is the loss and g is the regularizer. Examples include regularized regression in high-dimensional feature selection and low-rank matrix/tensor factorization. Sometimes the loss function and/or the regularizer is nonsmooth due to the nature of the problem, for example, $f(x)$ could be quantile loss to induce some robustness or to put more focus on different parts of the distribution other than the mean. In this paper we propose a general framework to deal with situations when you have nonsmooth loss or regularizer. Specifically we use low-rank matrix recovery as an example to demonstrate the main idea. The framework involves two main steps: the optimal smoothing of the loss function or regularizer and then a gradient based algorithm to solve the smoothed loss. The proposed smoothing pipeline is highly flexible, computationally efficient, easy to implement and well suited for problems with high-dimensional data. Strong theoretical convergence guarantee has also been established. In the numerical studies, we used L_1 loss as an example to illustrate the practicability of the proposed pipeline. Various state-of-the-art algorithms such as Adam, NAG and YellowFin all show promising results for the smoothed loss.

Keywords: Matrix factorization · Nonsmooth · Low-rank matrix · Nesterov's smoothing · Optimization

1 Introduction

Many problem in statistics and machine learning can be formulated as the following form:

$$\min_x F(x) = f(x) + g(x),$$

K. Arai (Ed.): FTC 2021, LNNS 358, pp. 848–862, 2022.
https://doi.org/10.1007/978-3-030-89906-6_54

where x is the parameter, f is the loss and g is the regularizer. Examples includes penalized regression in high-dimensional feature selection [30] and low-rank matrix/tensor recovery. Typically both $f(x)$ and $g(x)$ are proper convex functions such as using L_2 loss for $f(x)$. However, in some problems, due to the need of sparsity, robustness or other structural requirement of the parameter space, a nonsmooth or even nonconvex loss function or regularizer is often needed. In this paper we propose a general framework to deal with situations when you have nonsmooth loss or regularizer. Specifically we use low-rank matrix recovery as an example to illustrate the main idea.

In practice, many high dimensional matrices essentially have low-rank structure; see, for example, recommender systems [27], stochastic system realization [25] in systems and control, computer algebra [4], psychometrics [19] and even quantum state tomography [13] in physics. Meanwhile, these matrices usually are partially observed, and a lot of entries are left unobserved due to many different reasons. For example, we can only observe a few ratings from any particular recommender systems; or the quantum states have an exponentially large size so that it's not possible to obtain same scale observations. For these partially observed matrices with a high missing rate, it is of interest to ask "How to estimate the matrix with low-rank structure?" or "How to recover the low-rank matrix effectively?". This leads to an important problem of low-rank matrix recovery.

Since the Netflix prize competition in 2009, matrix factorization has been shown to outperform traditional nearest-neighbor based techniques in the sense that it allows the incorporation of additional information such as temporal effects, confidence levels and so on [18]. The basic idea of matrix factorization is to decompose the target matrix $M^* \in \mathbb{R}^{m\times n}$ into a bilinear form:

$$M^* = U^\top V$$

where $U \in \mathbb{R}^{r\times m}, V \in \mathbb{R}^{r\times n}$, and the rank of M^* is no more than r with $r \leq \min(m, n)$.

Due to the rapid improvement of computation power, matrix factorization has received more and more attention in various fields; see [6,16,26,37] and among others. Most currently used methods are based on the L_2 loss, which is the optimal choice when the noise is Gaussian distributed. However, it is sensitive to outliers, and one possible solution is to consider a loss function other than the L_2 loss; see [5,16,24].

Meanwhile, as shown in [9], the nonsmooth optimization problem plays an important role in many areas such as image restoration, signal reconstruction, optimal control, and so on. In statistics, the least absolute deviation is well known to be robust to highly skewed and/or heavy-tailed data. The Manhattan distance in machine learning is actually based on L_1 loss. Quantile regression, which corresponds to the quantile loss, is another important estimating method in statistics, and is also commonly used to handle the highly skewed data. It is noteworthy to point out that both L_1 and quantile loss functions are nonsmooth. Other useful nonsmooth functions in statistics and machine learning include indicator function, step function, max function and so on [7,33,34].

Thus it is of importance to consider matrix factorization with nonsmooth loss function for the matrix recovery. Various algorithms, including the simplex, subgradient and quasi-monotone methods, have been proposed to tackle with the nonsmooth optimization, while few of them are efficient for recovering low-rank matrices with high dimensions. Smooth approximation recently has been studied for nonsmooth optimization in many areas, such as complementarity problems, optimal control, eigenvalue optimization, etc., and it has been shown to be efficient even for the case with nonsmooth constraints; see, for example, [2,8] and [10].

This paper considers the problem of low-rank matrix recovery from linear measurements. A general nonsmooth loss function is considered here, and Nesterov's smoothing method [22] is then applied to obtain an optimal smooth approximation. In practice, according to the specific nature of the problem and data, one can choose a suitable nonsmooth loss function, satisfying Nesterov's assumptions in [22], such that an efficient algorithm can be obtained. Due to the bilinear structure of matrix factorization, the alternating minimization method is thus employed to search for the solutions and, at each step, we compare the performance of various algorithms, which are based on gradient descent and momentum. Compared with previous work, this paper is more general in the following ways: 1) the transformation matrices we considered are more general; 2) a strong convergence guarantee is established for the proposed algorithm; 3) different state-of-the-art gradient based algorithms are used and compared. For example, vanilla gradient descent, Nesterov's momentum method [22], Adam [17] as well as YellowFin [36] algorithm. All the algorithms substantially improve the performance of original nonsmooth problem.

Here are the structure of the paper. Section 2 introduces the mathematical settings of the problem, while the proposed algorithms are presented in details in Sect. 3, and the theoretical convergence analysis results can be found in Sect. 4. In Sect. 5, we illustrate the effectiveness of the proposed framework using the popularly used L_1 loss as a special example. Different gradient and momentum based algorithms are used to compare their performances.

2 Methodology Framework

This paper considers the model,

$$b_i = \langle A_i, M^* \rangle + \epsilon_i, \quad i = 1, \ldots, p, \tag{1}$$

where M^* is the true value, we can observe $\{A_i, b_i\}, i = 1, \cdots, p$, and ϵ_i is the error term. Here $A_i \in \mathbb{R}^{m \times n}$ with $1 \leq i \leq p$ are given transformation matrices. The low-rank matrix M^* has rank r with r smaller than $\min(m, n, p)$, which allows us to decompose M^* into two matrices. Specifically, we can write $M^* = U^{*\top} V^*$, with $U^* \in \mathbb{R}^{r \times m}$ and $V^* \in \mathbb{R}^{r \times n}$. The following optimization can be used to recover M^*:

$$\min_{U,V} \frac{1}{p} \sum_{i=1}^{p} f(b_i - \langle A_i, U^\top V \rangle), \tag{2}$$

where $f(\cdot)$ is a nonsmooth objective function. Let $\mathcal{A} : \mathbb{R}^{m \times n} \to \mathbb{R}^p$ be an affine transformation with the ith entry of $\mathcal{A}(M^*)$ being $\langle A_i, M^* \rangle$, and $\boldsymbol{f}(x) =$

$p^{-1}\sum_{i=1}^{p} f(x_i)$ for a vector $x = (x_1, ..., x_p)^\top$. Equation (2) can be rewritten as follows:

$$\min_{U,V} \boldsymbol{f}(b - \mathcal{A}(U^\top V)). \tag{3}$$

Assume that $\boldsymbol{f}(\cdot)$ is differentiable almost everywhere, and has the following structure:

$$\boldsymbol{f}(b - \mathcal{A}(U^\top V)) = \hat{\boldsymbol{f}}(b - \mathcal{A}(U^\top V)) + \max_u \left\{ \langle B(b - \mathcal{A}(U^\top V), u)_2 - \hat{\phi}(u) \rangle \right\}, \tag{4}$$

where $\hat{\boldsymbol{f}}$ is a continuous and convex function; see [22]. We then can obtain the following optimal smooth approximation:

$$\begin{aligned} \boldsymbol{f}_\mu(b - \mathcal{A}(U^\top V)) = \hat{\boldsymbol{f}}(b - \mathcal{A}(U^\top V)) + \\ \max_u \left\{ \langle B(b - (U^\top V)), u \rangle_2 - \hat{\phi}(u) - \mu d_2(u) \right\}, \end{aligned} \tag{5}$$

and μ is positive and used for smoothness.

The above smooth approximation is made to function $\boldsymbol{f}(\cdot)$ for vector $b - \mathcal{A}(U^\top V)$, rather than the unknown matrix parameters U and V, since it can be handled more conveniently. Moreover, under the restricted isometry property assumption on $\mathcal{A}$, the smooth approximation does not change the convergence rate.

In meanwhile, due to the scale problem, the direct solutions to (5) may not have a proper structure, and one commonly used correction is to introduce a penalty of $\lambda(\|U\|_F^2 + \|V\|_F^2)$; see, for example, [28,38] and among others. However, such penalization can not preserve the intrinsic structure for U and V. This paper use the Procrustes flow penalty [20,31] instead, and our optimization problem becomes

$$\min_{U,V} \boldsymbol{f}_\mu(b - \mathcal{A}(U^\top V)) + \lambda \|UU^\top - VV^\top\|_F^2, \tag{6}$$

where the ad hoc choice of λ is $1/16$, the objective function is smooth with respect to U and V, and can be denoted by $\boldsymbol{f}_\mu^\lambda(U, V)$. The original nonsmooth optimization problem corresponds to

$$\min_{U,V} \boldsymbol{f}(b - \mathcal{A}(U^\top V)) + \lambda \|UU^\top - VV^\top\|_F^2, \tag{7}$$

which can be represented by $\boldsymbol{f}^\lambda(U, V)$.

Due to the bilinear structure of $\boldsymbol{f}_\mu^\lambda(U, V)$ regarding to U and V at (6), we adopt the alternating minimization method to search for the solutions. During each iteration, one of the target matrices U and V will be fixed, and the algorithm will update on another one until the objective function stabilizes. The algorithm keeps updating until the objective function converges. In the literature of matrix factorization, [16] showed its convergence for L_2 loss function, and this paper will further establish the convergence for a general nonsmooth loss function.

Algorithm 1: Initialization by SVP algorithm

Input: $\mathcal{A}$, b, tolerance ϵ, step size ξ_t for $t = 0, 1, \cdots$, $M^0 = 0_{m\times n}$
Output: M^{t+1}

Repeat
$Y^{t+1} \leftarrow M^t - \xi_t \nabla_M \boldsymbol{f}_\mu(b - \mathcal{A}(M^t))$
Compute top r singular vectors of Y^{t+1}:U_r, Σ_r, V_r
$M^{t+1} \leftarrow U_r \Sigma_r V_r$
$t \leftarrow t + 1$
Until $\|M^{t+1} - M^t\|_F \leq \epsilon$

3 Algorithm

There are three steps in our algorithm, and the first one is the initialization; see Algorithm 1. The initialization step is crucial since it can affects how quickly the gradient updating approaches the optimal values. If the initial values for U and V are close to orthogonal to the true values, the gradient updating might be very slow or even impossible to reach the optimal. The singular value projection (SVP) is used to compute starting values, which was proposed by [15] and later used by [1,12,31] and so on.

Algorithm 1 can be written into

$$M^{t+1} \leftarrow \mathcal{P}_r\left(M^t - \xi_t \nabla_M \boldsymbol{f}_\mu(b - \mathcal{A}(M^t))\right),$$

where $\mathcal{P}_r$ denotes a projection onto the space of rank-r matrices. Algorithm 1 itself can be used to target matrix X^* [15] for small to moderate problems. When the dimensionality of X^* is large, the SVP algorithm can be slow, but a small number of iterations provide a sufficient start for further steps.

The second step is the alternating minimization; see Algorithm 2. During each iteration, one of the target matrices U and V will be fixed, and the algorithm will update on another one until the objective function stabilizes. The algorithm keeps updating until the objective function converges. $\hat{U}$ and $\hat{V}$ are used to denote the final outputs for U and V.

Within each update of U and V (step 1.1 and 1.2) in Sect. 2, we use Nesterov's accelerate gradient (NAG) method. Various other methods have been proposed in the literature for this use. [16] and [31] used the vanilla gradient descent method, while it may be slow in our nonsmooth matrix factorization settings. Algorithm 3 introduces Nesterov's momentum method to update the value of U, while that of V is fixed. Similarly we can gives the algorithm to update the value of V. In Algorithm 3, $\boldsymbol{\nu}_{(i)}^t$ represents the momentum term, γ represents the momentum parameter, η represents the learning rate. A typical choice of γ is 0.9 [29,36]. To explore how some other state-of-the-art gradient methods perform in the proposed setting, we consider two other momentum-based algorithms: Adam in [17] and Yellowfin in [36]. These methods have shown superior performances in other large-scale applications such as deep learning and deep reinforcement learning.

Algorithm 2: Alternating Minimization

Input: U^0, V^0
Output: $U^{n_{max}}, V^{n_{max}}$
Repeat
1.1.Update U^t with $U^{t+1} = NAG(U^t, V^t)$
1.2.Update V^t with $V^{t+1} = NAG(U^{t+1}, V^t)$
Until convergence

Algorithm 3: Nesterov's Accelerate Gradient (NAG) Method for U^{t+1}

Input: U^t, V^t
Output: U^{t+1}
Repeat
$\boldsymbol{\nu}_{(i)}^t = \gamma \boldsymbol{\nu}_{(i-1)}^t + \eta \nabla_U \boldsymbol{f}_\mu^\lambda (U_{(i-1)}^t - \gamma \boldsymbol{\nu}_{(i-1)}^t, V^t)$
$U_{(i)}^t = U_{(i-1)}^t + \boldsymbol{\nu}_{(i)}^t$
Until convergence

4 Convergence Analysis

Denote $\{\hat{U}^\pi, \hat{V}^\pi\} = \min_{U,V} \boldsymbol{f}_\mu^\lambda(U, V)$ and $\{\hat{U}, \hat{V}\} = \min_{U,V} \boldsymbol{f}^\lambda(U, V)$. The theoretical properties of the used algorithms in this paper is similar to the ones in [32], namely, the convergence of the smoothed objective function. Here we list the main algorithms and the details of the proof can be found in Appendix of [32].

Theorem 1. *(Convergence of optimal solution of smoothed objective function) As* $\pi \to 0^+$*, we have* $\hat{U}^{\pi\top}\hat{V}^\pi \to \hat{U}^\top \hat{V}$.

Many existing literatures on smoothing technique usually only focus on analyzing the theoretical properties while ignore the relationship between smooth objective function and original nonsmooth objective function, for example, [3]. However, Theorem 1 shows that if we only focus on optimizing smooth objective function, we can still obtain the optimal solution for nonsmooth objective function, the benefit is that we can thus have lots of choices with respect to the algorithms based on smooth objective function, then we can simple choose a fast one to obtain the solution.

The following matrices distance measure is used:

$$\text{dist}(U, U^\dagger) = \min_{R \in \mathbb{R}^{r\times r}: R^\top R = I_r} \|U - RU^\dagger\|_F,$$

where $U, U^\dagger \in \mathbb{R}^{r\times m}$ with $m \geq r$; see [31].

Theorem 2. *Let* $M \in \mathbb{R}^{m\times n}$ *be a rank r matrix, with singular values* $\sigma_1(M) \geq \sigma_2(M) \geq \cdots \geq \sigma_r(M) > 0$ *and condition number* $\kappa = \sigma_1(M)/\sigma_r(M)$*, let* $M = A^\top \Sigma B$ *be the SVD decomposition. Define* $U = A^\top \Sigma^{1/2} \in \mathbb{R}^{m\times r}$, $V = B^\top \Sigma^{1/2} \in \mathbb{R}^{n\times r}$. *Assume* $\mathcal{A}$ *satisfies a rank-6r RIP condition with RIP constant*

$\sigma_{6r} < \frac{1}{25}$, $\xi_t = \frac{1}{p}$. Then use $T_0 \geq 3\log(\sqrt{r}\kappa) + 5$ iterations in SVP initialization yields a solution U_0, V_0 obeying

$$dist\left(\begin{bmatrix} U_0 \\ V_0 \end{bmatrix}, \begin{bmatrix} U \\ V \end{bmatrix}\right) \leq \frac{1}{4}\sigma_r(U). \tag{8}$$

Furthermore, starting from any initial solution obeying (8), the t-th iterate of Algorithm 2 satisfies

$$dist\left(\begin{bmatrix} U_t \\ V_t \end{bmatrix}, \begin{bmatrix} U \\ V \end{bmatrix}\right) \leq \frac{1}{4}(1-\tilde{\tau}_1)^t \frac{\tilde{\mu}}{\tilde{\xi}} \frac{1+\delta_r}{1-\delta_r}\sigma_r(U) \tag{9}$$

under Nesterov's momentum method.

We make several contributions in Theorem 2, the first one is that we extend [15]'s SVP algorithm from least square matrix factorization to nonsmooth matrix factorization, in addition, we provide theoretical convergence guarantees for alternating minimization with Nesterov's momentum method for general objective function, this generalize [31]'s linear convergence guarantees for alternating minimization with gradient descent for least square objective function. We also would like to mention that though [35] also provide a smoothing approximation using Nesterov's smoothing technique, however, they are dealing with nonnegative matrix factorization case, which is much simpler than ours and they also did not provide rigorous theoretical analysis.

5 Numerical Studies

Several different commonly used unsmooth loss function can be used. Here we choose the mostly common seen L_1 loss during the simulation to illustrate the practical applicability of our theoretical results and provide insights about the choice of the algorithms. The corresponding smoothed approximation of L_1 loss is the popular Huber loss function (Huber 1981) in robust statistics. The Huber loss function is defined as

$$L_\mu(a) = \begin{cases} \frac{1}{2\mu}|a|^2 & \text{for} \quad |a| \leq \mu \\ |a| - \frac{\mu}{2} & \text{otherwise} \end{cases}, \tag{10}$$

where μ is the predetermined parameter controlling the tradeoff between smoothness and precision. When $\mu \to 0^+$, the Huber function converge to absolute loss uniformly. When $\mu \to +\infty$, the Huber function resembles the L_2 loss. Figure 1 further illustrates the differences of these loss functions.

5.1 Synthetic Data

The dataset is generated in the following manner: all entries in $U \in \mathbb{R}^{r\times m}$, $V \in \mathbb{R}^{r\times n}$ and $A_i \in \mathbb{R}^{m\times n}$ are independently sampled from Gaussian distribution $N(0,1)$. The ground truth M^* to recover is then calculated as $M^* = U^\top V$, and the rank of M^*

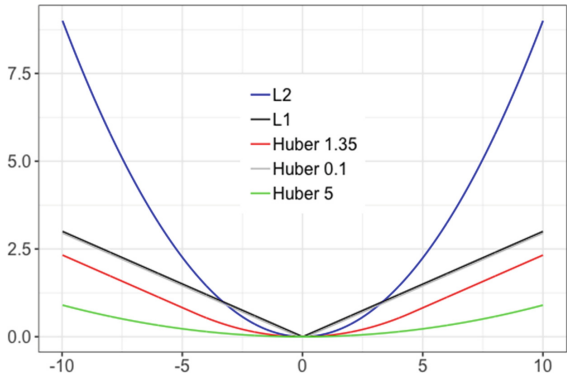


Fig. 1. The comparison of different loss functions. Here huber 1.35 in the legend denotes huber loss with $\mu = 1.35$.

is r. The observations b_i are generated following the assumed data generating process indicated in (1).

The following two metrics are used to measure the performances of different algorithms: 1) the value of the loss function; 2) relative recovery error: $\|\hat{U}^\top \hat{V} - M\|_F / \|M\|_F$, where $\hat{U}$ and $\hat{V}$ are the estimated U and V using the proposed smoothing method. For both metrics, a smaller value is desired.

Smooth Parameter μ : The smoothing parameter μ in (10) controls the tradeoff between the smoothness and precision. In this simulation we aim to investigate the relationship between the choice of μ and the relative recovery error. Without loss of generality, $m = n = 32$, $p = 512$ and $r = 10$ is used here. To enhance the difference between L_1 loss and L_2 loss, all oberservations b_i have been contaminated by a Cauchy noise e_i, and Nesterov's momentum method is used for this simulation (Fig. 2).

Figure 3 shows the trend between the smoothing parameter and the relative recovery error, and the x axis is the natural algorithm transformed μ. We observe three interesting patterns from this plot: 1) the performance of the algorithm is not sensitive to the choice of μ as we can see a reasonable small choice of μ will result in a small recovery error; 2) as μ approaches 0, the huber loss becomes very close to the L_1 loss, and recovery error increases slightly. This might due to the non-smoothness of L_1 loss; 3) as μ approaches $+\infty$, the huber loss becomes very close to the L_2 loss, and L_2 loss does not work well here due to the Cauchy noise added to the data. The empirical results here also further verifies our theoretical result in Theorem 1: as the smoothing parameter approaches to 0, the optimal solution of the smoothed objective function approaches to that of the nonsmooth one (L_1 loss here).

Algorithm Comparisons: The third step of the algorithm of updating U or V can be solved by many different algorithms. Four methods are implemented here: the vanilla gradient descent method (GD), the Nesterov's momentum method (NAG) [21], the

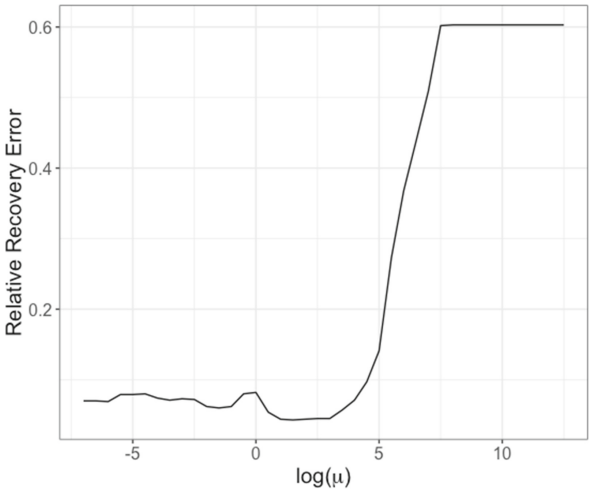


Fig. 2. Relative recovery error under different choices of smooth parameter μ. The horizontal axis is the μ in algorithm scale, and the vertical axis is the relative recovery error.

Adam [17] method and the YellowFin [36] algorithm that features step size auto-tuning capacity.

For each algorithm, we start with an unreasonably high step size, $\eta = 1$. Under such a high step size, most algorithms are expected to diverge or suffer from numerical instability. Then, we repetitively decrease the step size via multiplying η by $\frac{1}{\sqrt{10}}$ for each iteration until the algorithm starts to converge properly.

To limit our simulation in reasonable amount of time, m, n, r and p are chosen as 64, 64, 8 and 2048. Here we choose $\mu = 1.35$ as suggested by [14], and later on used by [23] and [11], among others. For each algorithm, the experiments are repeated 100 times. In each experiment, the step size tuning procedure yield incidental result for GD, NAG and ADAM. Step size are all decided as $10^{-2.5}$. The exception is the YellowFin algorithm, even extreme choices like 10 or 10^{-9} won't significantly alter the outcome of the optimization because of the fact that unlike other algorithms, the internal auto tuning process of YellowFin will override the preassigned step size before the end of first iteration.

One of the motivation to use nonsmooth objective function such as L_1 loss is the good robust performance of it. We have experimented different choices of error distributions for observations b_i. Due to the limited space here, we select two different scenarios have to present the findings: no contamination and adding a chi squared noise to the observations b_i. The findings of each scenario has been summarized below:
No Contamination: Figure 4 and Fig. 5 show the behaviors of the loss and relative recovery error of four different algorithms when the observed b_i contains no error. We observe 4 interesting findings: 1) all algorithms converge eventually and the final relative recovery errors are all very close to 0; 2) as expected, the vanilla gradient descent converges the slowest in terms of both loss and relative recovery error; 3) Adam and

Table 1. Number of iterations needed to reach a relative recovery error smaller than 20%, 10%, 5%, 1% for each algorithm under no contamination setting

	20%	10%	5%	1%
Adam	775	962	1105	1251
GD	3799	4788	>5000	>5000
NAG	709	881	1103	1148
YellowFin	1341	1509	1668	1764

Nestrov's momentum method have very similar behaviors, and Adam slightly outperforms Nestrov's momentum method at the later stage of optimization. Both of them are considerably more desirable than the vanilla gradient descent method and do not differ significantly in practical use. 4) The YellowFin algorithm have different behavior. It takes extra iterations for the algorithm to figure out optimal learning rate before the loss function starts to monotonously decrease.

Table 1 presents the number of iterations needed to reach a relative recovery error smaller than a certain threshold for each algorithms. We can see that all algorithms expect than gradient descent reaches 20% error in a relatively fast speed. Towards the end, from 5% to 1%, NAG has only needs less than 50 steps, while Adam takes about 150 steps.

Table 2. Number of iterations needed to reach a relative recovery error smaller than 50%, 30%, 25%, 20% for each algorithm under chi-squared noise setting, and here NaN means the algorithm can not reach an error smaller than this value

	50%	30%	25%	20%
Adam	351	813	1108	NaN
GD	1289	4464	>5000	NaN
NAG	257	852	1229	NaN
YellowFin	444	1334	1606	NaN

Chi Squared Noise: Figure 6 and Fig. 7 show the behaviors of the loss and relative recovery error of four different algorithms when each of the observed b_i is contaminated by a chi squared noise. Specifically, each b_i is replaced by $b_i + 10 * e_i$, where e_i follows a chi squared distribution with 3 degree of freedom. Compare to the no contamination setting, we have noticed that even though all algorithms converge eventually, the final loss and relative recovery error are not as close as to 0 as before. This is expected as the Cauchy error brought unnegligible noise to the observations. For the comparisons between different algorithms, the patterns are similar as the no contamination setting.

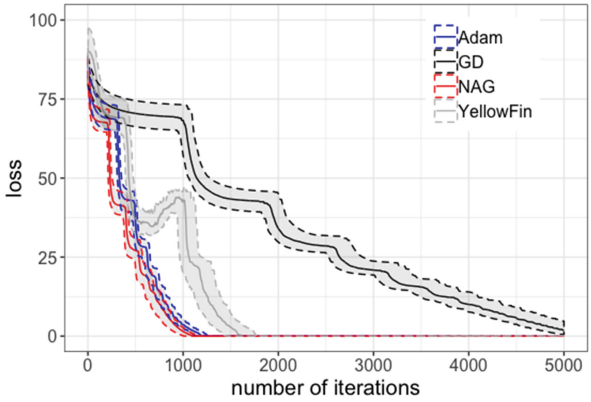


Fig. 3. Loss function curves under no contamination setting. The horizontal axis is training steps, and the vertical axis is the value of loss function. Each curve represents median over 100 runs, and the area between 0.25 and 0.75 quantile are plotted as Shadow.

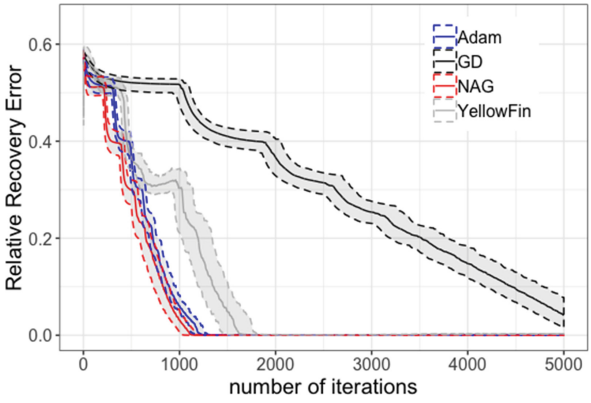


Fig. 4. Relative matrix recovery error curves under no contamination setting. The horizontal axis is training steps, and the vertical axis is the recovery error. Each Curve represents median over 100 runs, and the area between 0.25 and 0.75 quantile are plotted as shadow.

Table 2 presents the number of iterations needed to reach a relative recovery error smaller than a certain threshold for each algorithms. We can see that no algorithm can reach an error smaller than 20%, which shows that contaminated observations can bring serious trouble in the recovery of the original matrix. Adam and NAG have similar performances here, while Adam needs slightly smaller iterations to reach 25% error.

5.2 Real World Data Experiment

In this section, we demonstrate the efficiency of our method via a real world example. Not all data are normally distributed as is in our synthesized data set, furthermore, noise is ubiquitously unavoidable in real world practices.

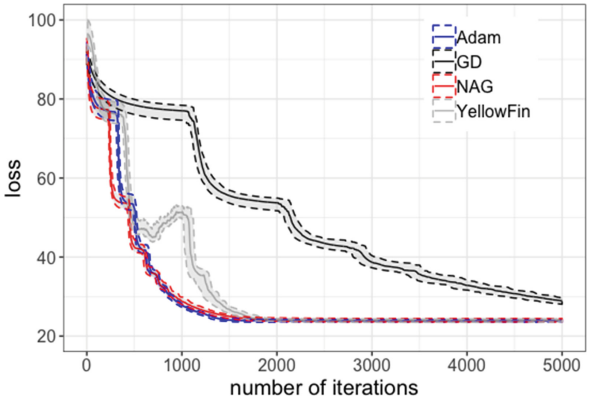


Fig. 5. Loss function curves under cauchy noise setting. The horizontal axis is training steps, and the vertical axis is the value of loss function. Each curve represents median over 100 runs, and the area between 0.25 and 0.75 quantile are plotted as shadow.

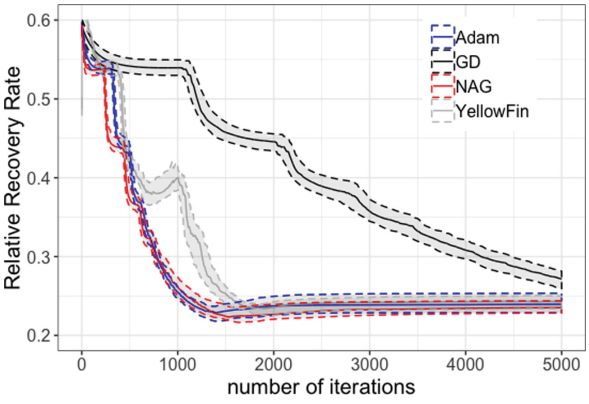
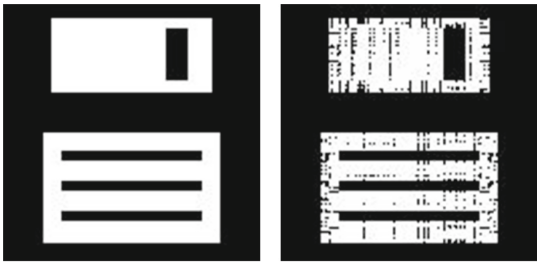


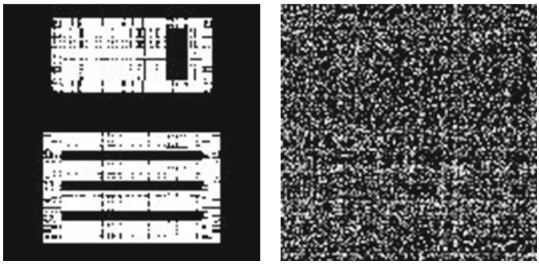
Fig. 6. Relative matrix recovery error curves under cauchy noise setting. The horizontal axis is training steps, and the vertical axis is the recovery error. Each curve represents median over 100 runs, and the area between 0.25 and 0.75 quantile are plotted as shadow.

The real world data we use in this experiment is an old-school gray-scale saving icon with dimension of $m = n = 128$ and the rank of this picture is $r = 6$. 8000 normal distributed A_i are generated in the setting of matrix sensing and b_i are calculated accordingly. To show the robustness trait of L_1 loss is well preserved by our smoothing method, a noise with independent *Cauchy* distribution is additionally applied to all b_i.

The results are shown in Fig. 7. L_2 loss can not recover the image and it turns out totally blurred. Both L_1 and Huber case can recover recognizable picture benefit from our loss. However, L_1 optimization based on nonsmooth subgradient method takes over 20x more time to reach converge than the Huber design and the ratio will increase even more as the scale of the problem exaggerates.



(a) Truth (Error: 0.00%) (b) L_1 (Error: 0.71%)



(c) Huber (Error: 1.01%) (d) L_2 (Error: $> 100\%$)

Fig. 7. Final recovery result for different loss function

6 Conclusion

This paper considers the matrix factorization for low-rank matrix recovery, and a general nonsmooth loss function is assumed. It includes the commonly used L_1 and quantile loss functions as special cases, and this gives us much flexibility by choosing a suitable form according to our knowledge and observations.

In the proposed algorithm, we first suggest an optimal smooth approximation of the nonsmooth objective function [22], then a lot of algorithms based on gradient can be applied to the problem, we use the vanilla gradient descent, Nesterov's momentum algorithm, adam as well as YellowFin as examples and compare their performance. Though smoothing changed the problem's structure; however, the benefit is that it brings us much more flexibility to choose different algorithms.

References

1. Achlioptas, D., McSherry, F.: Fast computation of low-rank matrix approximations. J. ACM (JACM) **54**(2), 9 (2007)
2. Alefeld, G., Chen, X.: A regularized projection method for complementarity problems with non-lipschitzian functions. Math. Comput. **77**(261), 379–395 (2008)
3. Aravkin, A.Y., Kambadur, A., Lozano, A.C., Luss, R.: Sparse quantile huber regression for efficient and robust estimation. arXiv preprint arXiv:1402.4624, 2014
4. Barnett, S.: Greatest common divisor of two polynomials. Linear Algebra Appl. **3**(1), 7–9 (1970)

5. Bhojanapalli, S., Kyrillidis, A., Sanghavi, S.: Dropping convexity for faster semi-definite optimization. In: Conference on Learning Theory, pp. 530–582 (2016)
6. Bokde, D., Girase, S., Mukhopadhyay, D.: Matrix factorization model in collaborative filtering algorithms: a survey. Procedia Comput. Sci. **49**, 136–146 (2015)
7. Bottou, L.: Large-scale machine learning with stochastic gradient descent. In: Proceedings of COMPSTAT 2010, pp. 177–186. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-7908-2604-3_16
8. Chen, X.: First order conditions for nonsmooth discretized constrained optimal control problems. SIAM J. Control Optim. **42**(6), 2004–2015 (2004)
9. Chen, X.: Smoothing methods for nonsmooth, nonconvex minimization. Math. Program. **134**(1), 71–99 (2012)
10. Chen, X., Womersley, R.S., Ye, J.J.: Minimizing the condition number of a gram matrix. SIAM J. Optim. **21**(1), 127–148 (2011)
11. Fan, J.: Local Polynomial Modelling and its Applications: Monographs on Statistics and Applied Probability, vol. 66. Routledge (2018)
12. Garg, R., Khandekar, R.: Gradient descent with sparsification: an iterative algorithm for sparse recovery with restricted isometry property. In: Proceedings of the 26th Annual International Conference on Machine Learning, pp. 337–344. ACM (2009)
13. Gross, D., Liu, Y.-K., Flammia, S.T., Becker, S., Eisert, J.: Quantum state tomography via compressed sensing. Phys. Rev. Lett. **105**(15), 150401 (2010)
14. Huber, P.J.: Robust Statistics. John Wiley and Sons (1981)
15. Jain, P., Meka, R., Dhillon, I.S.: Guaranteed rank minimization via singular value projection. In: Advances in Neural Information Processing Systems, pp. 937–945 (2010)
16. Jain, P., Netrapalli, P., Sanghavi, S.: Low-rank matrix completion using alternating minimization. In: Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing, pp. 665–674. ACM (2013)
17. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
18. Koren, Y., Bell, R., Volinsky, C.: Matrix factorization techniques for recommender systems. Computer **8**, 30–37 (2009)
19. Markovsky, I., Usevich, K.: Low Rank Approximation. Springer, London (2012). https://doi.org/10.1007/978-1-4471-2227-2
20. Mount, J.: Approximation by orthogonal transform (2014). https://www.winvector.gitbuh.io/xDrift/orthApprox.pdf. Accessed 05 Sept 2018
21. Nesterov, Yu.: A method of solving a convex programming problem with convergence rate o(1/sqr(k)). Soviet Mathematics Doklady **27**, 372–376 (1983)
22. Nesterov, Yu.: Smooth minimization of non-smooth functions. Math. Program. **103**(1), 127–152 (2005)
23. Owen, A.B.: A robust hybrid of lasso and ridge regression. Contemp. Math. **443**(7), 59–72 (2007)
24. Park, D., Kyrillidis, A., Caramanis, C., Sanghavi, S.: Finding low-rank solutions via nonconvex matrix factorization, efficiently and provably. arXiv preprint arXiv:1606.03168 (2016)
25. Picci, G.: Stochastic realization theory. In: Mathematical System Theory, pp. 213–229. Springer, Heidelberg (1991). https://doi.org/10.1007/978-3-662-08546-2_12
26. Recht, B., Fazel, M., Parrilo, P.A.: Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization. SIAM Rev. **52**(3), 471–501 (2010)
27. Resnick, P., Varian, H.R.: Recommender systems. Commun. ACM **40**(3), 56–58 (1997)
28. Sun, R., Luo, Z.-Q.: Guaranteed matrix completion via non-convex factorization. IEEE Trans. Inform. Theory **62**(11), 6535–6579 (2016)

29. Sutskever, I., Martens, J., Dahl, G., Hinton, G.: On the importance of initialization and momentum in deep learning. In: International Conference on Machine Learning, pp. 1139–1147 (2013)
30. Tibshirani, R.: Regression shrinkage and selection via the lasso. J. R. Stat. Soc. Ser. B (Methodological) **58**(1), 267–288 (1996)
31. Tu, S., Boczar, R., Simchowitz, M., Soltanolkotabi, M., Recht, B.: Low-rank solutions of linear matrix equations via procrustes flow. In: International Conference on Machine Learning, pp. 964–973 (2016)
32. Tu, W., et al.: M-estimation in low-rank matrix factorization: a general framework. In: 2019 IEEE International Conference on Data Mining (ICDM), pp. 568–577 (2019)
33. Christopher JCH Watkins and Peter Dayan: Q-learning. Mach. Learn. **8**(3–4), 279–292 (1992)
34. Witten, I.H., Frank, E., Hall, M.A., Pal, C.J.: Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann (2016)
35. Zuyuan Yang, Y., Zhang, W.Y., Xiang, Y., Xie, S.: A fast non-smooth nonnegative matrix factorization for learning sparse representation. IEEE Access **4**, 5161–5168 (2016)
36. Zhang, J., Mitliagkas, I.: Yellowfin and the art of momentum tuning. arXiv preprint arXiv:1706.03471 (2017)
37. Zhao, T., Wang, Z., Liu, H.: Nonconvex low rank matrix factorization via inexact first order oracle. In: Advances in Neural Information Processing Systems (2015)
38. Zhu, R., Niu, D., Li, Z.: Robust web service recommendation via quantile matrix factorization. In: INFOCOM 2017-IEEE Conference on Computer Communications, IEEE, pp. 1–9. IEEE (2017)