Low Rank and Sparse Decomposition for Image and Video Applications

Nematollah Zarmehi, Student Member, IEEE, Arash Amini, Senior Member, IEEE, and Farokh Marvasti, Senior Member, IEEE

Abstract—The matrix decomposing into a sum of low-rank and sparse components has found extensive applications in many areas including background extraction in video surveillance, computer vision, and medical imaging. In this paper, we propose a new algorithm for recovery of low rank and sparse components of a given matrix. We have also proved the convergence of the proposed algorithm. The simulation results with synthetic and real signals such as image and video signals indicate that the proposed algorithm has a better performance with lower run-time than the conventional methods.

Index Terms—Background modeling, Gradient Projection (GP), low rank recovery, sparse, smoothed ℓ₀ norm, video, video surveillance.

I. INTRODUCTION

THE decomposition of a matrix into low-rank and sparse (sometimes referred to as noise) components is used in various applications such as background extraction in video surveillance [1], [2], removing shadows and specularities from face images [3]–[5], data compression [6], matrix rigidity in computational complexity [7], link prediction in social networks [8], and subspace clustering [9]. A common mathematical model in these applications is to assume a low-rank matrix $L_o \in \mathbb{R}^{m \times n}$ for the signal of interest which is corrupted by a sparse (noise) matrix $E_o \in \mathbb{R}^{m \times n}$:

$$Y = L_o + E_o, \quad Y \in \mathbb{R}^{m \times n},$$

where $Y$ is the available data matrix. The recovery problem here refers to extracting $L_o$ and $E_o$ from their mixture $Y$. Oftentimes, the primary goal is to achieve an estimate of the low-rank component $L_o$; however, the sparse component $E_o$ might also carry some information such as the moving objects in the video surveillance example. As the noise component is not necessarily Gaussian, the conventional recovery techniques based on Tikhonov regularizers [10] are no longer applicable here. Further, we assume that the rank of $L_o$ and the sparsity number of $E_o$ are unknown.

A. Approach

Considering the data model in (1), this paper proposes an algorithm for recovery of low rank and noise components of a given matrix $Y$ by suggesting an optimization problem that approximates the following optimization problem:

$$P_0 : \argmin_{L, E} \text{rank}(L) + \lambda \|E\|_0,$$

subject to $Y = L + E$, \hspace{1cm} (2)

where rank of $L$ is number of non-zero Singular Values (SVs) of $L$, $\|E\|_0$ denotes the entry-wise ℓ₀-(pseudo) norm of the noise matrix $E$. The noise component $E$ is assumed to be sparse; however, no further information about its support set or the distribution of its non-zero elements is available.

B. Related Works

Principal Component Analysis (PCA) [11] is possibly the most popular technique to extract the low-rank component when the noise component is Gaussian. It is known that the solution to

$$\min_L \|E\|_F,$$

subject to $\text{rank}(L) \leq r,$

$$Y = L + E,$$

is found by the first $r$ principal components of $Y$; the computational approach is via finding the Singular Value Decomposition (SVD) of $Y$ and zeroing the min($m, n) - r$ right hand SVs. In (3), $\|\cdot\|_F$ denotes the Frobenius norm. PCA works well for high dimensional data that approximately form a low dimensional linear subspace. A typical example is when a low dimensional set of data is contaminated with a high dimensional Gaussian noise. Although the outcome is of high dimensions, the energy is mainly concentrated in the subspace of the original data. PCA gets the optimal solution when the noise components (the entries of $E$) are i.i.d. Gaussian. Moreover, in this method, $r$ should be known. However, it is also known that PCA fails to recover the low-rank component when just a single entry is substantially perturbed (sparse noise). Indeed, this type of noise is very common in real applications dealing with image and video signals. There are several techniques to make PCA robust against sparse noise [12]–[16]. The Principal Component Pursuit (PCP) introduced in [3] is one of the successful methods that can deal with sparse perturbations with unknown support and arbitrarily large magnitudes. Under certain conditions, the solution to the following convex optimization is guaranteed to yield the original low-rank component:

$$P_1 : \argmin_{L, E} \|L\|_* + \lambda \|E\|_1,$$

subject to $Y = L + E$, \hspace{1cm} (4)
where \( \| . \|_2 \) is the nuclear norm and \( \lambda > 0 \) is a weighting factor balancing the sparsity and the rank. In this optimization problem, the rank and the \( \ell_0 \)-norm of \( \mathcal{P}_0 \) in (2) are approximated by the nuclear norm and the \( \ell_1 \)-norm, respectively. For PCP to be successful, the rank of \( \mathbf{L}_o \) and the sparsity level of \( \mathbf{E}_o \) need to be sufficiently small; furthermore, \( \mathbf{L}_o \) and \( \mathbf{E}_o \) should be incoherent. An Augmented Lagrange Multiplier (ALM) algorithm is proposed to solve \( \mathcal{P}_1 \) [17], [18]. Interestingly, it is shown in [19] that \( \mathcal{P}_1 \) with suitable \( \lambda \) can recover the low-rank component even when the noise is not so sparse. Although \( \mathcal{P}_0 \) is a non-convex optimization problem and finding its minimizer is difficult task, it might lead to a suitable result in many instances that \( \mathcal{P}_1 \) fails. For example, one can simply verify that \( \mathcal{P}_1 \) fails to recover the original low rank component when

\[
\begin{align*}
\mathbf{Y} &= \begin{bmatrix} 0 & 2 & 3 \\
2 & 0 & 6 \\
3 & 6 & 0 
\end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\
2 & 4 & 6 \\
3 & 6 & 9 
\end{bmatrix} + \begin{bmatrix} -1 & 0 & 0 \\
0 & -4 & 0 \\
0 & 0 & -9 
\end{bmatrix} = \mathbf{L}_o + \mathbf{E}_o
\end{align*}
\]

Apart from convex relaxation approach [20]–[23], non-convex surrogates of low-rank functions are also introduced in the literature [24]–[26]. In [27], an empirical Bayesian approach is proposed for non-convex rank minimization using a variational approximation and marginalization. In this method, the log-determinant function is used to replace the rank function. Moreover, this method assumes a direct Gaussian prior on low-rank matrix. A low-rank matrix factorization model for matrix completion is proposed in [28], [29] whose implementation is called Low-rank Matrix Fitting (LMaFit). The LMaFit algorithm follows a non-linear and non-convex model which is solved using a non-linear successive over-relaxation algorithm. An Alternating Direction Method of Multipliers (ADMM) for matrix separation based on low-rank factorization is proposed in [30] which uses the LMaFit algorithm. This method is limited to the problems where the sparse matrix does not dominate the low-rank matrix in magnitude.

In this paper, unlike the common approach of approximating the rank with the nuclear norm and relaxing the \( \ell_0 \)-norm with the \( \ell_1 \)-norm, we adopt a smoothing technique which has been recently used for sparse representation [31], [32]. This technique consists of sequentially approximating the \( \ell_0 \)-norm with the family of smoothed \( \ell_0 \)-norm functions [33]. We explain more details in Section II-B. We also analytically study the convergence of the proposed algorithm and compare it with the state-of-the-art methods under different scenarios and apply it to real applications such as background modeling in video surveillance and removing shadows and specularities from the face images. We shall show by simulations that the proposed algorithm has better performance and lower run-time than the previous works. We believe that our double-smoothing technique as well as the mathematical analysis is rather new.

C. Outline

The paper is organized as follows: In Section II, we provide some preliminaries regarding the employed smoothing technique. In Section III, we explain our proposed algorithm for the low-rank/sparse decomposition. A discussion on the uniqueness of the solution and the proof of convergence of the proposed method are provided in Section IV. The experimental results are presented in Section V. We test the proposed algorithm on synthetic and real data such as face images and video frames. Finally, Section VI concludes the paper.

II. PRELIMINARIES

A. Notations

Throughout the paper, all the scalar variables, column vectors, matrices, and sets will be denoted by italic lower-case, boldface lower-case, boldface upper-case, and black-board-font upper-case letters, respectively. For example, \( x, x, \mathbf{X}, \) and \( \mathbb{X} \) are scalar, vector, matrix, and set, respectively. The elements of vectors and matrices are denoted by subscripts; i.e., \( x_i \) is the \( i \)-th element of vector \( x \) and \( a_{i,j} \) is the element of matrix \( \mathbf{A} \) at the intersection of \( i \)-th row and \( j \)-th column. We also use \( [x_i]_{i=1}^{\infty} \) and \( [a_{i,j}]_{i=1,j=1}^{m,n} \) to denote vector \( x \) of size \( n \) and matrix \( \mathbf{A} \) of size \( m \times n \), respectively. A sequence is shown by \( \{x_i\}_{i=1}^{\infty} \), \( \mathbf{A}^T \) and \( \mathbf{A}^\otimes 2 \) denote the transpose and entry-wise square of matrix \( \mathbf{A} \), respectively. For a matrix \( \mathbf{A} \in \mathbb{R}^{m \times n} \), \( e^{x} \mathbf{A} \) indicates a matrix with entries \( \{e^{a_{i,j}}\}_{i=1,j=1}^{m,n} \). We shall use \( \mathbf{A} \odot \mathbf{B} \) and \( \mathbf{A} \boxtimes \mathbf{B} \) to denote the entry-wise multiplication and division of equi-size matrices \( \mathbf{A} \) and \( \mathbf{B} \), respectively. We further denote the Frobenius, nuclear, \( \ell_\infty \), \( \ell_1 \), and \( \ell_0 \) norms by \( \| x \|_F \), \( \| x \|_1 \), \( \| x \|_\infty \), and \( \| x \|_0 \), respectively. Finally, for a matrix \( \mathbf{A} \in \mathbb{R}^{m \times n} \), \( \text{vec}(\mathbf{A}) \) refers to the vector in \( \mathbb{R}^{mn} \) obtained by stacking the columns of the matrix \( \mathbf{A} \) on bottom of one another and \( \text{sum}(\mathbf{A}) \) is the sum of all entries of \( \mathbf{A} \).

B. The Family of Smoothed \( \ell_0 \)-norm Functions

Due to non-convexity and discontinuity, \( \ell_0 \)-norm minimization is computationally very challenging. The most popular remedy in problems that deal with \( \ell_0 \)-norm minimization is to replace the \( \ell_0 \)-norm with \( \ell_1 \)-norm. Another approach devised simultaneously in [31] and [32] is to approximate the \( \ell_0 \)-norm by a family of smooth functions that tend to the Kronecker delta function in the limit. Because of the smoothness of the approximating functions, the resulting cost function also becomes smooth. In this approach, a zero-mean Gaussian family of functions is used to approximate the \( \ell_0 \)-norm of a vector \( x \in \mathbb{R}^n \) as follows:

\[
\| x \|_0 = \sum_{i=1}^{n} (1 - \delta(x_i)) = n - \lim_{\delta \to 0} \sum_{i=1}^{n} f_\delta(x_i), \tag{6}
\]

where \( f_\delta(x) = e^{-\frac{x^2}{2\delta^2}} \) and \( \delta(\cdot) \) is the discrete delta function. Besides smoothness, the main property of \( f_\delta(\cdot) \) as an approximation of the Kronecker delta function is that

\[
f_\delta(x) \approx \begin{cases} 1 & |x| \ll \delta \\ 0 & |x| \gg \delta \end{cases} \tag{7}
\]

Obviously, the Gaussian function family is not the only option here. The family of \textit{triangular} functions, the family...
of truncated hyperbolic functions, and homographic functions of the form $\delta^2/(x^2 + \delta^2)$ are also considered for smoothly approximating the $l_0$-norm [33].

Below, we define a general family $\{f_\delta(\cdot)\}$ of functions that approximate the Kronecker delta function.

**Definition 1.** Let $f : \mathbb{R} \to [0, 1]$ be a smooth function that
1) is analytic, unimodal,
2) $f(x) = 1 \iff x = 0$, and
3) $\lim_{|x| \to \infty} f(x) = 0$.
then, we define the family $\{f_\delta(\cdot)\}$ by $f_\delta(x) = f(x/\delta)$.

III. PROPOSED ALGORITHM

A. Main Idea

We recall that our goal is to recover the low-rank matrix $L_0$ and the noise component $E_0$ from their mixture $Y$, by finding the solution to the $P_0$ problem in (2). We approximate the rank function and the $l_0$-norm using a family $\{f_\delta(\cdot)\}$ of smooth functions. For the rank of a matrix $L \in \mathbb{R}^{m \times n}$, we use

$$\text{rank}(L) = \|\sigma(L)\|_0 \approx R_\delta(L) = h_\delta(\sigma(L)) = q - \sum_{i=1}^{q} f_\delta(\sigma_i(L)),$$  

(8)

where $\sigma(L) = [\sigma_1(L), \cdots, \sigma_q(L)]^T$ with $\sigma_i(L)$ being the $i$-th largest SV of $L$, $h_\delta(a_{i+1}) = \sum_{j=1}^{i} (1 - f_\delta(a_j))$, and $q = \min(m, n)$. Similarly, the sparsity level ($l_0$-norm) of the matrix $E \in \mathbb{R}^{m \times n}$ can be approximated as

$$\|E\|_0 \approx S_\delta(E) = h_\delta(\text{vec}(E)) = mn - \sum_{i=1}^{mn} f_\delta(|\text{vec}(E)|_i).$$  

(9)

As (8) and (9) indicate, $R_\delta(L)$ is the smoothed rank function which estimates the number of non-zero SVs of $L$, while $S_\delta(E)$ is the smoothed $l_0$-norm function that estimates the number of non-zero entries of $E$. Now, we approximate the problem (2) as follows:

$$\mathcal{G} : \text{argmin}_{L, E} G(L, E, \delta) = R_\delta(L) + \lambda S_\delta(E),$$

subject to $Y = L + E.$

(10)

We use the Gradient Projection (GP) [34] to solve this optimization problem. We suggest updating and projecting onto the feasible set as follows:

$$\mathcal{G} : \begin{cases} L \leftarrow L - \mu_i \nabla_L G(L, E, \delta) \\
E \leftarrow Y - L \\
E \leftarrow E - \rho_i \nabla_E G(L, E, \delta) \\
L \leftarrow E - Y - E \end{cases},$$

(11)

where $\mu_i$ and $\rho_i$ are the step sizes of the $i$-th iteration of GP algorithm.

As $\delta \to 0$, $R_\delta(\cdot)$ and $S_\delta(\cdot)$ result in better approximations for the rank and sparsity. But for small values of $\delta$, they have many local minima and the GP algorithm may get trapped in one of the local minima. Hence, we start with a large $\delta$ and decrease it at each iteration of the algorithm. The output of the $i$-th iteration will be used as the initial point for the $(i+1)$-th iteration. This is the Graduated Non-Convexity (GNC) approach for non-convex optimization [35].

**Remark 1.** One can select a sequence of decreasing $\delta_i$ to get a better approximation for the rank and sparsity. We decrease $\delta$ during the iterations of the algorithm by $\delta_i = \alpha \delta_{i-1}$, where $\alpha \in [0.5, 1]$ is the decreasing factor and $\delta_1$ is set to $\|\sigma(L_0)\|_\infty$ where $g > 1$ is a constant.

**Remark 2.** In our algorithm, we set the initial point as $(L_0, E_0) = (\lambda_1 Y, Y)$ because it satisfies the Karush–Kuhn–Tucker (KKT) conditions for the optimization problem $\mathcal{G}$ when $\delta \to \infty$. To show this, suppose that the SVD of $L$ is $L = U \text{diag}(\sigma_1, \cdots, \sigma_q) Y^T$. Using the method of Lagrange multipliers, we have the following Lagrangian function

$$\mathcal{L}(L, E, G) = R_\delta(L) + \lambda S_\delta(E) - \sum(G \odot (Y - L - E)),$$

(12)

where $G$ is the Lagrange multiplier. The gradients of the Lagrangian function at $(L, E, G)$ are as follows:

$$\begin{align*}
\nabla_L \mathcal{L}(L, E, G) &= -\delta^2 U \text{diag}(\{f'_\delta(\sigma_j)\}_{j=1}^q) Y^T + \delta^2 G \\
\nabla_E \mathcal{L}(L, E, G) &= -\lambda \delta^2 f_\delta(E) + \delta^2 G \\
\nabla_G \mathcal{L}(L, E, G) &= Y - L - E
\end{align*}$$

Moreover, note that for both homographic and Gaussian smoothed functions we have

$$\lim_{\delta \to \infty} \delta^2 f_\delta(x) = -\kappa x,$$

(14)

where $\kappa = 2, 1$ for the homographic and Gaussian families, respectively. Therefore, when $\delta \to \infty$, we have

$$\begin{align*}
\nabla_L \mathcal{L}(L, E, G) &= \kappa L + \delta^2 G \\
\nabla_E \mathcal{L}(L, E, G) &= \lambda E + \delta^2 G \\
\nabla_G \mathcal{L}(L, E, G) &= Y - L - E
\end{align*}$$

(15)

One can easily check that for $(L, E, G) = (\lambda_e Y, Y - \frac{\lambda_e}{\sqrt{1+\lambda_e^2}} Y)$, the gradient terms will be zero.

**Remark 3.** We use $\|\hat{L}_i - \hat{L}_{i-1}\|_F \leq \epsilon$ as the main stop criterion, where $\hat{L}_i$ and $\hat{L}_{i-1}$ are, respectively, the estimated low rank matrices at $i$-th and $(i-1)$-th iterations, and $\epsilon$ is a predetermined parameter. For the inner loop, we set a fixed parameter $K$ as the maximum number of iterations. In Section V, we show that only 3 or 4 iterations is enough for the internal loop.

B. The Proposed Algorithm

Now it is time to present the algorithm. The main algorithm is shown in Algorithm 1. We name it as the Low rank and Sparse Decomposition using Smoothed $l_0$-Norm (LSD-SN).

One may use some families of smoothed $l_0$-norm functions in lines 15 and 17 of the Algorithm 1. In case we are using the family of homographic smoothed functions, these two lines would be as follows:

$$L \leftarrow L - \mu_i \text{diag} \left( \left\{ \frac{2\sigma_j \delta^4}{(\delta_j^2 + \delta^2)^2} \right\}_{j=1}^q \right) V^\top,$$

(16)

$$E \leftarrow E - \rho_i \lambda E \odot \delta^4 \odot (E \Delta^2 + \delta^2) \Lambda^2,$$
Algorithm 1 LSD-SN

1: input: 
2: Data matrix $Y \in \mathbb{R}^{m \times n}$
3: Stopping threshold $\epsilon$
4: Decreasing factor $\alpha$
5: Step size constants $\gamma_\mu, \gamma_\rho$
6: Maximum number of iterations of the inner loop $K$
7: initialization:
8: $i \leftarrow 1$, $e \leftarrow \infty$, $\delta_1 \leftarrow 4\|\sigma(\hat{L}_0)\|_\infty$
9: $\lambda \leftarrow 1/\sqrt{\max(m, n)}$, $\hat{L}_0 \leftarrow \frac{1}{1+\lambda} Y$
10: while $e > \epsilon$ do
11: $L_i \leftarrow \hat{L}_{i-1}$, $\mu_i \leftarrow \gamma_\mu \delta_i^2$, $\rho_i \leftarrow \gamma_\rho \delta_i^2$
12: $\alpha \delta_i \leftarrow \left| f_{\delta_i^{-1}}(\lambda/n^2) \right|$, $\beta \delta_i \leftarrow \left| f_{\delta_i^{-1}}(1/mn^2) \right|$
13: for $k = 1 : K$ do
14: $[U, \Sigma, V] \leftarrow \text{svd}(L_i)$
15: $L_i \leftarrow L_i - \mu_i \nabla L_i R_{\delta_i}(L_i)$
16: $E_i \leftarrow Y - L_i$
17: $E_i \leftarrow E - \rho_i \lambda \Sigma_{\delta_i}(E_i)$
18: $E_i \leftarrow T_{\delta_i}(E_i)$
19: $L_i \leftarrow Y - E_i$
20: $L_i \leftarrow D_{\alpha \delta_i}(L_i)$
21: end for
22: $\hat{E}_i \leftarrow -E_i$
23: $\hat{L}_i \leftarrow L_i - \mu_i \nabla L_i R_{\delta_1}(L_i)$
24: $e \leftarrow \|L_i - \hat{L}_{i-1}\|_F$
25: $i \leftarrow i + 1$
26: $\delta_i \leftarrow \alpha \delta_{i-1}$
27: end while
28: $\hat{L}_0 \leftarrow \hat{L}_i$, $\hat{E}_0 \leftarrow Y - \hat{L}_0$
29: return $\hat{L}_0, \hat{E}_0$

and if we are using the family of Gaussian smoothed functions, then we have

$$L \leftarrow L - \mu_i \text{Udiag} \left( \left\{ \sigma_j e^{-\frac{\sigma_j^2}{2\sigma_j^2}} \right\}_{j=1}^q \right) V^T,$$

$$E \leftarrow E - \rho_i \lambda \Sigma \odot e^{\lambda(\frac{Y - e \hat{E}_i^2}{2\lambda})}.\tag{17}$$

To distinguish between these two families, we name the first one as LSD-HSN and the second one as LSD-GSN, where "H" and "G" denote homographic and Gaussian, respectively.

To better approximate the $\ell_0$-norm, we threshold the updated matrices $E$ and $L$ in lines 18 and 20 of the Algorithm 1, respectively. For a matrix $X \in \mathbb{R}^{m \times n}$ with SVD $X = U \Sigma V^T$, the operators $D_{\epsilon}(X)$ and $T_{\epsilon}(X)$ are defined as $D_{\epsilon}(X) = U \text{diag}(\Sigma) V^T$, where $r(X) = \max(x_{i,j} - \tau, 0))_{i=1,j=1}^{m,n}$. This thresholding expedites the convergence. An example of using and banning this operator is shown in Fig. 1 which shows the SNR versus the number of iterations in the outer loop.

IV. CONVERGENCE ANALYSIS

In this section, we provide the proof of convergence of the proposed algorithm. In Section III, several considerations were made to avoid GP algorithm getting trapped in local minima. Hence, it is assumed that the internal loop converges to the global minimum.

First, consider the following definitions and assumptions.

Fig. 1. An example of using and banning the thresholding operator ($n = 500$, $r = 0.05 \times n$, and $k = 0.3 \times n^2$). The number of iterations refer to the outer loop.

**Definition 2.** For $\epsilon > 0$, define the following set

$$S_{\epsilon} = \left\{ (L, E) \left| (L, E) \in \text{arginf}_{A,B} \text{rank}(A) + \lambda \|B\|_0 \right. \right.$$

subject to $\|A + B - Y\|_F \leq \epsilon$.\]

Also, define $S_0$ as

$$S_0 = \left\{ (L, E) \left| (L, E) \in \text{arginf}_{A,B} \text{rank}(A) + \lambda \|B\|_0 \right. \right.$$

subject to $\|A + B - Y\|_F = 0$.

**Definition 3.** For each $S_{\epsilon}$, define $v(S_{\epsilon}) \triangleq \text{rank}(L) + \lambda \|E\|_0$ which is constant for all $(L, E) \in S_{\epsilon}$.

**Assumption 1.** Assume that $S_{\epsilon}$ is bounded by the Frobenius norm. Moreover, assume that $S_0$ has a unique solution $(L_0, E_0)$, i.e., $S_0 = \{ (L_0, E_0) \}$.

**Remark 4.** For $S_{\epsilon}$, we have

$$\forall \epsilon \geq 0 : v(S_{\epsilon}) \leq n + \lambda n^2, \quad \text{and} \quad 0 < \epsilon_1 \leq \epsilon_2 : v(S_{\epsilon_1}) \geq v(S_{\epsilon_2}). \tag{18}$$

The last inequality is true since the ball defined with $\epsilon_2$ contains the ball defined with $\epsilon_1$; and if the feasible set in an optimization problem is enlarged from a ball of radius $r_1$ to a ball of radius $r_2$, the cost associated to the optimal point in the feasible set cannot get worse. Therefore, the limit of $v(S_{\epsilon})$ as $\epsilon \rightarrow 0$ exists. Let $\lim_{\epsilon \rightarrow 0} v(S_{\epsilon}) = v_*$. Notice that $v(.)$ can get finite values. Hence,

$$\exists \epsilon_T > 0, \forall 0 < \epsilon \leq \epsilon_T : v(S_{\epsilon}) = v_* \Rightarrow \forall \epsilon_1, \epsilon_2, \text{ such that } 0 < \epsilon_1 \leq \epsilon_2 \leq \epsilon_T : S_{\epsilon_1} \subseteq S_{\epsilon_2},$$

that means for $0 < \epsilon \leq \epsilon_T$, $S_{\epsilon}$ are nested sets.

In Assumption 1, we assumed that $S_{\epsilon}$ is bounded. In the following Lemma, we prove that it is also closed, thus, it is compact.

**Lemma 1.** The set $S_{\epsilon}$ defined in Definition 2 is compact.
Proof. We show that the limit of every convergent sequence in $S_e$ is in $S_e$. Let $\{(L_i, E_i)\}_{i=1}^{\infty}$ be converging sequence (by the Frobenius norm) and let $(\hat{L}, \hat{E})$ be the limit. Further, let
\[ \text{rank}(\hat{L}) = r \text{ and } \|\hat{E}\|_0 = k. \]
For all $(L_i, E_i)$ in the sequence, we have $\|L_i + E_i - Y\| \leq \epsilon$. As $(\hat{L}, \hat{E})$ is the limit of sequence $\{(L_i, E_i)\}_{i=1}^{\infty}$, the same inequality holds for $(\hat{L}, \hat{E})$, i.e., $\|\hat{L} + \hat{E} - Y\| \leq \epsilon$. To prove that $(\hat{L}, \hat{E}) \in S_e$, it is sufficient to show that it is the minimizer of $\text{rank}(\cdot) + \|L\|_0$. The sparsity number of $\hat{E}$ is $k$. Since $\{E_i\}_{i=1}^{\infty}$ converges to $\hat{E}$ element-wise, there exists an integer $i_k$ such that for all $i \geq i_k$, $E_i$ is non-zero at the support elements of $\hat{E}$. Hence, $E_i$ has at least $k$ non-zero entries. A similar statement holds for the rank of $\{L_i\}_{i=1}^{\infty}$, as sorted singular values are continuous functions of the entries: since $\{L_i\}_{i=1}^{\infty}$ converges to $\hat{L}$, there exists an integer $i_r$ such that, for all $i \geq i_r$, $L_i$ has at least $r = \text{rank}(\hat{L})$ SVs. Therefore,
\[ \forall i \geq \max(i_k, i_r) : \text{rank}(L_i) + \lambda \|E_i\|_0 \geq r + \lambda k. \quad (19) \]
On one hand, $v(S_e) = \text{rank}(L_i) + \lambda \|E_i\|_0$ is the minimum cost value among all pairs of $(L, E)$ that satisfy $\|L + E - Y\| \leq \epsilon$. On the other hand, the pair $(\hat{L}, \hat{E})$ satisfies the constraint while its cost does not exceed $v(S_e)$. Consequently, we shall have $r + \lambda k = v(S_e)$, which implies that $(\hat{L}, \hat{E}) \in S_e$. This completes the proof.

Lemma 2. The intersection of $S_e$ for $0 < \epsilon \leq \epsilon_T$ consists of a single element; more precisely, $\bigcap_{0 < \epsilon \leq \epsilon_T} S_e = S_0$.

Proof. According to Remark 4 and Lemma 1, for all $0 < \epsilon \leq \epsilon_T$, $S_e$ are nonempty nested compact sets. As a result of nested compact sets theorem [36, Theorem 2.3.6], the intersection is not empty. Let $\hat{S} = \bigcap_{0 < \epsilon \leq \epsilon_T} S_e$ and $(\hat{L}, \hat{E})$ be an arbitrary pair in $S_e$; this implies that $(\hat{L}, \hat{E}) \in S_e$, or alternatively, $\|\hat{L} + \hat{E} - Y\| \leq \epsilon$, for all $0 < \epsilon \leq \epsilon_T$. Therefore, we conclude that $\|\hat{L} + \hat{E} - Y\| = 0$. In addition, according to Remark 4, we have $v_e = v(\hat{S}) = v(S_{e_0}) \leq v(S_0)$. However, $v(S_0)$ is the minimum cost value among all pairs of $(L, E)$ that fulfill $\|L + E - Y\| = 0$. This proves that $v_e = v(S_0)$ and $(\hat{L}, \hat{E})$ is indeed the unique element of $S_0$. In summary:
\[ (\hat{L}, \hat{E}) = \bigcap_{0 < \epsilon \leq \epsilon_T} S_e = S_0. \]

Next, we show that the output $(\hat{L}_\delta, \hat{E}_\delta)$ of our algorithm at the iteration corresponding to the value $\delta$, belongs to $S_{e_\delta}$ for some $\epsilon_\delta > 0$. Furthermore, when $\delta \rightarrow 0$ we have that $\epsilon_\delta \rightarrow 0$. Since $\{(\hat{L}_\delta, \hat{E}_\delta)\}_\delta$ converge to the optimal solution $(L_0, E_0)$ as $\delta$ tends to 0.

Theorem 1. Let $(\hat{L}_\delta, \hat{E}_\delta)$ be the minimizer of the cost $G(L, E, \delta)$ with $\lambda = 1/n^2$, that is subject to the thresholding effects described in lines 18 and 20 of Algorithm 1. Then, for
\[ \alpha_\delta = \left| f_\delta^{-1}(1/n^2) \right|, \quad \beta_\delta = \left| f_\delta^{-1}(1/n^3) \right|, \quad \epsilon_\delta = \alpha_\delta n + \beta_\delta n^2 \]
we know that $(\hat{L}_\delta, \hat{E}_\delta) \in S_{e_\delta}$. Moreover, $\lim_{\delta \rightarrow 0} \epsilon_\delta = 0$.

Proof. Let $(L_\delta, E_\delta)$ be the minimizer of $R_\delta(.) + \lambda S_\delta(.)$ and $(L_0, E_0)$ be the unique solution of $S_0$ with $\text{rank}(L_0) = r_0$ and $\|E_0\|_0 = k_0$; Therefore,
\[ R_\delta(L_\delta) + \lambda S_\delta(E_\delta) \leq R_\delta(L_0) + \lambda S_\delta(E_0) \leq r_0 + \lambda k_0, \quad (20) \]
where the last inequality follows from the fact that $R_\delta(L) \leq \text{rank}(L)$ and $S_\delta(E) \leq E$, for all $L$ and $E$.

Assume that the SVs of $L_\delta$ and the entries of $E_\delta$ are sorted as $\sigma_1 \geq \cdots \geq \sigma_n$ and $\epsilon_1 \leq \cdots \leq \epsilon_n$, respectively. Define $\|a\|_3 = \{i \in \{1, \ldots, n\} | \sigma_i > \alpha_\delta \}$ and $\|b\|_3 = \{i \in \{1, \ldots, n^2\} | \epsilon_i > \beta_\delta \}$. This results in
\[ R_\delta(L_\delta) + \lambda S_\delta(E_\delta) = n - \sum_{i=1}^{n} f_\delta(\sigma_i) + \lambda n^2 - \lambda \sum_{i=1}^{n^2} f_\delta(\epsilon_i) \]
\[ = n - \sum_{i \in \|a\|_3} f_\delta(\sigma_i) - \sum_{i \not\in \|a\|_3} f_\delta(\sigma_i) \]
\[ \leq n - \frac{\lambda n^2}{\lambda n^2} \leq n - \|b\|_3 \]
\[ < \|a\|_3 + \lambda \|b\|_3 < 2 \frac{n}{n}. \]
Since $R_\delta(L_\delta) + \lambda S_\delta(E_\delta) \leq r_0 + \lambda k_0$, we have
\[ \|a\|_3 + \lambda \|b\|_3 < r_0 + \lambda k_0 + \frac{2}{2 \frac{n}{n}}, \]
or $\|a\|_3 + \lambda \|b\|_3 < r_0 + \lambda k_0$. Consequently,
\[ \|a\|_3 + \lambda \|b\|_3 < r_0 + \lambda k_0. \]
Note that both $\|a\|_3 + \lambda \|b\|_3 < r_0 + \lambda k_0$ are integers. This reveals that $A$ is also integer. Since $\frac{2}{2 \frac{n}{n}} < 1$ for $n > 2$, we conclude that $A < 0$, or
\[ \|a\|_3 + \lambda \|b\|_3 \leq r_0 + \lambda k_0. \]
If we threshold the SVs of $L_\delta$ and the entries of $E_\delta$ by $\alpha_\delta$ and $\beta_\delta$, respectively, we get new matrices $\tilde{L}_\delta$ and $\tilde{E}_\delta$ for which we have
\[ \|\sigma(\tilde{L}_\delta)\|_0 + \lambda \|\tilde{E}_\delta\|_0 = \|a\|_3 + \lambda \|b\|_3 \leq r_0 + \lambda k_0. \quad (21) \]
Because of the thresholding operators, the sum of $\tilde{L}_\delta$ and $\tilde{E}_\delta$ no longer equal to $Y$. Hence, we have an error:
\[ U_\delta = \tilde{L}_\delta + \tilde{E}_\delta - Y. \]
In the thresholding stage, we threshold at most $n$ SVs of $L_\delta$ and $n^2$ entries of $E_\delta$. Therefore, $\|U_\delta\|_0 \leq \alpha_\delta n + \beta_\delta n^2 = \epsilon_\delta$. If $\epsilon_\delta < \epsilon_T$, (21) demonstrates that the cost of $(\tilde{L}_\delta, \tilde{E}_\delta)$ is indeed optimal and equal to $v_e$, hence, $(\tilde{L}_\delta, \tilde{E}_\delta) \in S_{e_{\delta}}$. It should be mentioned that based on Definition 2, for any $\zeta > 0$, we can choose sufficiently small $\delta$ such that
\[ \alpha_\delta = \left| f_\delta^{-1}(\lambda/n^2) \right| < \zeta \quad \Rightarrow \quad \lim_{\delta \rightarrow 0} \alpha_\delta = 0 \quad \text{and} \quad \beta_\delta = \left| f_\delta^{-1}(1/n^3) \right| < \zeta \quad \Rightarrow \quad \lim_{\delta \rightarrow 0} \beta_\delta = 0. \]
This implies that $\epsilon_\delta \to 0$ as $\delta \to 0$, as well as $\epsilon_\delta \leq \epsilon_T$ for sufficiently small $\delta$ values.

V. NUMERICAL EXPERIMENTS AND APPLICATIONS

In this section, we present numerical experiments and compare the proposed algorithm empirically with some well known algorithms. In all numerical experiments, we use both homographic and Gaussian smoothed functions. As mentioned before, we use LSD-HSN for the former method and use LSD-GSN for the latter one. All simulations are done by MATLAB R2015a on Intel(R) Core(TM) i7-5960X @ 3GHz with 32GB-RAM. We use $\text{SNR}(L_o, \hat{L}_o) = 20 \log_{10} \left( \frac{\|L_o\|_F}{\|L_o - \hat{L}_o\|_F} \right)$ in dB as the evaluation criterion.

A. Parameter Effects

First, we investigate the effect of the parameters $\epsilon$, $\alpha$, and $K$ on the performance of the proposed algorithm. We produce $r$-rank square matrices of dimensions $m = n = 100$ as the product of two $n \times r$ matrices, i.e., $L_o = AB^T$ where both $A$ and $B$ are independently sampled from a $\mathcal{N}(0, 1/n)$ distribution.

The noise component $E_o$ has a support size of $k$ generated uniformly at random whose values are independently selected from the set $\{-1, +1\}$ with equal probabilities. Two random problems are considered in the following experiments with different values of rank and sparsity.

1) Effect of $\epsilon$: To investigate the effect of $\epsilon$, we vary $\epsilon$ from $10^{-2}$ to $10^{-16}$ and the recovered SNR is shown in Fig. 2. We fix other parameters as $\alpha = 0.8$ and $K = 3$. As expected, the proposed algorithm could better recover the low rank matrix by decreasing $\epsilon$.

2) Effect of $K$: To investigate the number of iterations of the internal loop on the performance of the proposed algorithm, we change $K$ from 1 to 8 and fix other parameters as $\epsilon = 10^{-12}$ and $\alpha = 0.8$. The results are shown in Fig. 3. It can be seen that only 3 or 4 iterations is enough for the internal loop.

3) Effect of $\alpha$: Finally, the effect of decreasing factor $\alpha$ is investigated. For this purpose, we set $\epsilon = 10^{-12}$ and $K = 3$ and change $\alpha$ from 0.1 to 0.9. Fig. 4 shows the results. It can be observed that the performance gets better as $\alpha$ is closer to 1.

In the following, we have taken into consideration the effects of the above parameters to use of the proposed algorithm.

B. Exact Recovery

This subsection presents the numerical results to demonstrate exact recovery of the proposed algorithm. Similar to [3], we produce $r$-rank square matrices of dimensions $n = 500, 1000, \ldots, 3000$ as described in Subsection V-A. In this subsection, we consider $r = 0.05 \times n$. The noise component is also generated as described in Subsection V-A. Tables I and II show the results for $k = 0.05 \times n^2$ and $k = 0.1 \times n^2$, respectively. As it can be seen, the proposed algorithm could exactly recover the low rank and sparse components, however, because of finite precision we could not get $\text{SNR} = \infty$ dB.

C. Comparison

In this subsection, we compare the proposed algorithm with the Inexact Augmented Lagrange Multiplier (IALM) [17],
TABLE I

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Rank</th>
<th>Sparsity</th>
<th>SNR (dB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>$r$</td>
<td>$k$</td>
<td>LSD-HSN</td>
</tr>
<tr>
<td>300</td>
<td>150</td>
<td>900,000</td>
<td>265.2</td>
</tr>
<tr>
<td>2500</td>
<td>125</td>
<td>312,500</td>
<td>265.8</td>
</tr>
<tr>
<td>2000</td>
<td>100</td>
<td>200,000</td>
<td>256.1</td>
</tr>
<tr>
<td>1500</td>
<td>75</td>
<td>112,500</td>
<td>255.2</td>
</tr>
<tr>
<td>1000</td>
<td>50</td>
<td>50,000</td>
<td>249.5</td>
</tr>
<tr>
<td>500</td>
<td>25</td>
<td>12,500</td>
<td>267.9</td>
</tr>
</tbody>
</table>

TABLE II

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Rank</th>
<th>Sparsity</th>
<th>SNR (dB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>$r$</td>
<td>$k$</td>
<td>LSD-HSN</td>
</tr>
<tr>
<td>300</td>
<td>150</td>
<td>900,000</td>
<td>265.2</td>
</tr>
<tr>
<td>2500</td>
<td>125</td>
<td>312,500</td>
<td>265.8</td>
</tr>
<tr>
<td>2000</td>
<td>100</td>
<td>200,000</td>
<td>256.1</td>
</tr>
<tr>
<td>1500</td>
<td>75</td>
<td>112,500</td>
<td>255.2</td>
</tr>
<tr>
<td>1000</td>
<td>50</td>
<td>50,000</td>
<td>249.5</td>
</tr>
<tr>
<td>500</td>
<td>25</td>
<td>12,500</td>
<td>267.9</td>
</tr>
</tbody>
</table>

LSSD [37], SpaRCS [38], LRSD-TNNSR [39], and LMaFit [30]. The low rank and sparse matrices are generated in the same way as explained in Subsection V-A. The main drawback of the LMaFit algorithm is its limitation to the problems where the sparse matrix does not dominate the low-rank matrix in magnitude; for this reason, we compare the proposed method with the LMaFit algorithm in different setup. Tables III, IV, and V show the results of comparison with the IALM, LSSD, SpaRCS, and LRSD-TNNSR methods for $r = 0.1 \times n^2$, $k = 0.2 \times n^2$, $0.3 \times n^2$, and $0.4 \times n^2$, respectively. It is clear that the proposed algorithm with both the homographic and Gaussian smoothed functions outperform the other methods. The IALM, LSSD, and LRSD-TNNSR methods have a high computation time in terms of run-time. According to the results of Tables IV and V, the IALM method fails to recover the low rank and sparse components when the sparsity number of the noise is high. Moreover, we can see that the LSSD fails to recover the low rank and sparse components in random problem setup which may be due to the fact that here, the noise component has no structure (random structure) while the LSSD method incorporates the structure sparsity of the noise component. Another important point is that the SpaRCS method needs an approximation of the rank and sparsity of the low rank and sparse components which seems to be unfair comparing to the other methods.

In the following, we compare the proposed algorithm with the LMaFit algorithm [30] which is an ADMM based algorithm for matrix separation based on low rank factorization. The MATLAB code of the LMaFit algorithm is downloaded from the LMaFit website [29] and the all parameters are set to their default values. As mentioned before, the LMaFit algorithm fails to recover the low rank and sparse components when the sparsity number of the noise is high. Moreover, we can see that the LSSD method fails to recover the low rank and sparse components in random problem setup which may be due to the fact that here, the noise component has no structure (random structure) while the LSSD method incorporates the structure sparsity of the noise component. Another important point is that the SpaRCS method needs an approximation of the rank and sparsity of the low rank and sparse components which seems to be unfair comparing to the other methods.

In the following, we compare the proposed algorithm with the LMaFit algorithm [30] which is an ADMM based algorithm for matrix separation based on low rank factorization. The MATLAB code of the LMaFit algorithm is downloaded from the LMaFit website [29] and the all parameters are set to their default values. As mentioned before, the LMaFit algorithm fails to recover the low rank and sparse components when the sparsity number of the noise is high. Moreover, we can see that the LSSD method fails to recover the low rank and sparse components in random problem setup which may be due to the fact that here, the noise component has no structure (random structure) while the LSSD method incorporates the structure sparsity of the noise component. Another important point is that the SpaRCS method needs an approximation of the rank and sparsity of the low rank and sparse components which seems to be unfair comparing to the other methods.

In the following, we compare the proposed algorithm with the LMaFit algorithm [30] which is an ADMM based algorithm for matrix separation based on low rank factorization. The MATLAB code of the LMaFit algorithm is downloaded from the LMaFit website [29] and the all parameters are set to their default values. As mentioned before, the LMaFit algorithm fails to recover the low rank and sparse components when the sparsity number of the noise is high. Moreover, we can see that the LSSD method fails to recover the low rank and sparse components in random problem setup which may be due to the fact that here, the noise component has no structure (random structure) while the LSSD method incorporates the structure sparsity of the noise component. Another important point is that the SpaRCS method needs an approximation of the rank and sparsity of the low rank and sparse components which seems to be unfair comparing to the other methods.

In the following, we compare the proposed algorithm with the LMaFit algorithm [30] which is an ADMM based algorithm for matrix separation based on low rank factorization. The MATLAB code of the LMaFit algorithm is downloaded from the LMaFit website [29] and the all parameters are set to their default values. As mentioned before, the LMaFit algorithm fails to recover the low rank and sparse components when the sparsity number of the noise is high. Moreover, we can see that the LSSD method fails to recover the low rank and sparse components in random problem setup which may be due to the fact that here, the noise component has no structure (random structure) while the LSSD method incorporates the structure sparsity of the noise component. Another important point is that the SpaRCS method needs an approximation of the rank and sparsity of the low rank and sparse components which seems to be unfair comparing to the other methods.

In the following, we compare the proposed algorithm with the LMaFit algorithm [30] which is an ADMM based algorithm for matrix separation based on low rank factorization. The MATLAB code of the LMaFit algorithm is downloaded from the LMaFit website [29] and the all parameters are set to their default values. As mentioned before, the LMaFit algorithm fails to recover the low rank and sparse components when the sparsity number of the noise is high. Moreover, we can see that the LSSD method fails to recover the low rank and sparse components in random problem setup which may be due to the fact that here, the noise component has no structure (random structure) while the LSSD method incorporates the structure sparsity of the noise component. Another important point is that the SpaRCS method needs an approximation of the rank and sparsity of the low rank and sparse components which seems to be unfair comparing to the other methods.

In the following, we compare the proposed algorithm with the LMaFit algorithm [30] which is an ADMM based algorithm for matrix separation based on low rank factorization. The MATLAB code of the LMaFit algorithm is downloaded from the LMaFit website [29] and the all parameters are set to their default values. As mentioned before, the LMaFit algorithm fails to recover the low rank and sparse components when the sparsity number of the noise is high. Moreover, we can see that the LSSD method fails to recover the low rank and sparse components in random problem setup which may be due to the fact that here, the noise component has no structure (random structure) while the LSSD method incorporates the structure sparsity of the noise component. Another important point is that the SpaRCS method needs an approximation of the rank and sparsity of the low rank and sparse components which seems to be unfair comparing to the other methods.

When the sparse component dominates the low rank component in magnitude. Therefore, we do the comparison in a distinct setup where the non-zero values of sparse component $E_o$ are chosen from the set $\{-\sigma, +\sigma\}$ with equal probabilities. We set $r = 0.05 \times n$, $k = 0.1 \times n^2$, and $\sigma = 0.1$ for this comparison. The results are shown in Table VI. It is obvious that the LMaFit algorithm fails to recover the low rank and sparse components even for $\sigma = 0.1$.

D. Phase Transition Between Rank and Sparsity

This subsection is devoted to empirically investigate the performance of the proposed algorithm with different values of rank and sparsity. The low rank matrices of size $m = n = 200$ are generated by the same approach explained in Subsection V-A. For obtaining the phase transition between rank and sparsity, each simulation generates 20 random problems and a solution declared successful if the recovered $\hat{L}_o$ satisfies $\text{SNR}(\hat{L}_o, \hat{L}_{o_0}) \geq 60 \, \text{dB}$. In the phase transition plot, the gray color indicates the recovery rate. Moreover, the white and black colors represent 100% success and failure areas, respectively, while the gray areas show a success probability between 0% to 100%. It should be noted that while, we plot the results in terms of rank or sparsity level, these values were not revealed to the algorithm (no prior information is used in the decomposition procedure). We consider two experiments for this purpose.

1) Random Noise: In this experiment, the noise matrix $E_o$ has a support that obeying a Bernoulli distribution with random signs. Each entry of noise matrix takes on values of $\{0, -1, +1\}$ with probabilities of $\{1 - p, p/2, p/2\}$. The phase transition of this experiment is depicted in Fig. 5. As expected in most phase-transition curves, the gray area shrinks at high dimensions (asymptotic sharp phase transition). One

Fig. 5. Phase transition of the proposed algorithm with random noise.

Fig. 6. Phase transition of the proposed algorithm with coherent noise.
can see that the phase transition of the proposed algorithm for both homographic and Gaussian smoothed functions are almost the same. Furthermore, for all pairs of \((p, r/n)\) less than \((0.26, 0.35)\), the low rank matrix could be recovered successfully.

2) Coherent Noise: Unlike the previous experiment, in this experiment, we assume that the noise components are coherent with the low rank matrix. First, a random binary mask \(M\) is generated that takes on values of 0 and 1 with probabilities \(1 - p\) and \(p\), respectively. Then, the noise matrix \(E_o\) is generated as \(E_o = M \odot \text{sgn}(L_o)\). Fig. 6 shows the phase transition of this experiment. Although this experiment seems to be hard, the successful area (white-colored area) of the phase transition plot is not so tight.

**E. Applications**

There are many applications in which one needs to recover a low rank matrix from the corrupted observations. Here, we consider two applications, removing shadows and specularities from face images and background modeling, and compare our method with ALM [13], LSSD [37], SpaRCS [38], LRSD-TNNSR [39], and RASL [40].

1) Removing Shadows and Specularities from Face Images: If there are enough images from a face with different luminances, shadows, and specularities, one can use the low rank and noise recovery method to get rid of such noises from the faces because this dataset has also low dimension [3]–[5]. We have selected three face images from the Extended Yale Face Database B (B+) dataset [41]. The results are shown in Fig. 7. The run-times of the LSD-HSN, LSD-GSN, ALM, LSSD, SpaRCS, LRSD-TNNSR, and RASL algorithms are 3.7, 3.6, 11.1, 65.9, 2.0, 5.1, and 12.3 seconds, respectively.

2) Background Modeling from Surveillance Video: In video surveillance applications, one needs to detect any activity or change in the successive frames [1], [3]. Each group of picture (GOP) of video has a main background. Therefore,
we get a low rank matrix if vectorize all frames of a GOP and put them in a matrix. In this way, any activity on the foreground can be modeled as a sparse noise that can be separated from the main background. To show this, we used the proposed method, ALM, LSSD, and SpaRCS, LRSD-TNNSR, and RASL methods to model the background of two selected video sequences, Hall of a business building [42] and a nominal sequence introduced in [43]. Fig. 8a shows three frames from the original Hall video. Figs. 8b, 8d, 8f, 8h, 8j, 8l, and 8n show the low rank component (background) and Figs. 8c, 8e, 8g, 8i, 8k, 8m, and 8o show the sparse component (moving object), respectively. The results for the second video sequence is shown in Fig. 9. It should be mentioned that in this application, the primary low rank and the noise components are not purely low rank and sparse, respectively. Although we can not see considerable differences between the results, the proposed algorithm has lower run-time. The run-times of the LSD-HSN, LSD-GSN, ALM, LSSD, SpaRCS, LRSD-TNNSR, and RASL methods are 135.3, 128.5, 252.4, 2786.2, 199.6, 155.9, and 1442.7 seconds, respectively.

VI. CONCLUSION

A new algorithm for recovery of low rank and noise components of a given data matrix was proposed. Unlike the state-of-the-art methods that have approximated the $\ell_0$-norm of the rank and sparsity with nuclear norm and $\ell_1$-norm, we approximated the $\ell_0$-norm by families of smoothed $\ell_0$-norm functions. The GP method is used for solving the minimization problem. Many considerations were made to avoid GP algorithm getting trapped in the local minima points. The convergence of the proposed algorithm was analytically provided. The proposed algorithm was compared with the Augmented Lagrange Multiplier (ALM) based method and LMaFit method. For the synthetic signals, the simulation results indicated that the proposed algorithm could exactly recover the low rank and noise matrices at much less complexity in terms of run-time; in some cases, the ALM-based method and LMaFit method failed to recover the low rank and sparse components. We have also applied the proposed algorithm for two real applications: background modeling for video surveillance and removing shadows and specularities from face images. We have also compared all results with those the ALM method. In general, the proposed algorithm had better performance at lower run-time.

ACKNOWLEDGMENT

The author would like to thank Dr. Pedram Pad for his help of reviewing the proof of convergence.

REFERENCES

Fig. 8. Background modeling results on a 176 × 144 video sequence. (a) Original frames. (b) Low rank component (ALM). (c) Sparse component (ALM). (d) Low rank component (LSSD). (e) Sparse component (LSSD). (f) Low rank component (SpaRCS). (g) Sparse component (SpaRCS). (h) Low rank component (LRSD-TNNSR). (i) Sparse component (LRSD-TNNSR). (j) Low rank component (RASL). (k) Sparse component (RASL). (l) Low rank component (LSD-HSN). (m) Sparse component (LSD-HSN). (n) Low rank component (LSD-GSN). (o) Sparse component (LSD-GSN).

Fig. 9. Background modeling results on a 360 × 240 video sequence. (a) Original frames. (b) Low rank component (ALM). (c) Sparse component (ALM). (d) Low rank component (LSSD). (e) Sparse component (LSSD). (f) Low rank component (SpaRCS). (g) Sparse component (SpaRCS). (h) Low rank component (LRSD-TNNSR). (i) Sparse component (LRSD-TNNSR). (j) Low rank component (RASL). (k) Sparse component (RASL). (l) Low rank component (LSD-HSN). (m) Sparse component (LSD-HSN). (n) Low rank component (LSD-GSN). (o) Sparse component (LSD-GSN).


