A FAST AND EFFICIENT ALGORITHM FOR LOW RANK MATRIX RECOVERY FROM INCOMPLETE OBSERVATIONS

Nam Nguyen[†], *Thong T. Do*[†], *Yi Chen*[†] *and Trac D. Tran*[†] *

[†] Department of Electrical and Computer Engineering
The Johns Hopkins University

ABSTRACT

Minimizing the rank of a matrix $\boldsymbol{X}$ over certain constraints arises in diverse areas such as machine learning, control system and is known to be computationally NP-hard. In this paper, a new simple and efficient algorithm for solving this rank minimization problem with linear constraints is proposed. By using gradient projection method to optimize $\boldsymbol{S}$ while consecutively updating matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ (where $\boldsymbol{X} = \boldsymbol{USV}^T$) in combination with the use of an approximation function for l^0-norm of singular values [1], our algorithm is shown to run significantly faster with much lower computational complexity than general-purpose interior-point solvers, for instance, the SeDuMi package [2]. In addition, the proposed algorithm can recover the matrix exactly with much fewer measurements and is also appropriate for large-scale applications.

Index Terms— Rank minimization, system identification, matrix norm, compressed sensing, convex optimization.

1. INTRODUCTION

The general rank minimization problem can be expressed as

$$\min_{\boldsymbol{X}} \quad rank(\boldsymbol{X}) \qquad \text{s.t.} \qquad \boldsymbol{X} \in C, \tag{1}$$

where $\boldsymbol{X} \in R^{n\times m}$ is the optimization variable and C is a convex set denoting the constraints.

In this paper, we concentrate on the scenario where the constraint is a linear map $\mathcal{A} : R^{n\times m} \rightarrow R^p$ [3]

$$\min_{\boldsymbol{X}} \quad rank(\boldsymbol{X}) \qquad \text{s.t.} \qquad \mathcal{A}(\boldsymbol{X}) = \boldsymbol{b}, \tag{2}$$

where the linear map $\mathcal{A}$ and vector $\boldsymbol{b} \in R^p$ are given. This problem has many practical applications such as: linear system realization, minimum-order system approximation, reduced-order controller design [4] and low rank matrix completion [5].

In general, the problem (2) is known to be NP-hard. A recent heuristic algorithm introduced by Fazel [4] replaces the rank function by the nuclear norm, or sum of singular values, over the constraint set. If we denote the singular values of the matrix $\boldsymbol{X}$ by s_i $(i = 1, 2, ..., n)$ with $s_1 \geq s_2 \geq ...$, then the nuclear norm is defined as: $\|\boldsymbol{X}\|_* = \sum_{i=1}^{n} s_i$.

The heuristic optimization is then given by

$$\min_{\boldsymbol{X}} \quad \|\boldsymbol{X}\|_* \qquad \text{s.t.} \qquad \mathcal{A}(\boldsymbol{X}) = \boldsymbol{b}. \tag{3}$$

In the inspiring paper of B. Recht et al. [3], the authors showed that if the linear map is *nearly isometrically distributed*, the solution of nuclear norm minimization (3) will coincide with (2), and solving (3) will give the exact solution.

By using the nuclear norm, the optimization in (3) is convex. Hence, it can be formulated as a primal-dual semidefinite program, and solved by the well-known interior-point methods [3], [5], [2]. However, the reformulation requires large auxiliary matrix variables, and thus might be impractical for large-scale problems.

In our algorithm, we directly search for the minimum number of non-zero singular values, or the l^0-norm of the singular vector $\boldsymbol{s}$. By doing this, we have significantly narrowed down the search space of variables. The space of decision variables are now the space of $\boldsymbol{s}$ which is only R^n (without loss of generality, we assume that $n < m$). In addition, while searching $\boldsymbol{s}$, we keep updating variables $\boldsymbol{U}$ and $\boldsymbol{V}$ in the sigular value decomposition $\boldsymbol{X} = \boldsymbol{USV}^T$. Fortunately, as $\boldsymbol{s}$ converges to the exact solution, $\boldsymbol{U}$ and $\boldsymbol{V}$ converge as well, leading $\boldsymbol{X}$ to the desirable value. This feature turns out to tremendously improve the speed and the precision of the algorithm.

The paper is organized as follows. The next section presents the main ideas and concepts behind our proposed method. In Section III, we introduce the algorithm with detailed explanations. Finally, the last two sections will present our experimental results and contain a few concluding remarks, respectively.

2. MAIN IDEAS

Generally, the cost function of (2) and (3) can be regarded as two instances of a function $F(\boldsymbol{s})$, where $F(\boldsymbol{s})$ can be l^p-norm with $0 \leq p \leq 1$. If $F(\boldsymbol{s})$ is l^0-norm of $\boldsymbol{s}$, it is equivalent to minimize rank($\mathbf{X}$). On the other hand, if $p = 1$, it is minimizing nuclear norm $\|\boldsymbol{X}\|_*$.

$$\min_{\boldsymbol{X}} \quad F(\boldsymbol{s}) \qquad \text{s.t.} \qquad \mathcal{A}(\boldsymbol{USV}^T) = \boldsymbol{b}. \tag{4}$$

One can see at first glance that (4) is irrelevant since unknown variables $\boldsymbol{U}$ and $\boldsymbol{V}$ disappear into the cost function. However, if somehow we can fix them at a time and try to find $\boldsymbol{s}$ closer to the minimizer, and then simultaneously update $\boldsymbol{U}$ and $\boldsymbol{V}$. Again, at the next iteration, $\boldsymbol{U}$ and $\boldsymbol{V}$ are kept fixed, then find a new $\boldsymbol{s}$. After a number of iterations, the algorithm will converge to a satisfactory solution. The proposed approach has a similar flavor to that of the celebrated EM algorithm.

If $\boldsymbol{U}$ and $\boldsymbol{V}$ are unchanged over many iterations, one can consider them as fixed sparsifying transforms that try to zero out as many singular values as possible. Consequently, (4) can be seen as the well-known compressed sensing problem and hence can be solved by many efficient methods [1]. On the other hand, if $\boldsymbol{U}$ and $\boldsymbol{V}$ keep changing in every iteration, then these matrices can be considered as adaptive sparsifying transforms that can adapt well to the varying behaviors of the signal.

*This work has been supported in part by the National Science Foundation under Grant CCF-0728893.

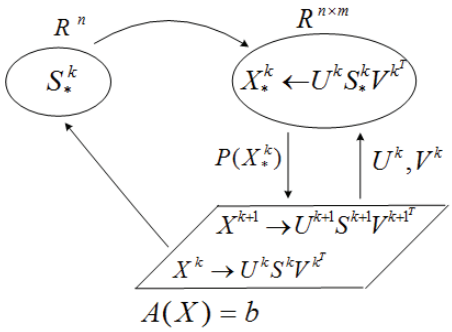


Fig. 1. Adaptive model

A gradient projection method can be employed to optimize (4). The diagram in Figure 1 clarifies the process, where the parallelogram area represents the linear constraint region. Suppose at iteration k, we obtain a feasible approximation $\boldsymbol{X}^k$. Then, $\boldsymbol{X}^k$ is decomposed using the SVD to obtain $\boldsymbol{U}^k$, $\boldsymbol{V}^k$ and $\boldsymbol{S}^k$. Subsequently, the vector $\boldsymbol{s}^k$ is updated to move closer to the minimizer of $F(\boldsymbol{s})$, where $\boldsymbol{s}^k$ is the singular vector of matrix $\boldsymbol{S}^k$. The new updated $\boldsymbol{s}_*^k$ will be combined with previous matrices $\boldsymbol{U}^k$ and $\boldsymbol{V}^k$ to generate a new $\boldsymbol{X}_*^k$. Here, $\boldsymbol{X}_*^k$ is not guaranteed to be a feasible point, so it needs to be projected back onto the constraint region to obtain a new feasible approximator $\boldsymbol{X}^{k+1}$. The cycle continues until we arrive at a certain chosen termination condition.

One more contribution of our paper is that, by the proposed method, we do not have to restrict ourself to optimizing l^0- or l^1-norm of the singular vector $\boldsymbol{s}$ of $\boldsymbol{X}$ (which is to minimize the rank or the nuclear norm of $\boldsymbol{X}$, respectively). Instead, we can minimize the more general l^p-norm of $\boldsymbol{s}$ as well. This non-convex optimization has been confirmed to work better (although slower) than the l^1-norm solution from the compressed sensing community [6].

In this paper, the l^0-norm is utilized as the cost function of (4). Unfortunately, l^0-norm is a discontinuous function that is not differentiable. We take advantage of a recent striking discovery: approximating l^0-norm by a smooth function [1] has been shown to be very fast and effective for sparse signal recovery.

3. ALGORITHM DESCRIPTION

Firstly, let us re-introduce the smooth approximation function of the l^0-norm of $\boldsymbol{s}$ [1]. Define the following smooth function

$$f_\sigma(s_i) = \exp(-s_i^2/\sigma^2).$$

The value of σ is used to control the quality of the estimation

$$f(s_i) \approx \begin{cases} 1 & \text{if } s_i \ll \sigma \\ 0 & \text{if } s_i \gg \sigma \end{cases}$$

In other words, $f(s_i) = 1$ as $s_i = 0$ and $f(s_i) = 0$ as $\sigma \rightarrow 0$. Then by defining

$$F_\sigma(\boldsymbol{s}) = n - \sum_{i=1}^{n} f(s_i), \tag{5}$$

it is not difficult to verify that $F_\sigma(\boldsymbol{s}) \approx \|\boldsymbol{s}\|_0 - n$ as $\sigma \rightarrow 0$. Therefore, the rank minimization problem is now reduced to finding the minimum of a differential function $F_\sigma(\boldsymbol{s}) = n - \sum_{i=1}^{n} \exp(-s_i^2/\sigma^2)$ subject to constraints

$$\min_{\boldsymbol{X}} \quad n - \sum_{i=1}^{n} \exp(-s_i^2/\sigma^2) \qquad \text{s.t.} \quad \mathcal{A}(\boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T) = \boldsymbol{b}, \tag{6}$$

when σ is very small. Note that for small values of σ, $F_\sigma(\boldsymbol{x})$ is highly oscillatory and contains a lot of local minima, causing difficulty in finding the global solution. On the other hand, as σ increases, $F_\sigma(\boldsymbol{x})$ has much fewer local minima and is easier to solve. In [1], the authors recommend to start from a large σ, then at each fixed σ, (6) is optimized and that minimizer is used as the initial approximator for the next smaller σ. Therefore, at each inner loop, initiated from a good point will help the algorithm avoid most local minima, and as $\sigma \rightarrow 0$, the algorithm will converge to the global solution.

One can observe that even with a fixed σ, (6) is difficult to solve since the cost function is not convex. Moreover, the decision variable vector $\boldsymbol{s}$ is hidden in the constraint set and there is no easy access to matrices $\boldsymbol{U}$ and $\boldsymbol{V}$. A simple solution to this problem is to apply the aforementioned idea of alternatively updating $\boldsymbol{U}$, $\boldsymbol{V}$ and $\boldsymbol{s}$ at each fixed σ (see Figure 1 again). The algorithm should return the exact solution after a small number of iterations.

The pseudocode for the proposed algorithm is shown in Algorithm 1. For elegance of presentation, we consider the algorithm as two separated processes. The first process called the outer loop operates with decreasing values of σ. The second process called the inner loop optimizes function $F_\sigma(\boldsymbol{s})$ at each fixed value of σ. The minimizer of the current σ will be employed as the initial point of the next σ. So, let us first present how to select the initial parameters.

Initialization:
$\boldsymbol{X}^0$, σ_{max}, σ_{min}, tolerance ϵ, $d > 1$ and $\sigma = \sigma_{max}$
while $\sigma > \sigma_{min}$ **do**
  $\boldsymbol{X}^0 \rightarrow \boldsymbol{U}^0 \boldsymbol{S}^0 \boldsymbol{V}^{0^T}$ and $\boldsymbol{s}^0 \leftarrow$ diagonal$(\boldsymbol{S}^0)$
  $k \leftarrow 1$
  repeat
    $\nabla F_\sigma(\boldsymbol{s}^k) \leftarrow \frac{2}{\sigma^2}[s_1^k e^{-(s_1^k)^2/\sigma^2}, ..., s_n e^{-(s_n^k)^2/\sigma^2}]$
    $\boldsymbol{s}_*^k \leftarrow \boldsymbol{s}^k - \beta \cdot \nabla F_\sigma(\boldsymbol{s}^k)$ {Gradient descent}
    $\boldsymbol{X}_*^k \leftarrow \boldsymbol{U}^k \boldsymbol{S}_*^k \boldsymbol{V}^{k^T}$ {New approximation}
    $\boldsymbol{X}^{k+1} \leftarrow \mathcal{P}_{(\mathcal{A}(\boldsymbol{X})=\boldsymbol{b})}(\boldsymbol{X}_*^k)$ {Projection}
    $\boldsymbol{X}^{k+1} \rightarrow \boldsymbol{U}^{k+1} \boldsymbol{S}^{k+1} \boldsymbol{V}^{k+1^T}$ {SVD decomposition}
    $k \leftarrow k + 1$
  until halting condition true
  $\sigma \leftarrow \sigma / d$ {Decreasing σ}
  $\boldsymbol{X}^0 \leftarrow \boldsymbol{X}^k$
end while
Output: $\boldsymbol{X}^* \leftarrow \boldsymbol{X}^0$

Algorithm 1: Rank minimization algorithm

3.1. Initialization

In general, any feasible initial point vec$(\boldsymbol{X}^0)$ can be chosen as $\boldsymbol{A}^+\boldsymbol{b}$, where vec$(\boldsymbol{X})$ denotes the vectorized version of matrix $\boldsymbol{X}$, $\boldsymbol{A}$ denotes the matrix representation of the linear map $\mathcal{A}$ and $\boldsymbol{A}^+$ is the Moore-Penrose pseudoinverse of $\boldsymbol{A}$. However, for large-scale problems or when we need to execute the linear map implicitly, it is costly to compute the pseudoinverse $\boldsymbol{A}^+$. We can instead choose a random point, then project it onto the feasible space. In many cases, we can select vec$(\boldsymbol{X}^0) = \boldsymbol{A}^T\boldsymbol{b}$.

The orthogonal projection of $\boldsymbol{X}^0$ onto the affine space $\mathcal{A}(\boldsymbol{X}) = \boldsymbol{b}$ can be performed as follows

$$\mathcal{P}_{(\mathcal{A}(\boldsymbol{X})=\boldsymbol{b})}(\boldsymbol{X}^0) = \text{vec}(\boldsymbol{X}^0) - \boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{A}^T)^{-1}(\boldsymbol{A}\text{vec}(\boldsymbol{X}^0) - \boldsymbol{b}))$$

The equation is greatly simplified and easy to execute if the linear map is orthogonal

$$\boldsymbol{X}^0 \leftarrow \mathcal{P}_{(\mathcal{A}(\boldsymbol{X})=\boldsymbol{b})}(\boldsymbol{X}^0) = \boldsymbol{X}^0 - \mathcal{A}^T(\mathcal{A}(\boldsymbol{X}^0) - \boldsymbol{b}))$$

As shown above, a good starter for σ_{max} is $\sigma \gg s_i^0$, hence it is sufficient to choose $\sigma_{max} = 2\max(s^0)$ because at large σ, the exponential function only changes slightly with a substantial change of σ.

3.2. Outer loop

At the outer loop, the changes of σ significantly affect quality and speed of the algorithm. When σ decreases dramatically, we need much lower number of iterations at the outer loop, hence speeding up the algorithm. The tradeoff here is the higher possibility to get trapped into a local minimum, since the convergence rate of $\boldsymbol{s}$ might not follow the decreasing speed of σ. Therefore, it causes $s_i \gg \sigma$ that lead to $\exp(-s_i^2/\sigma^2) \approx 0$. Consequently, $\beta \cdot \nabla F_\sigma(\boldsymbol{s}^k) \approx 0$ and $\boldsymbol{s}^k$'s are kept nearly fixed in every inner loop (see the equation of step 8 in the pseudo code of Algorithm 1). Finally, the algorithm is not able to converge.

When σ is reduced gradually, the algorithm needs more iterations for the outer loop, and s_i's move toward to the convergent point in parallel with the movement of σ. This characteristic guarantees the convergence of matrices $\boldsymbol{U}^k$ and $\boldsymbol{V}^k$, and so does the overall algorithm. Choosing the best decreasing factor d is dependent on specific applications. Experimentally, a suitable range for d's to ensure convergence is $d = 2 \rightarrow 4$. One can also choose a non-continuous sequence σ for his/her own purpose.

It is noteworthy that when σ is very small, only s_i's smaller than σ can affect the fluctuation of $F_\sigma(\boldsymbol{s})$, hence σ can be seen as a soft threshold of the algorithm. Consequently, the choice of σ_{min} will control the precision of the final solution. The smaller σ_{min}, the slower the algorithm runs, but it will return a more precise solution. In the proposed algorithm, we find that $\sigma_{min} = 10^{-4}$ is a safe choice.

3.3. Inner loop

In each inner-loop process, σ is kept fixed. We use a gradient projection method to optimize (6).

At the gradient descent step (step 8 in the pseudocode), the step size β needs to be considered. Since at smaller σ, $F_\sigma(\boldsymbol{s})$ is fluctuating more, hence β should be small to escape from bypassing over the global minimum. In our proposed algorithm, β is chosen to be proportional to σ. Particularly, $\beta = \sigma^2$.

At each fixed σ, the decision to terminate the inner loop is rather difficult. Obviously, we wish to stop at the optimal solution of $F_\sigma(\boldsymbol{s})$. Since the objective function value is simple to compute, we decide to use it as a criteria for termination. The inner loop is executed as long as the absolute value of the difference between $F_\sigma(\boldsymbol{s}^k)$ and $F_\sigma(\boldsymbol{s}^{k+1})$ keeps above a specified threshold ϵ. In other words, when

$$|F_\sigma(\boldsymbol{s}^{k+1}) - F_\sigma(\boldsymbol{s}^k)| < \epsilon, \tag{7}$$

the inner loop is terminated.

The choice of ϵ is also dependent on specific applications. Since $F_\sigma(\boldsymbol{s})$ is a decreasing exponential function that is not sensitive as $\boldsymbol{s}$ does not change very much at each iteration. This is especially true as σ is very small. When ϵ is small, it results in more iterations at each inner loop, but $\boldsymbol{s}^k$ will come closer to the optimal solution of $F_\sigma(\boldsymbol{s})$. Therefore, the choice of ϵ is a tradeoff between computational cost and the precision of the algorithm. In our experiments, ϵ is chosen to be 0.005.

Remark. With the large scale problems, the full SVD decompositions at every outer and inner loops are computationally expensive. Instead, a slight modification of the algorithm is at each step, only compute first largest r singular vectors $\boldsymbol{u}_i^k, \boldsymbol{v}_i^k$ and singular values s_i^k of $\boldsymbol{X}^k$, where r is rank of the matrix we need to find. This approach can be seen as a singular value hard thresholding where all singular values of $\boldsymbol{X}^k$, except r largest ones, are zeroed out. This modification significantly improve the speed of the algorithm and make the algorithm tractable in solving large matrices.

4. SIMULATION RESULTS

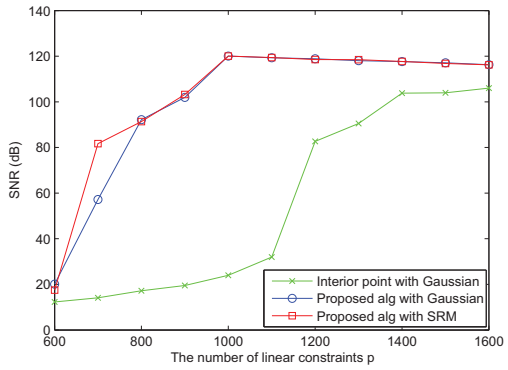
In this section, the performance and speed of the proposed algorithm are experimentally justified and compared with the interior-point semidefinite programming (SDP) method which has been implemented in a freely available software SeDuMi [2]. Note that in all experiments, parameters of the algorithm are set as follows:

$$\sigma_{max} = \max(s^0), \quad \sigma_{min} = 10^{-3}, \quad d = 2, \quad \epsilon = 0.005, \quad \beta = \sigma^2$$

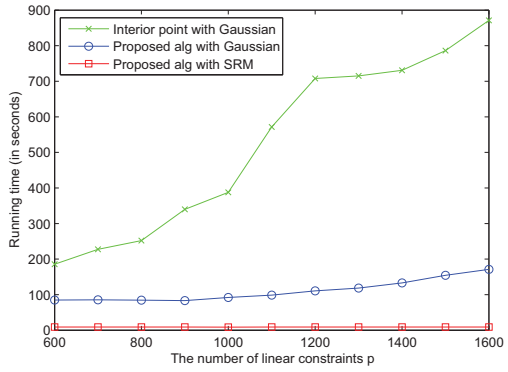
Experiment 1: This experiment is devoted to compare the recovery performance and speed of our algorithm with SDP using SeDuMi [2]. We adopt the MIT logo matrix $\boldsymbol{X}$ [3], which has size 46×81 and rank $r = 5$ as the test input. The matrix is sampled using an orthogonal Gaussian i.i.d measurement matrix with the number of measurements ranging from 600 to 1600 ($p = 600 \rightarrow 1600$ with step size 100). Figure 2a and 2b depict the performance curves and computation time respectively. In both figures, the numerical values on the x-axis represent the number of linear constraints (or measurements) p while the values on y-axis of Figure 2a represent the Signal to Noise ratio (SNR) between the original matrix ($\boldsymbol{X}$) and the recovered one ($\boldsymbol{X}_r$). Those on y-axis of Figure 2b represent the running time of the algorithms. As one can observe, our algorithm offers significant improvements in both recovery performance as well as computational complexity.

Experiment 2: In this experiment, we sample the image using Structurally Random Matrix (SRM) [7] to take advantage of its implicit construction, a property that fully random matrices do not have, and show that our algorithm can handle the implicit form of the linear map. This form is particularly important, especially when working with large-scale applications, since it is impossible to create a very large random Gaussian matrix. The sampling process of SRM is as follows: at first, the test image $\boldsymbol{X}$ is vectorized, then the signs of its entries are changed in a uniformly random manner, the output is then passed into a fast transform such as DWT, FFT or DCT, etc. (in this experiment, the DCT transform is chosen). Afterward, the measurements are uniformly and randomly selected. We take the same number of measurements as in Experiment 1. The performance and time computation curves are depicted in Figures 2a and 2b. While the performance of SRM is nearly the same as that of the Gaussian matrix (even slightly better with small p), the computational cost is substantially lowered since the algorithm is able to exploit the implicitly fast and efficient structure of SRM transforms.

Experiment 3: This experiment compares the performance between the proposed algorithm and SeDuMi with randomly-generated inputs. Matrices of size $n \times n$ and rank r are produced by generating pairs of two Gaussian matrices $\boldsymbol{U} \in R^{n\times r}$ and $\boldsymbol{V} \in R^{r\times n}$



(a)



(b)

Fig. 2. Performance and time computation curves of i.i.d Gaussian and SRM measurement matrices: SNR and running time vs. the number of measurements using of interior-point SeDuMi solver and proposed algorithm.

and setting $\boldsymbol{X} = \boldsymbol{UV}$, where $n = 50$ and rank $r = \{5, 6, 7\}$. At each fixed r, i.i.d Gaussian matrices are used to sample various number of measurements $p = \{400, 600, ..., 1500\}$. Figure 3 represents the performance curve of two reconstruction algorithms with different ranks. In the figure, the x- and y-axis stand for the number of measurements and SNR, respectively. Once again, our reconstruction algorithm outperforms SeDuMi. Note that SNR $= 50dB$ in floating-point precision is often regarded as perfect recovery.

Matrix size	Rank	p/n^2	Relative error	Time (s)
1000×1000	10	0.15	3.29×10^{-4}	25
	50	0.27	4.94×10^{-4}	95
	100	0.38	6.35×10^{-4}	256
3000×3000	10	0.1	5.75×10^{-4}	215
	50	0.2	1.97×10^{-5}	768

Table 1: Experimental results of matrix completion with various matrix sizes and ranks.

Experiment 4: The last experiment is devoted to the large scale matrix completion problem [5]. Matrices $\boldsymbol{X}$ of size $n \times n$ and rank r are produced as the above experiment, where $n = \{1000, 4000\}$ and $r = \{10, 50, 100\}$. We pick a subset of p entries of $\boldsymbol{X}$ uniformly at random. In this experiment, we use the singular value hard thresholding method as presented in the remark of the Subsection 3.3. The

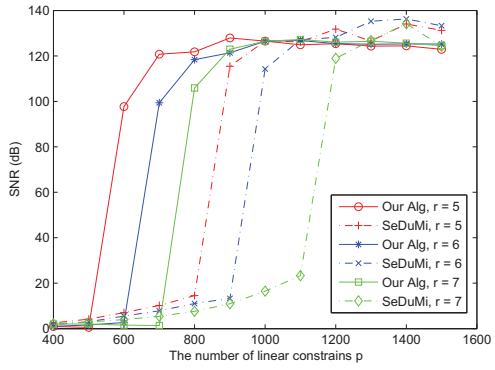


Fig. 3. Performance curves: SNR vs. the number of measurements using interior-point SeDuMi solver and proposed algorithm with various ranks.

algorithm is run on a laptop computer with 2.0GHz CPU and 3GB RAM. All the results are described in the Table 1, where the relative Frobenius norm error is defined as $\|X - X^*\|_F / \|X\|_F$. As shown in the table, one can see that a matrix 1000×1000 of rank 10 can be found exactly from 85% corruption only in less than half of a minute. We note that it is impossible to run such large matrices on the SeDuMi software.

5. CONCLUSION

In this paper, a new fast and efficient rank minimization algorithm is proposed. Extensive experimental results show that the proposed algorithm performs extremely well in reconstructing low rank matrices under linear constraints. Moreover, by exploiting the implicit structure of the linear map such as orthogonality and fast transforms, our algorithm is also proven to be appropriate for very large scale applications.

6. REFERENCES

[1] H. Mohimani, M. B. Zadeh, and C. Jutten, "Complex-valued sparse representation based on smoothed l^0-norm," *Proc. IEEE Int. Conf. of Acous., Speech and Signal Proc.*, pp. 3881–3884, Apr 2008.

[2] J. F. Sturm, "Using SeDuMi 1.02, a Matlab toolbox for optimization over symmetric cones," *Optimization Methods and Software*, pp. 625–653, 1999.

[3] B. Recht, M. Fazel, and P. A. Parrilo, "Guaranteed minimum rank solutions to linear matrix equations via nuclear norm minimization," Submitted to SIAM Review, 2007.

[4] M. Fazel, "Matrix rank minimization with applications," *Phd thesis, Stanford University*, 2002.

[5] E. J. Candès and B. Recht, "Exact matrix completion via convex optimization," Submitted for publication.

[6] R. Chartrand, "Exact reconstructions of sparse signals via nonconvex minimization," *IEEE Signal Proc. Lett.*, vol. 14, pp. 707–710, 2007.

[7] T. T. Do, T. D. Tran, and L. Gan, "Fast compressive sampling with structurally random matrices," *Proc. IEEE Int. Conf. of Acous., Speech and Signal Proc.*, pp. 3369–3372, May 2008.