Minimum Rank Matrix Completion with CVX in MATLAB

Jeremy Lerner
Department of Applied Mathematics and Statistics
Stony Brook University

May 28, 2015

Contents

Part I
Background

The minimum rank matrix completion problem stems from a desire to describe an incomplete set of observations with the simplest model possible.

Given an incomplete set of entries of a matrix, $M_{i,j}\ \forall\,(i,j)\ \in \Omega$, where Ω is the set of indices for the observations, the problem can be formulated as (1).

$$\begin{aligned} &\min_{X}\ \operatorname{rank}(X) \\ &\text{subject to } X_{i,j} = M_{i,j}\ \forall\,(i,j) \in \Omega \end{aligned} \tag{1}$$

However, this problem is NP-hard, see [14, 4], and there are no efficient methods to calculate the optimal X for this problem as stated.

1 The Netflix Problem

As an example of a minimum rank matrix completion problem, Netflix famously offered a million dollar prize for an algorithm that could, with a high degree of accuracy, predict user i's rating of movie j, given an incomplete matrix of ratings for many users and movies, see [1, 2]. The users, represented by rows of the matrix, rate movies, columns of the matrix, but not all users have rated all movies. Simply stated, the goal of the Netlix problem is, given an incomplete set of ratings, to find the least complicated model to predict ratings for unrated movies. Minimizing the rank of the completed matrix corresponds to the least complicated model, as long as we assume that only a relatively small number of factors contribute to an individual's tastes, see [4].

Part II
Convex Heuristics

1 Weighted Trace Heuristic

Since (1) is NP-hard, we seek viable heuristics to estimate it. One such heuristic for $X\ \in\ \mathbb{S}^n_+$ (positive semidefinite matrices) is the weighted trace, (2), where C is a convex constraint set.

$$\begin{aligned} &\text{Given diagonal weight } W > 0, \text{ find} \\ &\min_{X} \{\operatorname{trace}(WX) |\, X \in C,\ X \geq 0\} \end{aligned} \tag{2}$$

With a proper weight, W, (2) provides a lower bound for (1) because $\mathrm{tr}\,(X) = \sum_{i=1}^{n} \lambda_i\,(X) = \|\lambda\,(X)\|_1$ for $X \in S_+^n$. This is because, for symmetric matrices, $\mathrm{rank}\,(X) = \sum_{\lambda_i \neq 0} 1$, that is $\mathrm{rank}\,(X)$ is the number of nonzero eigenvalues, see [8, 10, 18]. However, the matrix we are attempting to recover may not be positive definite, so the weighted trace heuristic is not universally applicable to the minimum rank matrix completion problem.

2 Nuclear Norm Heuristic

2.1 Background and Equivalent Form

A more general heuristic, where $X \in \mathbb{R}^{m \times n}$, is the nuclear norm, $\|X\|_*$, this is also known as the trace norm and the Ky-Fan n-norm. First, we define the singular value decomposition of X to be $U\Sigma V^T$, where Σ is diagonal, $\Sigma_{ii} = \sigma_i$, and σ_i is the i^{th} largest singular value of X, and finally we define $\|X\|_* = \sum_{i=1}^{\min(m,n)} \sigma_i\,(X) = \mathrm{tr}\,(\Sigma)$ see [6, 4, 8, 12, 16]. This is because $\mathrm{rank}\,(X) = \sum_{\sigma_i \neq 0} 1$, that is, the rank of a matrix is the number of nonzero singular values, see [17]. In this way, we formulate a convex relaxation of (1) as

$$\begin{aligned} &\min_{X} \ \|X\|_* \\ &\text{subject to } X_{i,j} = M_{i,j}\ \forall\,(i,j) \in \Omega \end{aligned} \tag{3}$$

To show that (3) is actually a convex optimization problem, we can put (3) in epigraph form

$$\begin{aligned} &\min_{X,t} \ t \\ &\text{subject to } \|X\|_* \leq t \\ &\qquad\qquad X_{i,j} = M_{i,j}\ \forall\,(i,j) \in \Omega \end{aligned} \tag{4}$$

Next we need to use the following lemma, which is proven in section 5 of [8]: for $X \in \mathbb{R}^{m \times n}$ and $t \in \mathbb{R}$, we have $\|X\|_* \leq t$ if and only if there exist $Y \in \mathbb{R}^{m \times m}$ and $Z \in \mathbb{R}^{n \times n}$ such that

$$\begin{bmatrix} Y & X \\ X^T & Z \end{bmatrix} \geq 0 \text{ and } \mathrm{tr}\,(Y) + \mathrm{tr}\,(Z) \leq 2t \tag{5}$$

Therefore, we can equivalently state (3) as the semidefinite program (see [16, 3, 8, 13, 4])

$$\min_{X,Y,Z} \frac{\operatorname{tr}(Y) + \operatorname{tr}(Z)}{2}$$
$$\text{subject to} \begin{bmatrix} Y & X \\ X^T & Z \end{bmatrix} \geq 0$$
$$X_{i,j} = M_{i,j} \; \forall (i,j) \in \Omega \tag{6}$$

Where the optimal objective value $(\operatorname{tr}(Y^*) + \operatorname{tr}(Z^*))/2 = \|X^*\|_*$, where X^*, Y^* and Z^* are the minimizers of (6) and the same X^* is the minimizer of (3). Note that using the objective function $\operatorname{tr}(Y) + \operatorname{tr}(Z)$ is an equivalent problem, however, $(\operatorname{tr}(Y) + \operatorname{tr}(Z))/2$ is more closely tied to the original problem, (3), as seen in the above lemma.

2.2 Discussion

When we wish to recover a square matrix, $M \in \mathbb{R}^{n \times n}$, of rank r, there are $(2n - r)\, r$ degrees of freedom, where we denote the i^{th} row of M is m_i. This can be seen simply in the case $r = 1$, as $m_i = k_i \cdot m_1 \; \forall k = 2...n$, so there are $n - 1$ degrees of freedom for k_i and n degrees of freedom for the entries in m_1, giving $(2n - 1)(1) = 2n - 1$ degrees of freedom in total. Similarly, in the case $r = n - 1$, there are $n - 1$ rows with n entries each, giving $n^2 - n$ degrees of freedom and one row is a linear combination of the other rows, $m_1 = k_2 m_2 + k_3 m_3 + ... + k_n m_n$, so there are another $n - 1$ degrees of freedom, giving $(2n - (n-1))(n-1) = (n+1)(n-1) = n^2 - 1$ degrees of freedom.

Given so many degrees of freedom, we must sample data effectively to recover the minimum rank, and even more selectively if we wish to recover a *unique* minimizer, X^*. Since we must have least some information about each variable, to exactly recover a unique minimizer, a necessary condition is observing at least one value per row and one value per column, see [4]. Additionally, we would expect to make more observations than there are variables, and we can actually prove that if M is an $n_1 \times n_2$ matrix of rank r, sampled from a random orthogonal model (see section IIA of [4]), let $n = \max(n_1, n_2)$ and we observe m entries of M (sampled uniformly at random). Then $\exists C, c \in \mathbb{R}$ such that if

$$m \geq C \cdot n^{5/4} \cdot r \cdot \log(n) \tag{7}$$

then $X^* = M$ with probability at least $1 - cn^{-3}$, see Theorem 2.1 of [4]. Here we will simply use (7) on matrices formed by creating a random $u \in \mathbb{R}^{n_1 \times r}$ and $v \in \mathbb{R}^{r \times n_2}$, where $M = u \cdot v \in \mathbb{R}^{n_1 \times n_2}$ and M is rank r.

Also, it is important to note that $\|X\|_*$ varies under scaling; that is $\|2X\|_* = 2\|X\|_*$, while rank (the number of nonzero singular values) does not vary under scaling (that is, $\operatorname{rank}(X) = \operatorname{rank}(2X)$). So, while (3) is an excellent way to recover a low rank matrix, the optimal objective value of (3) does not give the optimal rank, instead it is the sum of the singular values. We simply use (3) to attempt to recover a matrix, not estimate its rank.

Part III
Numerical Algorithm

1 Setup and Code

To create the numerical results below, we use (6) to find X^*, the matrix that minimizes the sum of the singular values. To estimate the rank of X, we find the singular value decomposition of $X^* = USV^T$, where $S_{ii} = \sigma_i$ and S is diagonal, and the rank should be equal to the number of singular values greater than zero, though due to roundoff error, we find the number of singular values greater than a tolerance, $\epsilon = 10^{-6}$. We solve (6) in MATLAB using CVX with the following code (where the first five lines creates a rank r matrix $Q \in \mathbb{R}^{n_1 \times n_2}$, the matrix we are attempting to recover, and fills in exactly m entries of M from Q, leaving the rest of the entries as NaN-not a number).

```
Q = randn(n1,r)*randn(r,n2);
list1=randperm(numel(Q));
list1=list1(1:m);
M = nan(n1,n2);
M(list1) = Q(list1);
[m,n] = size(M);
nan_index = find(~isnan(M));
cvx_begin
    variables X(m,n) Y(m,m) Z(n,n)
    minimize trace(Y)/2 + trace(Z)/2
    subject to
        [Y X; X' Z] == semidefinite(n+m);
        X(nan_index) == M(nan_index);
cvx_end
epsilon = 1e-6;
[U, S, V] = svd(full(X));
numberOfSingularValues = numel(find(diag(S)>epsilon));
```

2 Results

2.1 A Simple Example

As an example, we attempt to recover the rank one matrix

$$Q = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix}$$

And we sample from Q to create M, where entries with NaN are not sampled (that is, $m = 6$).

$$M = \begin{bmatrix} \text{NaN} & \text{NaN} & 1 \\ 2 & 2 & \text{NaN} \\ 3 & 3 & 3 \end{bmatrix}$$

Running the above code, with high probability, we arrive at the following recovered matrix,

$$X = \begin{bmatrix} 0.999999984920282 & 0.999999984920398 & 0.999999992066022 \\ 1.999999996380770 & 1.999999996380779 & 1.999999985227588 \\ 3.000000003378126 & 3.000000003378888 & 3.000000000708987 \end{bmatrix}$$

which is approximately rank one, that is the singular values are approximately 6.48, $9.81 * 10^{-9}$ and $2.50 * 10^{-13}$. This implies that X is actually rank one but is somewhat corrupted by roundoff error. When M has even fewer entries specified, the algorithm is almost never able to recover Q exactly.

2.2 Tables

Similarly to the above example, we create the below tables where $Q \in \mathbb{R}^{n \times n}$ (Q is a real valued $n \times n$ matrix), $\text{rank}\,(Q) = r$, and we observe m values of Q to create M. The value recorded in the table is the number of runs of the program where the number of singular values greater than $\epsilon = 10^{-6}$ is exactly equal to $\text{rank}\,(Q)$, out of 100 runs. Specifically, we create table 1 with $m = n^2/4$, table 2 with $m = n^2/2$, table 3 with $m = 3m^2/4$ and table 4 with $m = 9n^2/10$.

Table 1: Correct Estimates, out of 100, $m = \frac{n^2}{4}$

$n \setminus r$	1	$\frac{n}{10}$	$\frac{2n}{10}$	$\frac{3n}{10}$	$\frac{4n}{10}$	$\frac{5n}{10}$	$\frac{6n}{10}$	$\frac{7n}{10}$	$\frac{8n}{10}$	$\frac{9n}{10}$	n
10	0	0	0	2	13	34	34	16	12	1	0
20	2	0	0	0	36	15	1	0	0	0	0
30	8	0	0	0	29	4	0	0	0	0	0
40	24	0	0	0	29	2	0	0	0	0	0
50	61	0	0	0	26	0	0	0	0	0	0
100	100	0	0	0	18	2	0	0	0	0	0

Table 2: Correct Estimates, out of 100, $m = \frac{n^2}{2}$

$n \setminus r$	1	$\frac{n}{10}$	$\frac{2n}{10}$	$\frac{3n}{10}$	$\frac{4n}{10}$	$\frac{5n}{10}$	$\frac{6n}{10}$	$\frac{7n}{10}$	$\frac{8n}{10}$	$\frac{9n}{10}$	n
10	28	28	0	0	0	25	58	23	1	0	0
20	71	24	0	0	0	0	53	2	0	0	0
30	89	35	0	0	0	0	56	1	0	0	0
40	99	38	0	0	0	0	46	0	0	0	0
50	99	44	0	0	0	0	32	4	0	0	0
100	100	75	0	0	0	0	10	1	0	0	0

Table 3: Correct Estimates, out of 100, $m = \frac{3n^2}{4}$

$n \setminus r$	1	$\frac{n}{10}$	$\frac{2n}{10}$	$\frac{3n}{10}$	$\frac{4n}{10}$	$\frac{5n}{10}$	$\frac{6n}{10}$	$\frac{7n}{10}$	$\frac{8n}{10}$	$\frac{9n}{10}$	n
10	80	80	49	6	0	0	0	30	65	3	0
20	96	95	43	1	0	0	0	4	66	0	0
30	100	97	62	2	0	0	0	0	50	0	0
40	100	100	77	0	0	0	0	0	38	0	0
50	100	100	89	1	0	0	0	0	37	0	0
100	100										

Table 4: Correct Estimates, out of 100, $m = \frac{9n^2}{10}$

$n \setminus r$	1	$\frac{n}{10}$	$\frac{2n}{10}$	$\frac{3n}{10}$	$\frac{4n}{10}$	$\frac{5n}{10}$	$\frac{6n}{10}$	$\frac{7n}{10}$	$\frac{8n}{10}$	$\frac{9n}{10}$	n
10	98	98	91	63	36	8	0	0	19	84	6
20	100	100	95	74	23	0	0	0	1	68	0
30	100	100	97	84	28	0	0	0	0	65	0
40	100	100	100	91	22	0	0	0	0	62	0
50	100	100	100	95	19	0	0	0	0	58	0
100	100										

2.3 Discussion

As we would expect, the bottom left of all the above tables has the highest numbers, this is because as n increases, n^2 dominates $C \cdot n^{5/4} \cdot r \cdot \log(n)$ (see (7)). For example, in table 1 when $r = 1$ and $n = 100$, X^* always has the correct rank, this is because $100^{5/4} \cdot 1 \cdot \log(100) \approx 1456.28 < m = 100^2/4 = 2500$, whereas for $n = 30$, we have $30^{5/4} \cdot 1 \cdot \log(30) \approx 238.80 > m = 30^2/4 = 225$. For $n = 40$ and $n = 50$, we can attribute the low probability to the numerical constant, C, which is probably slightly greater than 1.

However, unexpectedly, the columns on the right side of every table where the rank is high but the algorithm correctly estimates the rank very often. Specifically, in table 1, we note the column $r = 4n/10$, in table 2 the column $r = 6n/10$, in table 3, the column $r = 8n/10$ and in table 4 the column $r = 9n/10$. In these columns, we get a "broken clock" correctness, as the estimates of the rank tends to stay near these columns for all matrices that the algorithm never estimates correctly. The broken clock analogy is apt here, because a broken clock is incidentally right twice a day, just like the rank estimate is accurate only incidentally. For example, when $m = n^2/4$, we get the average ranks in table 5. As the table shows, the average guess, in the columns where the estimate is almost never correct, is about $4 \cdot n/10$. Similarly for the other values of m. This shortcoming in the algorithm can be overcome, or at least avoided, by having a little advanced information, such as an estimated rank of the matrix we are trying to recover and understanding the general behavior of the estimate for a certain matrix size and number of observations.

Table 5: Average Estimated Rank, over 100 runs, $m = \frac{n^2}{4}$

$\frac{4n}{10}$	$n \setminus r$	1	$\frac{n}{10}$	$\frac{2n}{10}$	$\frac{3n}{10}$	$\frac{4n}{10}$	$\frac{5n}{10}$	$\frac{6n}{10}$	$\frac{7n}{10}$	$\frac{8n}{10}$	$\frac{9n}{10}$	n
4	10	4.88	4.71	5.53	5.51	5.54	5.49	5.52	5.71	6.04	5.77	5.81
8	20	4.86	7.33	7.84	8.20	8.70	8.46	8.77	8.84	8.99	9.00	9.05
12	30	3.84	10.76	11.34	11.69	12.33	12.40	12.46	12.76	12.58	13.00	12.94
16	40	2.41	14.06	15.27	15.90	16.17	16.59	16.65	16.56	16.88	17.17	17.41
20	50	1.81	17.76	19.13	20.03	20.74	20.77	21.20	21.76	21.27	21.75	21.83
40	100	1.03	36.14	40.17	41.94	42.72	44.40	44.83	44.33	45.62	44.72	46.20

References

[1] Netflix prize [online]. Available: http://www.netflixprize.com/, May 2015.

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

[3] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[4] E. Candès and B. Recht. Exact low-rank matrix completion via convex optimization. *Foundations of Computational Mathematics*, 9.6, 2009.

[5] E. Candès and T. Tao. The power of convex relaxation: Near-optimal matrix completion. In *Information Theory: IEEE Transactions on*, volume 56.5, pages 2053–2080, 2010.

[6] H. Chao. *First-Order Methods for Trace Norm Minimization*. PhD thesis, University of California, Los Angeles, 2013.

[7] T. Do, T. Chen, L. Gan, and T. Tran. A fast and efficient heuristic nuclear norm algorithm for affine rank minimization. In *Proceedings of the 2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2009.

[8] M. Fazel. *Matrix Rank Minimization With Applications*. PhD thesis, Stanford University, 2002.

[9] M. Fazel, H. Hindi, and S. Boyd. A rank minimization heuristic with application to minimum order system approximation. In *2001 American Control Conference*, volume 6, 2001.

[10] M. Fazel, H. Hindi, and S. Boyd. "rank minimization and applications in system theory.". In *American Control Conference, 2004. Proceedings of the 2004*, volume 4. IEEE, 2004.

[11] M. Grant, S. Boyd, and Y. Yinyu. *CVX: MATLAB Software for Disciplined Convex Programming*, 2008.

[12] M. Jaggi and M. Sulovsk. "a simple algorithm for nuclear norm regularized problems.". In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010.

[13] Z. Liu and L. Vandenberghe. Interior-point method for nuclear norm approximation with application to system identification. *SIAM Journal on Matrix Analysis and Applications*, 31.3:1235–1256., 2009.

[14] B. Recht. A simpler approach to matrix completion. *The Journal of Machine Learning Research*, 12:3413–3430, 2011.

[15] S. Shalev-Shwartz, A. Gonen, and O. Shamir. Large-scale convex minimization with a low-rank constraint. *arXiv preprint arXiv:1106.1622*, 2011.

[16] L. Vandenberghe and S. Boyd. Semidefinite programming. *SIAM review*, 38.1:49–95, 1996.

[17] C. Virginia. The singular value decomposition: Its computation and some applications. In *IEEE TRANSACTIONS ON AUTOMATIC CONTROL*, volume 25.2, 1980.

[18] Zhaoxia Yu. Stat200c: Review of linear algebra [online. Available: http://www.ics.uci.edu/∼zhaoxia/teaching/stat200c/ReviewLinearAlg.pdf, May 2015.