Convex and Nonsmooth Optimization
HW9

Sébastien Kleff - sk8001

03/31/2020

My commented MATLAB codes are available in appendix. I talked with Avadesh Meduri about this assignment, in particular we actively collaborated for Problems 1, 5 and 8.

Problem 1

Let σ be a non-zero eigenvalue of A. There exist singular vectors $(u, v) \in \mathbb{R}^p \times \mathbb{R}^q$ such that

$$\begin{aligned} Au &= \sigma v \\ A^T v &= \sigma u \end{aligned}$$

Then we have

$$\begin{aligned} B \begin{bmatrix} u \\ v \end{bmatrix} &= \begin{bmatrix} 0 & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} A^T v \\ Au \end{bmatrix} \\ &= \sigma \begin{bmatrix} u \\ v \end{bmatrix} \end{aligned}$$

So σ is a non-zero eigenvalue of B. Similarly, consider

$$\begin{aligned} B \begin{bmatrix} u \\ -v \end{bmatrix} &= \begin{bmatrix} 0 & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} u \\ -v \end{bmatrix} = \begin{bmatrix} -A^T v \\ Au \end{bmatrix} \\ &= \sigma \begin{bmatrix} -u \\ v \end{bmatrix} \\ &= -\sigma \begin{bmatrix} u \\ -v \end{bmatrix} \end{aligned}$$

So $-\sigma$ is also a nonzero eigenvalue of B. Since $\text{Rank}(A) = q$, there are q such $\sigma \neq 0$, so we exhibited $2q$ nonzero eigenvalues of B. In fact we could have shown this without explicitly constructing the eigenvectors of B by using B^2. We will now use this approach to show that the remaining $p+q-2q = p-q$ eigenvalues of B must be 0. Let us denote $\text{eig}(B) = \{\lambda_1(B), ..., \lambda_{p+q}(B)\}$ the spectrum of B. Since B is symmetric we have

$$\text{eig}(B^2) = \{\lambda_1(B)^2, ..., \lambda_{p+q}(B)^2\}$$

(This is easily verified by diagonalizing B) where

$$B^2 = \begin{bmatrix} A^T A & 0 \\ 0 & AA^T \end{bmatrix}$$

Therefore $\text{eig}(B^2) = \text{eig}(A^T A) \bigcup \text{eig}(AAT)$. In other words, $\{\lambda_1(B)^2, ..., \lambda_{p+q}(B)^2\}$ are exactly the eigenvalues of $A^T A$ and AA^T, i.e. the squares of the singular values of A. So for any $\lambda \in \mathbb{R}$, $\lambda^2 \in \text{eig}(B^2)$ if and only if there exist some singular value of A, say σ, such that $\lambda^2 = \sigma^2$, which shows that $\lambda = \pm\sigma$ (the previous result). Now since A has rank q, there are q such nonzero σ. The remaining $p - q$ singular values of A (or equivalently, the remaining $p - q$ eigenvalues of AA^T) are 0. Therefore B has $p - q$ zero eigenvalues. Note finally that

$$(u, v)^T \in \text{Ker}(B) \iff u \in \text{Ker}(A) \ \wedge v \in \text{Ker}(A^T)$$

Problem 2

(a)

In class we proved that $\|X\|_{2,d} = \|X\|_*$, where by definition $\|X\|_{2,d} = d^*$ is the optimal value of the dual SDP (D) and $\|X\|_*$ is the nuclear norm. In particular we exhibited the primal feasible point

$$W = \begin{bmatrix} U\Sigma U^T & U\Sigma V^T \\ V\Sigma U^T & V\Sigma V^T \end{bmatrix}$$

and we showed that its corresponding primal value is $\|X\|_* = \mathrm{Tr}(\Sigma)$. Note that to guarantee strong duality of an SDP, it is sufficient that Slater's condition holds for either one of the primal and dual problems. Notice that for instance $Y = 0$ is strictly feasible for (D), so Slater's condition holds for the dual SDP and thus strong duality holds, i.e. $d^* = p^*$. Since we already know that $\|X\|_* = d^*$, we have $\|X\|_* = p^*$. In conclusion, W is optimal for (P).

(b)

Let $Y = UV^T$ where U, V are orthogonal matrices achieving the SVD of X. Let us show that Y is feasible for (D), that is

$$\begin{bmatrix} I_m & Y \\ -Y^T & I_n \end{bmatrix} \succeq 0$$

We can use the characterization of the 2-norm shown in class with $t = 1$: notice first that the SVD of Y is trivially $Y = UIV^T$, therefore the maximum singular value of Y is 1, that is $\|Y\|_2 = 1$. Thus $\|Y\|_2 \leq 1$, which is equivalent to

$$\begin{bmatrix} I_m & Y \\ -Y^T & I_n \end{bmatrix} \succeq 0$$

by the characterization of the 2-norm. Hence Y is feasible for (D). Let us show now that Y is optimal for (D). We simply calculate the corresponding dual objective value

$$\begin{aligned} \mathrm{Tr}(X^T Y) &= \mathrm{Tr}(X^T UV^T) \\ \mathrm{Tr}(X^T Y) &= \mathrm{Tr}(V\Sigma U^T UV^T) \\ \mathrm{Tr}(X^T Y) &= \mathrm{Tr}(V\Sigma V^T) \\ \mathrm{Tr}(X^T Y) &= \mathrm{Tr}(\Sigma) \\ \mathrm{Tr}(X^T Y) &= \|X\|_* \end{aligned}$$

Therefore $\mathrm{Tr}(X^T Y)$ equals the optimal primal objective value p^* mentioned previously. Since strong duality holds, Y is optimal for (D).

(c)

We check the complementarity between the primal matrix W and the dual slack matrix S associated with (D'')

$$S = \frac{1}{2}\begin{bmatrix} I_m & 0 \\ 0 & I_n \end{bmatrix} - \sum_{i,j} y_{ij} E_{ij}$$
$$S = \frac{1}{2}\begin{bmatrix} I_m & -Y \\ -Y^T & I_n \end{bmatrix}$$

We simply calculate the product WS

$$WS = \frac{1}{2}\begin{bmatrix} U\Sigma U^T & U\Sigma V^T \\ V\Sigma U^T & V\Sigma V^T \end{bmatrix}\begin{bmatrix} I_m & -Y \\ -Y^T & I_n \end{bmatrix}$$
$$WS = \frac{1}{2}\begin{bmatrix} U\Sigma U^T I_m - U\Sigma V^T Y^T & -U\Sigma U^T Y + U\Sigma V^T I_n \\ V\Sigma U^T I_m - V\Sigma V^T Y^T & -V\Sigma U^T Y + V\Sigma V^T I_n \end{bmatrix}$$
$$WS = \frac{1}{2}\begin{bmatrix} U\Sigma U^T - U\Sigma V^T V U^T & -U\Sigma U^T U V^T + U\Sigma V^T \\ V\Sigma U^T - V\Sigma V^T V U^T & -V\Sigma U^T U V^T + V\Sigma V^T \end{bmatrix}$$
$$WS = \frac{1}{2}\begin{bmatrix} U\Sigma U^T - U\Sigma U^T & -U\Sigma V^T + U\Sigma V^T \\ V\Sigma U^T - V\Sigma U^T & -V\Sigma V^T + V\Sigma V^T \end{bmatrix}$$
$$WS = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Therefore complementarity is checked.

CVX verification

The MATLAB code corresponding to this question is implemented as *pb2.m* in Appendix A. We generated a random matrix $X \in \mathbb{R}^{3\times 2}$

$$X = \begin{bmatrix} 0.9109 & -0.1020 \\ 0.0397 & 0.7144 \\ -0.4276 & 0.3110 \end{bmatrix}$$

We performed the SVD of X with $\text{svd}(X, 0)$ and obtained

$$U = \begin{bmatrix} -0.8415 & -0.3463 \\ 0.2255 & -0.9227 \\ 0.4910 & -0.1697 \end{bmatrix}$$
$$V = \begin{bmatrix} -0.9242 & -0.3818 \\ 0.3818 & -0.9242 \end{bmatrix}$$
$$\Sigma = \begin{bmatrix} 1.0468 & 0 \\ 0 & 0.7321 \end{bmatrix}$$

We solved the primal and dual SDPs (P) and (D). As expected, strong duality is checked

$$p^* = 1.7789$$
$$d^* = 1.7789$$

and the optimal primal and dual matrices W and Y are the same as the one mentioned in Questions (a) and (b)

$$W = \begin{bmatrix} 0.8290 & 0.0353 & -0.3895 & 0.9109 & -0.1020 \\ 0.0353 & 0.6764 & 0.2305 & 0.0397 & 0.7144 \\ -0.3895 & 0.2305 & 0.2735 & -0.4276 & 0.3110 \\ 0.9109 & 0.0397 & -0.4276 & 1.0009 & -0.1111 \\ -0.1020 & 0.7144 & 0.3110 & -0.1111 & 0.7779 \end{bmatrix}$$
$$Y = \begin{bmatrix} 0.9099 & -0.0012 \\ 0.1439 & 0.9389 \\ -0.3890 & 0.3443 \end{bmatrix}$$

Furthermore, complementarity is also verified as expected from Question (c)

$$WS = 1.0e-08 * \begin{bmatrix} 0.2145 & 0.0052 & -0.0229 & 0.0646 & 0.0029 \\ 0.0150 & 0.2212 & 0.0199 & 0.0048 & 0.0635 \\ -0.0336 & 0.0294 & 0.1524 & -0.0296 & 0.0218 \\ 0.0695 & -0.0001 & -0.0181 & 0.2258 & 0.0064 \\ -0.0004 & 0.0660 & 0.0169 & -0.0026 & 0.2282 \end{bmatrix}$$
$$WS \simeq 0$$

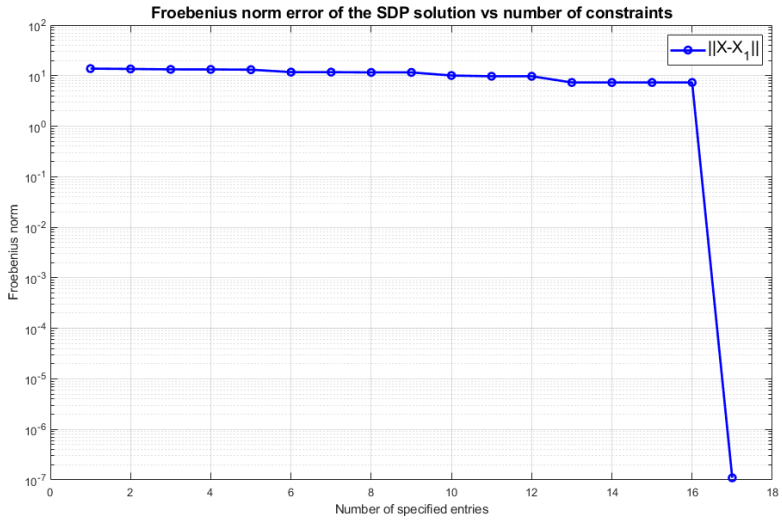


Figure 1: Log plot of Froebenius norm error over the sequence of SDPs for X_1. Notice the sharp drop when 16 entries are specified which indicates that the original matrix has been approximately recovered within the tolerance 10^{-6}.

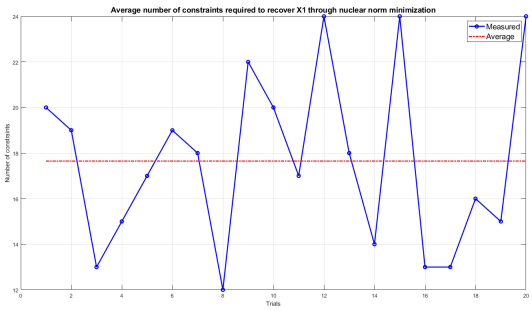
Problem 3

The MATLAB code corresponding to this question is implemented as *pb3.m* and *run_SDPs.m* in Appendix B. We consider the following SDP

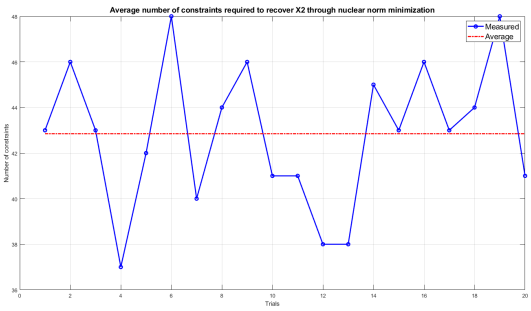
$$\begin{aligned} \min_{W_1,W_2} \quad & \frac{1}{2}(\mathrm{Tr}(W_1) + \mathrm{Tr}(W_2)) \\ \text{s.t.} \quad & \begin{bmatrix} W_1 & X \\ X^T & W_2 \end{bmatrix} \succeq 0 \\ & X_{ij} = m_{ij} \end{aligned}$$

We run a sequence of such SDPs adding 1 new constraint each time. At each SDP in the sequence, the pair of indices (i, j) describing the newly specified entry is drawn randomly from $[1, m] \times [1, N]$ without repetition, i.e. we stop if the full matrix is constrained (as the full original matrix is then recovered). Besides, we introduce a stopping criteria based on the Froebenius norm $\|X - X^*\|_F$ in order to decide whether the SDP solution X is close enough the original matrix X^* (where $X^* = X_1, X_2$ or X_3 from the data set). The tolerance is set to 10^{-6}. Figure 1 shows the Froebenius norms error of the solutions of the SDPs associated with X_1 in a log plot.

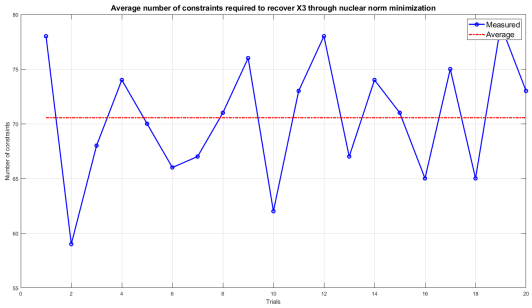
In order to average our tests, we perform 20 trials for each of the data matrices X_1, X_2, X_3, i.e. we solve 20 times the sequence of increasingly constrained SDPs described above. In particular, we are interested in the average number of entries that are required to recover the original matrix, namely $\bar{N}_1, \bar{N}_2$ and $\bar{N}_3$. We obtain



(a) Number of entries required to recover X_1



(b) Number of entries required to recover X_2



(c) Number of entries required to recover X_3

Figure 2: 20 trials performed on X_1, X_2, X_3.

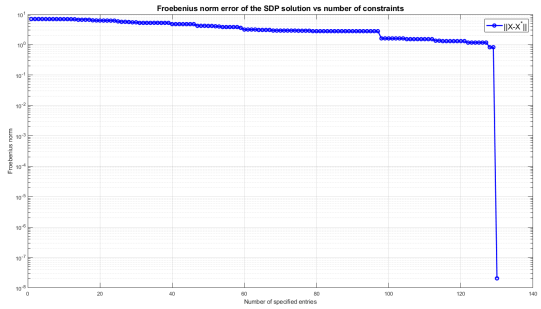
$$\bar{N}_1 = 17.65$$
$$\bar{N}_2 = 42.85$$
$$\bar{N}_2 = 70.55$$

We plot the results of these trials in Figures 2a,2b,2c. We can see that the average number of specified entries required to achieve a good reconstruction is about 70% of the total number of entries $mn = 25$ for X_1, 86% for X_2 (50 entries) and 88% for X_3 (80 entries).

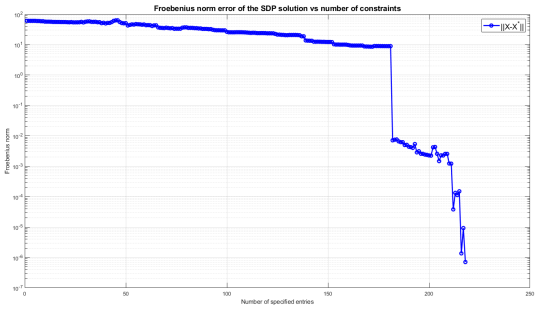
By performing SVD on X_1, X_2, X_3, we notice that these matrices have respective ranks 1, 2 and 3. This previous analysis suggests that more specified entries are required to recover the full data matrix for higher ranks. This makes sense since we are minimizing the nuclear norm which encourages low rank. If the low rank assumption on the original matrix is not accurate then it will be harder to recover it through approximate rank minimization (i.e. in our case through the nuclear norm minimization).

Problem 4

The code corresponding to this section is available in (cf. Appendix B), we generate random matrices $X_{\text{low}}, X_{\text{high}} \in \mathbb{R}^{q \times q}$ with rank 3 and 10 respectively. We fix $q = 15$ such that the matrices have 225 entries (which yields a long but reasonable computation time in both cases) and run the sequence of SDPs with increasing number of specified entries as previously. The results are shown in Figures 3a, 3b for a tolerance of 10^{-6} on the Froebenius norm error.



(a) Log plot of the Froebenius norm error for X_{low} (rank 3). 130 entries are required to recover the full matrix.



(b) Number of entries required to recover X_{high} (rank 10). 218 entries are required to recover the full matrix.

Figure 3: Log plots of Froebenius norm error for the nuclear norm minimization to recover X_{low} (rank 3) and X_{high} (rank 10).

Again, we observe a sharp drop in the Froebenius norm error once a certain threshold has been reached in the number of constraints. Furthermore, as expected, this threshold is larger when the rank is larger: we need to specify 130 entries in order to approximately recover X_{low} (rank 3) whereas 218 entries are required to recover X_{high} (rank 10), i.e. nearly all of them. Repeated trials confirm this trend : the nuclear norm minimization works better, in a sense that fewer constraints are required to recover the data, when the low rank assumption is satisfied.

Problem 5

The ADMM algorithm for SDP and the code for this problem are available in Appendix C. We first need to write the nuclear norm minimization SDP under ADMM form. The original form of the SDP is

$$\begin{aligned} \min_{W_1,W_2} \ & \frac{1}{2}(\mathrm{Tr}(W_1)+\mathrm{Tr}(W_2)) \\ \text{s.t.} \ & \begin{bmatrix} W_1 & X \\ X^T & W_2 \end{bmatrix} \succeq 0 \\ & \mathcal{A}(X) = b \end{aligned}$$

where $\mathcal{A}:\mathbb{R}^{m\times n}\to\mathbb{R}^p$ is a linear map and $b\in\mathbb{R}^p$, with p the number of constraints: for any $k\in[1,p]$ we have $< A_k,\ X >= b_k$. This equality constraint encodes the specification of p entries $X_{ij}=m_{ij}$ for some p pairs of indices $(i,j)\in[1,m]\times[1,n]$: if the k^{th} constraint represents the specification of entry (i,j), we define $b_k=m_{ij}$ and $A_k\in\mathbb{R}^{m\times n}$ to be 0 everywhere except for $(A_k)_{ij}=1$. Consequently, the inner product of A_k with X then simply selects the entry X_{ij}

$$< A_k,\ X >= \mathrm{Tr}(A_k^T X) = X_{ij}$$

But in order to put the system under ADMM form we need to express this constraint on the full design variable (not solely X). Thus let us define the optimization variable $x\in\mathbb{R}^{(m+n)\times(m+n)}$ and the linear map $\tilde{\mathcal{A}}:\mathbb{R}^{(m+n)\times(m+n)}\to\mathbb{R}^p$ as

$$\begin{aligned} x &= \begin{bmatrix} W_1 & X \\ X^T & W_2 \end{bmatrix} \\ \tilde{A}_k &= \frac{1}{2}\begin{bmatrix} 0 & A_k \\ A_k^T & 0 \end{bmatrix} \end{aligned}$$

So that

$$< \tilde{A}_k,\ x >= \frac{1}{2}(\mathrm{Tr}(A_k X^T)+\mathrm{Tr}(A_k^T X)) = X_{ij}$$

Then $< \tilde{A}_k,\ x >= b_k$ is equivalent to $< A_k,\ X >= b_k$, and the problem can be now put under ADMM form as

$$\begin{aligned} \min_x \ & f(x)+g(z) \\ \text{s.t.} \ & x - z = 0 \\ & \tilde{\mathcal{A}}(x) = b \end{aligned}$$

where $f(x)=\mathrm{Tr}(x)= \ < I_{m+n},\ x >$[1] and $g(z)=\mathcal{I}_{\mathcal{S}_+^{n+m}}(z)$ (indicator function of the positive semi-definite cone). The augmented Lagrangian to minimize during the primal update of ADMM (under scaled form) is

[1] $< .,. >$ denotes the inner product associated with the Froebenius norm without ambiguity.

$$L(x,\nu,z,u) = f(x) + g(z) + \frac{\rho}{2}\|x - z + u\|_F + \sum_k \nu_k < \tilde{A}_k,\ x > -\nu^T b$$
$$L(x,\nu,z,u) = f(x) + g(z) + \frac{\rho}{2}\|x - z + u\|_F + < \sum_k \nu_k \tilde{A}_k,\ x > -\nu^T b$$
$$L(x,\nu,z,u) = f(x) + g(z) + \frac{\rho}{2}\|x - z + u\|_F + < \tilde{\mathcal{A}}^*(\nu),\ x > -\nu^T b$$

Notice that since the problem is constrained over x we included the Lagrange multiplier $\nu \in \mathbb{R}^p$. Hence the ADMM algorithm will include a KKT solve step during the update of the primal variable x. The primal and dual optimality conditions on x for this SDP are given by[2]

$$\nabla_x L(x,\nu,z,u) = I_{m+n} + \rho(x - z + u) + \tilde{\mathcal{A}}^*(\nu) = 0$$
$$\tilde{\mathcal{A}}(x) = b$$

In order to write out the above conditions under the form of a linear system as for LPs[3] we "vectorize" the above equations and cast them into an $(m+n)^2 + p$ matrix H

$$\underbrace{\begin{bmatrix} \rho I_{(m+n)^2} & \text{vec}(\tilde{\mathcal{A}}_1) & \dots & \text{vec}(\tilde{\mathcal{A}}_p) \\ \text{vec}(\tilde{\mathcal{A}}_1)^T & & & \\ \dots & & 0_{p\times p} & \\ \text{vec}(\tilde{\mathcal{A}}_p)^T & & & \end{bmatrix}}_{H} \begin{bmatrix} \text{vec}(x) \\ \nu \end{bmatrix} = \begin{bmatrix} \text{vec}(z - u - \frac{1}{\rho} I_{m+n}) \\ b \end{bmatrix}$$

Moreover we need also to project z onto the cone of p.s.d. matrices. This amounts to change the z update in the algorithm (we simply "clamp" the negative eigenvalues to 0 in the diagonalized form). Therefore the full ADMM iteration is

$$(x^{k+1}, \nu^{k+1}) = \arg\min_{x,\nu}\ L(x,\nu,z^k,u^k)$$
$$z^{k+1} = \Pi_{\mathcal{S}^n_+}(x^{k+1} + u^k)$$
$$u^{k+1} = u^k + x^{k+1} - z^{k+1}$$

where the x update is obtained by solving the big KKT system defined previously

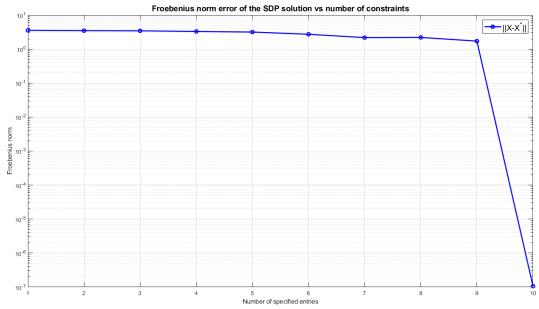
$$\begin{bmatrix} \text{vec}(x) \\ \nu \end{bmatrix} = H^{-1} \begin{bmatrix} \text{vec}(z - u - \frac{1}{\rho} I_{m+n}) \\ b \end{bmatrix}$$

Instead of inverting H directly we use MATLAB's function "\" and then the vector $\text{vec}(x)$ is reshaped into an $m + n$ matrix to carry on the z and u updates (see in my code).

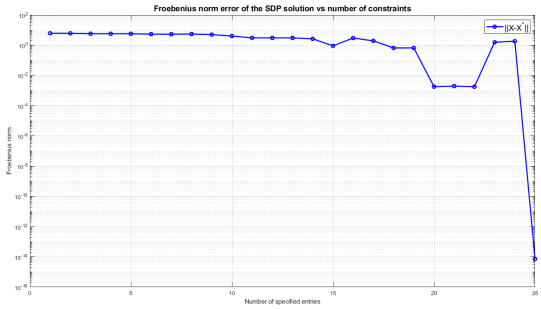
For each ADMM run, the stopping condition is implemented as in the previous homework[4] and set the relative tolerance to 10^{-8} and the absolute tolerance to 10^{-8}. The maximum number of iterations is set

[2] We use the property that $\nabla_X \text{Tr}(A^T X) = A^T$.
[3] ADMM paper, Section 5.2, p.37.
[4] cf. Boyd's ADMM paper, Section 3.3.1 p. 19

to 500. The sequence of SDPs is implemented as in Problem 3, that is we increase the number of equality constraints (i.e. specified entries) at each iteration and stop the algorithm whenever the Froebnius norm error $\|X^* - X\|_F$ reaches the recovery tolerance (set to 10^{-5}). We generate randomly a matrix X^* of size 5 and rank 1. Figure 4a shows that only 10 entries are required to recover the full original matrix. In fact I found out during my experiments that the number of entries required to approximately recover the initial matrix depends heavily on the ADMM parameters, in particular on the tolerance. If we allow a large number of iterations per SDP with a strict tolerance (e.g. our current parameters), the solution computed through ADMM will be accurate - but it may naturally take a longer time to compute - and the initial matrix is recovered with fewer entries. On the contrary, if we set loose tolerances on each SDP, say 10^{-4} for both relative and absolute tolerances and 100 iterations at most, we obviously obtain much faster a less accurate solution. However, as shown in Figure 4b, in this case all entries (25) are required to recover the initial matrix (i.e. the number of SDP to be solved equals the number of entries in the initial matrix).



(a) We set small tolerances on each ADMM run (10^{-8}) in order to produce accurate solutions. Only 10 entries are required to recover the matrix.

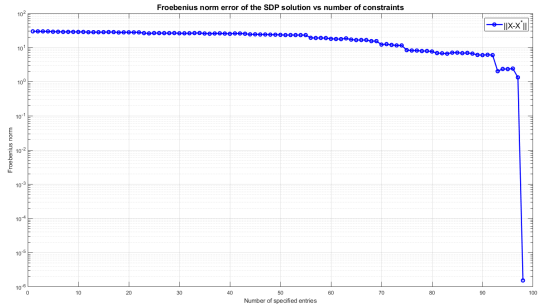


(b) We set loose tolerances on each ADMM run (10^{-4}) in order to produce quickly moderately accurate solutions. All entries (25) are required to recover the matrix.

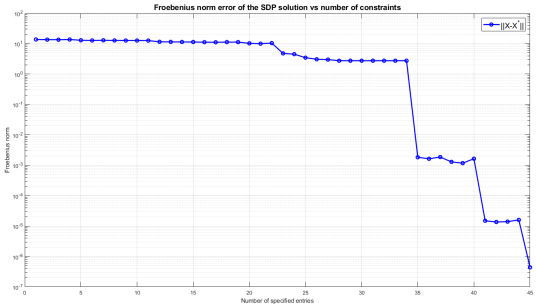
Figure 4: The number of entries required to recover the original matrix is very sensitive to the ADMM accuracy.

Now using allowing ADMM to be very accurate (regardless of computation time considerations) we can see that the observations made earlier about the rank still hold: matrices with lower rank will require fewer specified entries than matrices with higher rank in order to be approximately recovered. Figures 5a and 5b show the number of iterations required to recover an initial matrix of rank 10 and 1 respectively. We can see that the matrix of rank 10 is recovered within the Freobenius norm error tolerance with all entries (100) specified, whereas the rank 1 matrix only requires 45 entries.

In my experience ADMM turned out to be less practical than CVX in a sense that for the same matrix recovery tolerance, ADMM needs to be very accurate and thus longer to compute than CVX. At the same time, a less strict ADMM requires nearly all entries to be specified in order to meet the Froebenius norm error threshold. This suggests that recovering matrices with only few entries requires to solve each SDP with an accuracy that ADMM's slow convergence rate doesn't practically allow. In fact this trade off between the computation time and the number of entries to specify becomes even more acute when dealing with larger matrices (I couldn't go beyond size 15 while keeping the computation time reasonable).



(a) Log plot of the Froebenius norm error for a matrix of size 10 and rank 10. All entries are required to recover the full matrix.



(b) Log plot of the Froebenius norm error for a matrix of size 10 and rank 1. Only e entries are required to recover the full matrix.

Figure 5: Log plots of Froebenius norm error for the nuclear norm minimization to recover matrices of rank 10 and 1 respectively.

1 Problem 6

Let us denote $C = I_{m+n}$. The original SDP is

$$\begin{aligned} \min_x \ \mathrm{Tr}(x) = \ & < C,\ x > \\ \text{s.t. } & x \succeq 0 \\ & \tilde{\mathcal{A}}(x) = b \end{aligned}$$

where as in Problem 5 we use the notation

$$x = \begin{bmatrix} W_1 & X \\ X^T & W_2 \end{bmatrix}$$
$$\tilde{A}_k = \frac{1}{2}\begin{bmatrix} 0 & A_k \\ A_k^T & 0 \end{bmatrix}$$

The Lagrangian is given by

$$\begin{aligned} L(x, \Lambda, \nu) = \ & < C,\ x > - < \Lambda,\ x > + \sum_k \nu_k (< \tilde{A}_k,\ x > - b_k) \\ L(x, \Lambda, \nu) = \ & < C - \Lambda + \tilde{\mathcal{A}}^*(\nu),\ x > - \nu^T b \end{aligned}$$

where $\Lambda \in \mathbb{R}^{m \times n}, \nu \in \mathbb{R}^p$ and $\tilde{\mathcal{A}}^* : \mathbb{R}^p \to \mathbb{R}^{m \times n}$ is the dual of $\tilde{\mathcal{A}}$. The Lagrange dual function is then

$$\begin{aligned} g(\Lambda, \nu) &= \inf_x \ < C - \Lambda + \tilde{\mathcal{A}}^*(\nu),\ x > - \nu^T b \\ g(\Lambda, \nu) &= \begin{cases} -\nu^T b & \text{if } \tilde{\mathcal{A}}^*(\nu) = \Lambda - C \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

Thus the dual problem is

$$\begin{aligned} \max_{\Lambda \succeq 0,\ \nu} \ & - b^T \nu \\ \text{s.t. } & \tilde{\mathcal{A}}^*(\nu) + C = \Lambda \end{aligned}$$

which is equivalent to

$$\begin{aligned} \max_{\nu} \ & - b^T \nu \\ \text{s.t. } & \tilde{\mathcal{A}}^*(\nu) + C \succeq 0 \end{aligned}$$

Recalling the definition of $\tilde{\mathcal{A}}^*(\nu)$ we get

$$
\begin{aligned}
\tilde{\mathcal{A}}^*(\nu) &= \sum_k \nu_k \tilde{A}_k \\
\tilde{\mathcal{A}}^*(\nu) &= \sum_k \nu_k \begin{bmatrix} 0 & A_k \\ A_k^T & 0 \end{bmatrix} \\
\tilde{\mathcal{A}}^*(\nu) &= \begin{bmatrix} 0 & \sum_k \nu_k A_k \\ \sum_k \nu_k A_k^T & 0 \end{bmatrix} \\
\tilde{\mathcal{A}}^*(\nu) &= \begin{bmatrix} 0 & A^*(\nu) \\ A^*(\nu)^T & 0 \end{bmatrix}
\end{aligned}
$$

Injecting the above result in the dual problem and replacing $C = I_{m+n}$ we get

$$
\begin{aligned}
\sup_z \quad & -b^T z \\
\text{s.t.} \quad & \begin{bmatrix} I_m & A^*(z) \\ A^*(z)^T & I_n \end{bmatrix} \succeq 0
\end{aligned}
$$

Now changing z to $-z$, and noticing that changing $A^*(-z) = -A^*(z)$ to $-A^*(-z) = A^*(z)$ doesn't affect the positive semi-definiteness of the matrix, we finally recover the expected form

$$
\begin{aligned}
\sup_z \quad & b^T z \\
\text{s.t.} \quad & \begin{bmatrix} I_m & A^*(z) \\ A^*(z)^T & I_n \end{bmatrix} \succeq 0
\end{aligned}
$$

Problem 7

We use the Table 2.1, p.476 and show the following inequalities for the norms $\|.\|_1, \|.\|_2$ and $\|.\|_\infty$ for any $x \in \mathbb{R}^n$

$$\|x\|_\infty \leq \|x\|_2 \leq \|x\|_1 \leq \sqrt{r}\|x\|_2 \leq r\|x\|_\infty$$

where r is the cardinality of x.

Proof of $\|x\|_\infty \leq \|x\|_2$

We have

$$\begin{aligned}
\|x\|_\infty^2 &= (\max_i \ |x_i|)^2 \\
\|x\|_\infty^2 &\leq \sum_{i=1}^{r} x_i^2 \qquad \text{since we add non-negative terms} \\
\|x\|_\infty^2 &\leq \|x\|_2^2
\end{aligned}$$

So $\|x\|_\infty \leq \|x\|_2$.

Proof of $\|x\|_2 \leq \|x\|_1$

We have

$$\begin{aligned}
\|x\|_1^2 &= (\sum_{i=1}^{r} |x_i|)^2 \\
\|x\|_1^2 &= \sum_{i,j=1}^{r} |x_i||x_j| \\
\|x\|_1^2 &\geq \sum_{i=j} |x_i||x_j| \qquad \text{since all cross-terms } |x_i||x_j|, i \neq j \text{ are positive} \\
\|x\|_1^2 &\geq \sum_{i=1}^{r} |x_i|^2 \\
\|x\|_1^2 &\geq \|x\|_2^2
\end{aligned}$$

So $\|x\|_2 \leq \|x\|_1$.

Proof of $\|x\|_1 \leq \sqrt{r}\|x\|_2$

We define $\bar{1} \in \mathbb{R}^n$ such that for any $i \in [1, n]$, $\bar{1}_i = 1$ if $x_i \neq 0$ and $\bar{1}_i = 0$ if $x_i = 0$. Also we denote $|x| \in \mathbb{R}^n$ the vector of $|x_i|$'s.

$$\begin{aligned}
\|x\|_1 &= \bar{1}^T|x| \\
\|x\|_1 &= \ < \bar{1}, |x| > \\
\|x\|_1 &\leq \|\bar{1}\|_2\|x\|_2 \qquad \text{by the Cauchy-Schwarz inequality} \\
\|x\|_1 &\leq \sqrt{r}\|x\|_2
\end{aligned}$$

Proof of $\sqrt{r}\|x\|_2 \leq r\|x\|_\infty$

We have

$$\begin{aligned}
\|x\|_2 &= \sqrt{\sum_{i=1}^{r} x_i^2} \\
\|x\|_2 &\leq \sqrt{\sum_{i=1}^{r} \|x\|_\infty^2} \qquad \text{since } |x_i| \leq \|x\|_\infty \implies x_i^2 \leq \|x\|_\infty \\
\|x\|_2 &\leq \sqrt{r\|x\|_\infty^2} \\
\|x\|_2 &\leq \sqrt{r}\|x\|_\infty
\end{aligned}$$

So $\sqrt{r}\|x\|_2 \leq r\|x\|_\infty$.

Now that we proved these inequalities for vector norms, we can extend the same tricks to matrix norms. Let $X \in \mathbb{R}^{m\times n}$ with rank r. We shall prove that

$$\|X\| \leq \|X\|_F \leq \|X\|_* \leq \sqrt{r}\|X\|_F \leq r\|X\|$$

where the matrix norms are defined as

$$\begin{aligned}
\|X\| &= \max_i \ \sigma_i(X) \\
\|X\|_F &= \sqrt{\mathrm{Tr}(X^T X)} = \sqrt{\sum_{i,j} X_{ij}^2} = \sqrt{\sum_{i=1}^{r} \sigma_i(X)^2} \\
\|X\|_* &= \sum_{i=1}^{r} \sigma_i(X)
\end{aligned}$$

and $\sigma_i(X)$ the i^{th} singular value of X.

Proof of $\|X\| \leq \|x\|_F$

We have

$$\begin{aligned}
\|X\|^2 &= (\max_i \ \sigma_i(X))^2 \\
\|X\|^2 &\leq \sum_{i=1}^{r} \sigma_i(X)^2 \qquad \text{since we add non-negative terms} \\
\|X\|^2 &\leq \|X\|_F^2
\end{aligned}$$

So $\|X\| \leq \|X\|_F$.

Proof of $\|X\|_F \leq \|X\|_*$

We have

$$\begin{aligned}
\|X\|_*^2 &= (\sum_{i=1}^{r} \sigma_i(X))^2 \\
\|X\|_*^2 &= \sum_{i,j=1}^{r} \sigma_i(X)\sigma_j(X) \\
\|X\|_*^2 &\geq \sum_{i=j} \sigma_i(X)\sigma_j(X) \qquad \text{since all cross-terms } \sigma_i(X)\sigma_j(X), i \neq j \text{ are positive} \\
\|X\|_*^2 &\geq \sum_{i=1}^{r} \sigma_i(X)^2 \\
\|X\|_*^2 &\geq \|X\|_F^2
\end{aligned}$$

So $\|X\|_F \leq \|X\|_*$.

Proof of $\|X\|_* \leq \sqrt{r}\|X\|_F$

We denote the SVD of X as $X = U\Sigma V^T$.

$$\begin{aligned}
\|X\|_*^2 &= \text{Tr}(\Sigma) \\
\|X\|_*^2 &= \text{Tr}(U^T U \Sigma V^T V) \qquad \text{since } U, V \text{ are orthogonal matrices} \\
\|X\|_*^2 &= \text{Tr}(U^T X^T V) \\
\|X\|_*^2 &= \text{Tr}(X^T V U^T) \qquad \text{property of the trace} \\
\|X\|_*^2 &\leq \|VU^T\|_F \|X\|_F \qquad \text{by the Cauchy-Schwarz inequality} \\
\|X\|_*^2 &\leq r\|X\|_F
\end{aligned}$$

so $\|X\|_* \leq \sqrt{r}\|X\|_F$.

Proof of $\sqrt{r}\|X\|_F \leq r\|X\|$

We have

$$\begin{aligned}
\|X\|_F &= \sqrt{\sum_{i=1}^{r} \sigma_i(X)^2} \\
\|X\|_F &\leq \sqrt{\sum_{i=1}^{r} \|X\|^2} \qquad \text{since } \sigma_i(X) \leq \|X\| \implies \sigma_i(X)^2 \leq \|X\| \\
\|X\|_F &\leq \sqrt{r\|X\|^2} \\
\|X\|_F &\leq \sqrt{r}\|X\|
\end{aligned}$$

So $\sqrt{r}\|X\|_F \leq r\|X\|$.

Problem 8

Claim (3)

Let $n = 2$, $r = 1$ and $\mathcal{A}(X) = [X_{11}, X_{12}, X_{21}]^T$. Choosing any matrix $X \in \mathbb{R}^{m\times 2}$ of rank 1 such that $\|\mathcal{A}(X)\| = 0$ enables to show by contradiction that there is no $\delta_r < 1$: assume that there exist $\delta_r < 1$, that is $\forall X \in \mathbb{R}^{m\times 2}$ with $\text{Rank}(X) = 1$ the following inequality holds

$$(1 - \delta_r)\|X\|_F \leq \|\mathcal{A}(X)\| \leq (1 + \delta_r)\|X\|_F$$

Since $\text{Rank}(X) = 1$ let us denote by σ the only nonzero singular value of X. Then trivially $\|X\|_F = \sigma$ and also $\|\mathcal{A}(X)\| = \sqrt{X_{11}^2 + X_{12}^2 + X_{21}^2}$. Then we have

$$(1 - \delta_r)\sigma \leq \sqrt{X_{11}^2 + X_{12}^2 + X_{21}^2} \leq (1 + \delta_r)\sigma$$

In particular, since $\delta_r < 1$ and $\sigma \neq 0$ then $(1 - \delta_r)\sigma > 0$ so $\sqrt{X_{11}^2 + X_{12}^2 + X_{21}^2} > 0$ should hold for any matrix X of rank 1, which is obviously not true since we can construct such a matrix X with $X_{11} = X_{12} = X_{21} = 0$. Take for instance

$$X = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \end{bmatrix}$$

Then $\sigma = 1$ and $\|\mathcal{A}(X)\| = 0 \geq 1 - \delta_r > 0$ gives a contradiction. Therefore, in this case, there is no $\delta_r < 1$.

Claim (4)

Let[5] $n = m = 2$, $r = 1$ and $\mathcal{A}(X) = [X_{11}, X_{12} + X_{21}, X_{22}]^T$. Since X has rank 1, then there exist $\alpha \in \mathbb{R}$ such that $X_{12} = \alpha X_{11}$ and $X_{22} = \alpha X_{21}$. Therefore $X_{11}X_{22} = X_{21}X_{12}$. We have

$$\begin{aligned}
&\|\mathcal{A}(X)\|^2 = X_{11}^2 + (X_{12} + X_{21})^2 + X_{22}^2 \\
&\|\mathcal{A}(X)\|^2 = X_{11}^2 + X_{22}^2 + X_{12}^2 + X_{21}^2 + 2X_{12}X_{21} && \text{we recognize here } \|X\|_F^2 = X_{11}^2 + X_{22}^2 + X_{12}^2 + X_{21}^2 \\
&\|\mathcal{A}(X)\|^2 = \|X\|_F^2 + X_{12}X_{21} + X_{11}X_{22} && \text{we split the cross term since } X_{21}X_{12} = X_{11}X_{22} \\
&\|\mathcal{A}(X)\|^2 \leq \|X\|_F^2 + \frac{X_{12}^2 + X_{21}^2}{2} + \frac{X_{11}^2 + X_{22}^2}{2} && \text{using the quadratic inequality on the cross terms} \\
&\|\mathcal{A}(X)\|^2 \leq \|X\|_F^2 + \frac{\|X\|_F^2}{2} \\
&\|\mathcal{A}(X)\|^2 \leq \frac{3}{2}\|X\|_F^2
\end{aligned}$$

For the other inequality, we have from above development that

[5] I couldn't see how to prove the inequalities for any m since $\|X\|_F$ then depends on all the entries X_{m1}, X_{m2} that we don't know anything about, so I assumed the question was only concerned with square matrices of size 2.

$$
\begin{aligned}
\|X\|_F^2 &= \|\mathcal{A}(X)\|^2 - 2X_{12}X_{21} \\
\|X\|_F^2 &\leq \|\mathcal{A}(X)\|^2 + 2|X_{12}X_{21}| \\
\|X\|_F^2 &\leq \|\mathcal{A}(X)\|^2 + |X_{12}X_{21}| + |X_{11}X_{22}| \\
\|X\|_F^2 &\leq \|\mathcal{A}(X)\|^2 + \frac{X_{12}^2 + X_{21}^2}{2} + \frac{X_{11}^2 + X_{22}^2}{2} \quad \text{using the quadratic inequality on the cross terms} \\
\|X\|_F^2 &\leq \|\mathcal{A}(X)\|^2 + \frac{\|X\|_F^2}{2} \\
\|\mathcal{A}(X)\|^2 &\geq \frac{1}{2}\|X\|_F^2
\end{aligned}
$$

If we summarize the inequalities that we proved we get

$$
\begin{aligned}
\frac{1}{2}\|X\|_F^2 \leq \|\mathcal{A}(X)\|^2 \leq \frac{3}{2}\|X\|_F^2 \\
\frac{1}{\sqrt{2}}\|X\|_F \leq \|\mathcal{A}(X)\| \leq \sqrt{\frac{3}{2}}\|X\|_F
\end{aligned}
$$

So we can solve $\delta_r = 1 - \frac{1}{\sqrt{2}}$ so that $1 - \delta_r = \frac{1}{\sqrt{2}}$. Then we notice that $1 + \delta_r = 2 - \frac{1}{\sqrt{2}} > \sqrt{\frac{3}{2}}$ so RHS inequality still inequality holds.

A MATLAB code for Problem 2

```
%% Homework 9 – Problem 2

% Random matrix X
m = 3;
n = 2;
X = randn(m,n);

% SVD of X
[U,S,V] = svd(X,0);

% Compute primal and dual solutions using CVX
    % PRIMAL
cvx_begin sdp
    variable W(m+n,m+n) symmetric
    minimize ( .5*trace(W'*eye(m+n)) )
    W(1:m,m+1:end) == X;W
    W ≥ 0;
cvx_end
pstar = cvx_optval;

    % DUAL
cvx_begin sdp
    variable Y(m,n)
    maximize ( trace(X'*Y) )
    [eye(m), -Y; -Y', eye(n)] ≥ 0;
cvx_end
dstar = cvx_optval;

% Check strong duality
norm(svd(X),1)
pstar
dstar

% Check primal solution
X
W

% Check dual solution
U*V'
Y

% Check complementarity
W*[eye(m), -Y; -Y', eye(n)]
```

B MATLAB code for Problems 3 & 4

```
%% Homework 9 – Problemd 3 & 4

%% Problem 3
% For each of X1, X2, X3 we will perform N trials to average the results
```

```
5
6  % Each trial = run successive nuclear norm minimization SDPs, increasing
7  % the number of constraints each time. We stop the SDPs when Xi has been
8  % approximately recovered, i.e. when solution X of SDP is close enough
9  % in a sense of the 2-norm (parameter 'tol')
10
11 % The code corresponding to 1 trial is in 'run_SDPs.m'
12
13 % 2-norm tolerance on SDP solution : did we recover X1 accurately enough ?
14 tol = 1e-6;
15
16 % Load data
17 load('Xdata.mat');
18
19 % Perform a single trial on X1 to test the function and plot the Froebenius
20 % norm error across SDPs in a log plot
21 [~, ~, fnorms] = run_SDPs(X1, tol);
22 figure
23 semilogy(fnorms, 'b-o', 'LineWidth', 2);
24 grid on
25 title('Froebenius norm error of the SDP solution vs number of constraints',...
26     'FontSize', 14);
27 xlabel('Number of specified entries');
28 ylabel('Froebenius norm');
29 legend('||X-X_{1}||', ...
30        'FontSize', 14, 'Location', 'NorthEast');
31
32
33 % The following part is the analysis of repeated trials on X1, X2, X3
34 % Number of trials for averaging
35 N = 20;
36
37 %% Problem 3 - X1
38 %  To store the number of constraints required to converge for each trial
39 nb_cstr_1 = zeros(N,1);
40
41 % Store the 'final' recovered X (i.e. within 'tol' from X1)
42 X_final_1 = {};
43
44 % Nuclear norm of the final X (to be compared with nuclear norm of X1)
45 nnorm_final_1 = zeros(N,1);
46
47 % Run N trials
48 for i = 1:N
49     [Xs, ids, fnorms] = run_SDPs(X1, tol);
50     X_final_1{i} = Xs{end};
51     nnorm_final_1(i) = fnorms(end);
52     nb_cstr_1(i) = length(Xs);
53 end
54
55 % Average number of constraints required to recover X1 within tolerance
56 avg1 = sum(nb_cstr_1)/N
57
58 % Plot number of iterations
59 figure
60 plot(nb_cstr_1, 'b-o', 'LineWidth', 2);
61 hold on
62 plot(avg1*ones(N,1), 'r-.', 'LineWidth', 2);
```

```
grid on
title('Average number of constraints required to recover X1 through nuclear norm ...
    minimization',...
    'FontSize', 14);
xlabel('Trials');
ylabel('Number of constraints');
legend('Measured', ...
       'Average', ...
       'FontSize', 14, 'Location', 'NorthEast');

%% Problem 3 - X2
%  To store the number of constraints required to converge for each trial
nb_cstr_2 = zeros(N,1);

% Store the 'final' recovered X (i.e. within 'tol' from X1)
X_final_2 = {};

% Nuclear norm of the final X (to be compared with nuclear norm of X1)
nnorm_final_2 = zeros(N,1);

% Run N trials
for i = 1:N
    [Xs, ids, fnorms] = run_SDPs(X2, tol);
    X_final_2{i} = Xs{end};
    nnorm_final_2(i) = fnorms(end);
    nb_cstr_2(i) = length(Xs);
end

% Average number of constraints required to recover X1 within tolerance
avg2 = sum(nb_cstr_2)/N

% Plot number of iterations
figure
plot(nb_cstr_2, 'b-o', 'LineWidth', 2);
hold on
plot(avg2*ones(N,1), 'r-.', 'LineWidth', 2);
grid on
title('Average number of constraints required to recover X2 through nuclear norm ...
    minimization',...
    'FontSize', 14);
xlabel('Trials');
ylabel('Number of constraints');
legend('Measured', ...
       'Average', ...
       'FontSize', 14, 'Location', 'NorthEast');

%% Problem 3 - X3
%  To store the number of constraints required to converge for each trial
nb_cstr_3 = zeros(N,1);

% Store the 'final' recovered X (i.e. within 'tol' from X1)
X_final_3 = {};

% Nuclear norm of the final X (to be compared with nuclear norm of X1)
nnorm_final_2 = zeros(N,1);

% Run N trials
for i = 1:N
```

```
    [Xs, ids, fnorms] = run_SDPs(X3, tol);
    X_final_3{i} = Xs{end};
    nnorm_final_3(i) = fnorms(end);
    nb_cstr_3(i) = length(Xs);
end

% Average number of constraints required to recover X1 within tolerance
avg3 = sum(nb_cstr_3)/N

% Plot number of iterations
figure
plot(nb_cstr_3, 'b-o', 'LineWidth', 2);
hold on
plot(avg3*ones(N,1), 'r-.', 'LineWidth', 2);
grid on
title('Average number of constraints required to recover X3 through nuclear norm ...
    minimization',...
    'FontSize', 14);
xlabel('Trials');
ylabel('Number of constraints');
legend('Measured', ...
       'Average', ...
       'FontSize', 14, 'Location', 'NorthEast');


%% Problem 4
% What follows is using the same function 'run_SDPs' for random matrices of
% specified ranks (3 and 10) and square size q

%% Generate random matrix of size q and rank 3 (X_low)
% Matrix size
q = 15;

% Matrix desired rank
rnk = 1;

% Random matrix of rank 3
% Note that rank can be checked by counting the non-zero entries of svd(X)
tmp = randn(q, rnk);

% Make it square
% Rank(X) = Rank(tmp)
X_low = tmp*tmp';

% Run SDP sequence
[Xs, ids, fnorms] = run_SDPs(X_low, tol);

% Plot error in Froebenius norm
figure
semilogy(fnorms, 'b-o', 'LineWidth', 2);
grid on
title('Froebenius norm error of the SDP solution vs number of constraints',...
    'FontSize', 14);
xlabel('Number of specified entries');
ylabel('Froebenius norm');
legend('||X-X^{*}||', ...
       'FontSize', 14, 'Location', 'NorthEast');

```

```
%% Generate random square matrix of rank 10 (X_high)
% Matrix size
q = 15;

% Matrix desired rank
rnk = 10;

% Random matrix of rank 3
% Note that rank can be checked by counting the non-zero entries of svd(X)
tmp = randn(q, rnk);

% Make it square
% Rank(X) = Rank(tmp)
X_high = tmp*tmp';

% Run SDP sequence
[Xs, ids, fnorms] = run_SDPs(X_high, tol);

% Plot error in Froebenius norm
figure
semilogy(fnorms, 'b-o', 'LineWidth', 2);
grid on
title('Froebenius norm error of the SDP solution vs number of constraints',...
    'FontSize', 14);
xlabel('Number of specified entries');
ylabel('Froebenius norm');
legend('||X-X^{*}||', ...
       'FontSize', 14, 'Location', 'NorthEast');
```

```
function [Xs, ids, fnorms] = run_SDPs(X_data, tol)
% Using CVX
% This function runs nuclear norm minimization SDPs adding information from
% X_data by adding 1 equality constraint on X. We stop the iterations when
% X has become close enough to X_data (within 'tol' in froeb-norm).

% INPUT : X_data : data set
%         tol    : tolerance (stopping condition)

% OUTPUT Xs     : sequence of solutions to SDP
%        ids    : pairs of indices (i,j)
%        fnorms : froebenius norm errors of X's

% Get size
[m,n] = size(X_data);

% To store optimal X's
Xs = {};

% To store froebenius norms error of X's
fnorms = [];

% Generate random indices (i,j) from [1,m]*[1,m] without repetition
ids = [];
while size(ids,1) ~= n*m
  ids = unique([ids ; ceil(m*rand), ceil(n*rand)], 'rows');
end
```

```
28
29 % Shuffle indices
30 ids = ids(randperm(size(ids,1)),:);
31
32 % Nb of constraints = nb of iterations
33 nb_iter = 1;
34
35 % While there are still constraints to add
36 while nb_iter≤size(ids,1)
37
38     % Solve SDP
39     cvx_begin sdp
40       variable W1(m,m) symmetric
41       variable W2(n,n) symmetric
42       variable X(m,n)
43       minimize ( .5*(trace(W1)+trace(W2)) )
44       [W1, X; X', W2] ≥ 0;
45       % Constraints
46       for k=1:nb_iter
47         i = ids(k,1);
48         j = ids(k,2);
49         X(i,j) == X_data(i,j);
50       end
51     cvx_end
52
53     % Record X and row/col ids
54     Xs{end+1} = full(X);
55
56     % Record froebenius norm error
57     fnorms = [fnorms, norm(full(X) - X_data, 'fro')];
58
59     % If X is close enough to X_data we stop
60     % e.g. use froeb norm
61     if norm(full(X) - X_data, 'fro') ≤ tol
62         break
63     end
64
65     % Update counter
66     nb_iter = nb_iter + 1;
67 end
68
69 end
```

C MATLAB code for Problem 5

```
1 %% %% Homework 9 - Problemd 5
2
3 % This time we perform nuclear norm minimization using ADMM
4 % The main algorithm in implemented in 'admm_sdp.m'
5
6 %% Run sequence of SDPs solved through ADMM
7
8 % Generate matrix of desired rank & size
9 % Matrix desired rank
```

```
rnk = 1;

% Matrix desired size
q = 10;

% Note that rank can be checked by counting the non-zero entries of svd(X)
tmp = randn(q, rnk);

% Make it square
% Rank(X_data) = Rank(tmp)
X_data = tmp*tmp';

% Tolerance on recovery error (Froeb norm)
tol = 1e-5;

% Run sequence of SDPs solved through ADMM
[Xs, ids, fnorms] = run_SDPs_admm(X_data, tol);

% Plot nicely
figure
semilogy(fnorms, 'b-o', 'LineWidth', 2);
grid on
title('Froebenius norm error of the SDP solution vs number of constraints',...
    'FontSize', 14);
xlabel('Number of specified entries');
ylabel('Froebenius norm');
legend('||X-X^{*}||', ...
       'FontSize', 14, 'Location', 'NorthEast');
```

```
function [Xs, ids, fnorms] = run_SDPs_admm(X_data, tol)
% Using ADMM
% This function runs nuclear norm minimization SDPs adding information from
% X_data by adding 1 equality constraint on X. We stop the iterations when
% X has become close enough to X_data (within 'tol' in froeb-norm).

% INPUT : X_data : data set
%         tol    : tolerance (stopping condition)

% OUTPUT Xs     : sequence of solutions to SDP
%        ids    : pairs of indices (i,j)
%        fnorms : froebenius norm errors of X's

% Get size
[m,n] = size(X_data);

% Initialize optim variables for SDP (block matrices)
W1 = zeros(m,m);
W2 = zeros(n,n);
x0 = [W1 X_data; X_data' W2];

% Initialize z and u
z0 = eye(m+n);
u0 = eye(m+n);

% ADMM parameters
rho = 1;
```

```
tol_admm = 1e-4;
maxit = 100;

% Objective function under ADMM form
fun = @(x, z)sdp_admm_fun(x, z, 0);

% To store optimal X's of successive SDPs
Xs = {};

% To store froebenius norms error of X's
fnorms = [];

% Generate random indices (i,j) from [1,m]*[1,m] without repetition
ids = [];
while size(ids,1) ~= n*m
  ids = unique([ids ; ceil(m*rand), ceil(n*rand)], 'rows');
end

% Shuffle indices
ids = ids(randperm(size(ids,1)),:);

% Nb of constraints = nb of iterations
nb_iter = 1;

% ids = [1 1]
% While there are still constraints to add
while nb_iter<=size(ids,1)

    % Select a subset of specified entries to send to the current SDP
    sub_ids = ids(1:nb_iter,:);

    [~,~, x_all,~,~,r_all,s_all] = admm_sdp(fun, sub_ids, X_data, rho, x0, z0, u0, ...
        tol_admm, maxit);
    length(x_all)
%     r_all(end)
%     s_all(end)
    % Retrieve optimal X
    X = x_all{end}(1:m,m+1:end);

    % Record optimal X in the list of SDP solutions
    Xs{end+1} = full(X);

    % Record froebenius norm error
    norm(full(X) - X_data, 'fro')
    fnorms = [fnorms, norm(full(X) - X_data, 'fro')];

    % If X is close enough to X_data we stop
    % e.g. use froeb norm
    if norm(full(X) - X_data, 'fro') <= tol
        break
    end

    % Update counter
    nb_iter = nb_iter + 1;
end

end
```

```
function [f_all, g_all, x_all, z_all, u_all, r_all, s_all] = admm_sdp(fun, ids, ...
    X_data, rho, x0, z0, u0, tol, maxit)
% ADMM for SDP
% INPUT : fun      : objective function
%         rho      : quadratic penalization term
%         x0,z0,u0 : initial primal and dual iterates
%         tol      : absolute tolerance on residuals
%         maxit    : maximum number of iterations

% OUTPUT : f_all, g_all : objective function in ADMM form
%          x_all, z_all : primal variables
%          u_all        : dual variable
%          r_all        : primal residuals (2-norm)
%          s_all        : dual residuals (2-norm)

% To store values of objective (x,z,u) and residual r
f_all = [];
g_all = [];
x_all = {};
z_all = {};
u_all = {};
r_all = [];

% Initialize current xk and yk at the starting point x0
xk = x0;
zk = z0;
uk = u0;

% Dimension
[m, n]= size(X_data);
N = m+n;

% Useful identity matrices
IN = eye(N);
IN2 = eye(N^2);

% Nb of constraints
p = size(ids,1);

% Big H matrix for big KKT system during x ADMM update
H = zeros(N^2 + p);

% Upper left block
H(1:N^2,1:N^2) = rho*IN2;
for k = 1:p
    % Extract the entry location (i,j) of the k^th constraint
    i = ids(k,1);
    j = ids(k,2);

    % A_k is defined as zero everywhere except at specified entry (i,j)
    A_k = zeros(m,n);
    A_k(i,j) = 1;
    % Construct block matrix A_tilde_k encoding the k^th equality constraint on x
    Atilde_k = 0.5*[zeros(m,m), A_k; A_k', zeros(n,n)];

    % Fill upper right and bottom left block of big KKT matrix H
    H(1:N^2,N^2+k) = vec(Atilde_k);
```

```
    H(N^2+k,1:N^2) = vec(Atilde_k)';
end

% Relative tolerance
eps_rel = 1e-4;

% Main loop of the gradient descent
k=1;
while (k <= maxit)

    % Record objective value
    [fk, gk] = fun(xk, zk);
    f_all(k) = fk;
    g_all(k) = gk;

    % Primal iterates
    x_all{end+1} = xk;
    z_all{end+1} = zk;

    % Dual iterate
    u_all{end+1} = uk;

    % Primal residual
    rk = xk - zk;
    r_all(k) = norm(rk, 2);

    % RHS vector of KKT system
    h = zeros(N^2+p);
        % Dual part
    h(1:N^2) = vec(zk - uk -(1/rho)*IN);
        % Primal part (vector "b" with specified entries m_ij's)
    for kp=1:p
        i = ids(kp,1);
        j = ids(kp,2);
        h(N^2+kp) = X_data(i,j);
    end

    % Directly \
    tmp = real(H\h); % Returns x and v
    xkp1 = reshape(tmp(1:N^2), [N,N]);
    zkp1 = project_cone_psd(xkp1 + uk);
    ukp1 = uk + xkp1 - zkp1;

    % Dual residual
    sk = -rho*eye(N)*(zkp1 - zk);
    s_all(k) = norm(sk, 2);

    % Stopping criterion
    eps_pri = sqrt(N)*tol + eps_rel*max(norm(xk,2), norm(zk,2));
    eps_dua = sqrt(N)*tol + eps_rel*norm(rho*uk, 2);
    if all((r_all(k) <= eps_pri)) && all((s_all(k) <= eps_dua))
        break
    else
        % Update iterates
        xk = xkp1;
        zk = zkp1;
        uk = ukp1;
        k = k+1;
```

```
    end
end
end
```

```
function [f, g] = sdp_admm_fun(x, z, use_sparse)
% Objective function under ADMM form

% INPUT :
%         x,z        : point at which to evaluate
%         use_sparse : [boolean] (1) use sparse matrices or (0) not

% OUTPUT : f      : value of the objective for x
%          g      : value of the objective for z

% g(z) objective : indicator function of Sn+ for z
g = all(eig(z) ≥ 0) && issymmetric(z);

% f(x) objective : trace of x
f = trace(x);
end
```