



**DEPT. OF CEN**  
**AMRITA SCHOOL OF ENGINEERING**  
**II-B.TECH**  
**CSE-AI**

**SEMISTER-4**  
**21MAT212**  
**MATHEMATICS FOR INTELLIGENT SYSTEMS - 4**  
**END SEMISTER PROJECT**  
**DATE: 05-07-2022**

**Matrix Factorization**

Submitted by:

TEAM : 11

**DIVITH PHOGAT (CB.EN.U4AIE20013)**  
**KARNATI SAI PRASHANTH (CB.EN.U4AIE20027)**  
**LOGESH KSR (CB.EN.U4AIE20032)**  
**MADHAV KISHOR (CB.EN.U4AIE20033)**  
**SVS DHANUSH (CB.EN.U4AIE20068)**

Under the supervision of  
Prof. Dr. Sowmya V.

## ABSTRACT

**Keywords** - Matrix Factorization ,low rank matrix , Matrix Completion

In this paper, the ratings of the previous rated movies( $i$ ) by the users( $j$ ) is used to predict the new ratings for ( $i,j$ ) which are not seen before using the Matrix Factorization. Matrix factorization is a way to generate latent features when multiplying two different kinds of entities. Here we assume the matrices to be a low rank matrices. Collaborative filtering is the application of matrix factorization to identify the relationship between items and users' entities. With the input of users' ratings on the items, we would like to predict how the users would rate the items so the users can get the recommendation based on the prediction.

Since not every user gives ratings to all the movies, there are many missing values in the matrix and it results in a sparse matrix. Hence, the null values not given by the users would be filled with 0 such that the filled values are provided for the multiplication. For example, two users give high ratings to a certain movie when the movie is acted by their favorite actor and actress or the movie genre is an action one, etc.

# Contents

<b>Abstract</b>	<b>2</b>
<b>CHAPTER 1: Matrix Factorization</b>	<b>2</b>
1.1 Overview	2
<b>CHAPTER 2: Low-Rank Models</b>	<b>3</b>
2.1 Rank-1 Model	3
2.2 Rank-r model	8
<b>CHAPTER 3: Matrix Completion</b>	<b>10</b>
3.1 Missing data	10
3.2 Nuclear Norm Minimizing	13
3.2.1 Convexity Test	14
3.3 Implementation In Matlab	15
3.4 Algorithms	19
3.5 Implementation of algorithm in Matlab	19
3.6 Alternating minimization	21
<b>CHAPTER 4: Structured low-rank models</b>	<b>22</b>
4.1 Non negative matrix factorization	22
<b>CHAPTER 5: Conclusion</b>	<b>23</b>
5.1 Appendix	23

# Chapter 1

## Matrix Factorization

### 1.1 Overview

Matrix factorization techniques are commonly considered when dealing with recommendation systems. The scenario corresponds to a group of  $m$  entities mapping to a group of  $n$  other entities. A very common example is the movie recommendation for users. In this case the task is to determine proximity and distance between a movie and a user as if both are same type of entities. In order to obtain such a representation where movies and users can be compared and matched, they both need to be cast as vectors of identical dimension.

In general, the basic ideology is that we will be able to produce the original matrix by multiplying all the resulting ‘factor’ matrices together. Matrix Factorization helps to perform more complex operations on decomposed matrix rather than on the original matrix itself. It helps in the imputation of missing/incomplete data, image segmentation and noise removal, and compression.

There are many different matrix decomposition techniques, each finds use among a particular class of problems.

# Chapter 2

## Low-Rank Models

It is a general, parallelized optimization algorithm that applies to a variety of loss and regularization functions. LRM is useful for reconstructing the missing values and identifying important features in heterogeneous data. Given large collections of data with numeric and categorical values, entries in the table may be noisy or even missing altogether. Low-rank models facilitate the understanding of tabular data by producing a condensed vector representation for every row and column in the dataset.

### 2.1 Rank-1 Model

Consider the problem of modeling a quantity  $y[i;j]$  that depends on two indices  $i$  and  $j$ . To ideas, assume that  $y[i; j]$  represents the rating assigned to a movie  $i$  by a user  $j$ . If we have available a data set of such ratings, how can we predict new ratings for  $(i; j)$  that we have not seen before? A possible assumption is that some movies are more popular in general than others, and some users are more generous. This is captured by the following simple model

$$y[i, j] \approx a[i] b[j] \quad (2.1)$$

$a$  - Users [Some users might be generous]

$b$  - Movies[Some movies might be popular]

here  $a, b$  are the features

MODEL: Combing the features multiplicatively

The features  $a$  and  $b$  capture the respective contributions of the movie and the user to the ranking. If  $a[i]$  is large, movie  $[i]$  receives good ratings in general. If  $b[j]$  is large, then user  $[j]$  is generous, they give good ratings in general.

In the model (1) the two unknown parameters  $a[i]$  and  $b[j]$  are combined multiplicatively. As a result, if we fit these parameters using observed data by minimizing a least-squares cost function, the optimization problem is not convex. Figure 2.1 illustrates this for the function.

$$f(a,b) := (1 - ab)^2 \quad (2.2)$$

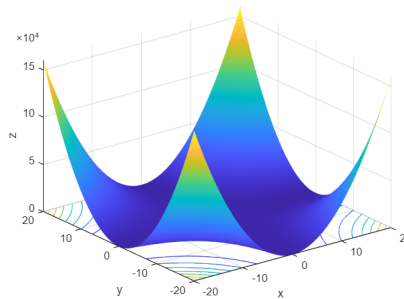


Figure 2.1: Plot the above example

which corresponds to an example where there is only one movie and one user and the rating equals one. **Nonconvexity is problematic** because if we use algorithms such as gradient descent to minimize the cost function they may get stuck in local minima corresponding to parameter values that do not fit the data well.

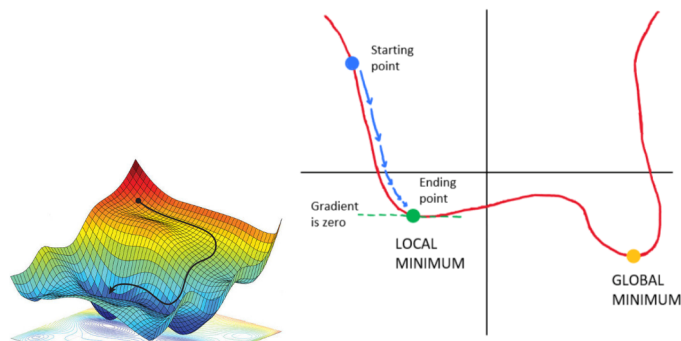


Figure 2.2: Non Convex Data

In addition, there is a scaling issue: the pairs (a; b) and (ac; b=c) yield the same cost for any constant c. This motivates incorporating a constraint to restrict the magnitude of some of the parameters. For example, in Figure 2.1 we add the constraint  $|a| = 1$ , which is a nonconvex constraint set.

Assume that there are m movies and n users in the data set and every user rates every movie. If we store the ratings in a matrix Y such that  $Y_{ij} := y[i, j]$  and the movie and user features in the vectors  $\vec{a} \in R^m$  and  $\vec{b} \in R^n$ , then equation (2.1) is equivalent to

$$Y \approx \vec{a} \vec{b}^T \quad (2.3)$$

Note that  $\vec{a} \vec{b}^T$  is a rank-1 matrix and conversely any rank-1 matrix can be written in this form where  $\|\vec{a}\|_2 = 1$  ( $\vec{a}$  is equal to any of the columns normalized by their  $l_2$  norm). The problem is consequently equivalent to)

$$\min_{X \in R^{m \times n}} \|Y - X\|_F \text{ subject to } \text{rank}(X) = 1 \quad (2.4)$$











the solution  $X_{min}$  to this optimization problem is given by the truncated singular-value decomposition (SVD) of Y

$$X_{min} = \sigma_1 \vec{u}_1 \vec{v}_1^T \quad (2.5)$$

where  $\sigma_1$  is the largest singular value and  $\vec{u}_1$  and  $\vec{v}_1$  are the corresponding singular vectors. The corresponding solutions  $\vec{a}_{min}$  and  $\vec{b}_{min}$  are

**Example 1.1 (Movies)** . Madhav, Svs, Prash and Loki rate the following six movies from 1 to 5.

**A** =

	 MADHAV	 SVS	 PRASH	 LOKI	
	1	1	5	4	VIKRAM 
	2	1	4	5	VADA CHENNAI 
	4	5	2	1	THOR : LOVE AND THUNDER 
	5	4	2	1	CHARLIE 777 
	4	5	1	2	DOCTOR STRANGE 
	1	2	5	5	BEAST 

To fit a low-rank model, we first subtract the average of rating matrix from each entry in the matrix to obtain a centered matrix  $C$  and then compute its singular-value decomposition,

$$\mu = \frac{1}{mn} \sum_{i=1}^m \sum_{j=1}^n A_{ij} \quad (2.6)$$

```
mu = (1/(m*n))*sum(sum(A))
```

```
mu = 3
```

we will be doing all our computations using MATLAB, The value of  $\mu$  is founded as 3.

To compute the SVD of matrix we use inbuild command *svd* as shown below



```
[U , S ,V] = svd(A - mu*ones(s))
```

```
U = 6×6
```

```
-0.4464    0.3717    0.4202   -0.6015    0.1937    0.2943
-0.3905   -0.0000   -0.5625    0.0000    0.6875   -0.2415
 0.3905    0.0000    0.5625   -0.0000    0.4939   -0.5358
 0.3850    0.6015   -0.0834    0.3717    0.3228    0.4906
 0.3850   -0.6015   -0.0834   -0.3717    0.3228    0.4906
-0.4464   -0.3717    0.4202    0.6015    0.1937    0.2943
```

```
S = 6×4
```

```
 7.7851         0         0         0
         0    1.6180         0         0
         0         0    1.5468         0
         0         0         0    0.6180
         0         0         0         0
         0         0         0         0
```

```
V = 4×4
```

```
 0.4780    0.3717   -0.5210    0.6015
 0.5210   -0.6015    0.4780    0.3717
-0.4780    0.3717    0.5210    0.6015
-0.5210   -0.6015   -0.4780    0.3717
```

The fact that the first singular value is significantly larger than the rest suggests that the matrix may be well approximated by a rank-1 matrix. With the first singular value of S itself we will try to reconstruct the matrix

```
reconstructingA = mu*ones(s) + S(1,1)*(U(:,1))*(V(:,1)')
```

```
reconstructingA = 6×4
```

```
 1.3387    1.1893    4.6613    4.8107
 1.5466    1.4160    4.4534    4.5840
 4.4534    4.5840    1.5466    1.4160
 4.4328    4.5616    1.5672    1.4384
 4.4328    4.5616    1.5672    1.4384
 1.3387    1.1893    4.6613    4.8107
```

The first left singular vector is equal to,

```
u1 = U(:,1)
```

```
u1 = 6×1
```

```
-0.4464
-0.3905
 0.3905
 0.3850
 0.3850
-0.4464
```

This vector allows us to cluster the movies: movies with negative entries are similar and movies with positive entries are similar. The first right singular vector is equal to

$$v1 = V(:,1)$$

$$v1 = 4 \times 1$$

$$\begin{matrix} 0.4780 \\ 0.5210 \\ -0.4780 \\ -0.5210 \end{matrix}$$

This vector allows to cluster the users: negative entries indicate users that like action movies but hate romantic movies, whereas positive entries indicate the contrary.

## 2.2 Rank-r model

Our rank-1 matrix model is extremely simplistic. Different people like different movies. In order to generalize it we can consider  $r$  factors that capture the dependence between the ratings and the movie/user

$$y[i, j] \approx \sum_{l=1}^r a_l[i] b_l[j] \quad (2.7)$$

The parameters of the model have the following interpretation:

1.  $a_l[i]$  : movie  $i$  is positively ( $> 0$ ), negatively ( $< 0$ ) or not ( $\approx 0$ ) associated to factor  $l$ .
2.  $b_l[j]$  : movie  $j$  is positively ( $> 0$ ), negatively ( $< 0$ ) or not ( $\approx 0$ ) associated to factor  $l$ .

The model learns the factors directly from the data. In some cases, these factors may be interpretable for example, they can be associated to the genre of the movie as in Example 1.1 or the age of the user but this is not always the case.

The model corresponds to a rank- $r$  model

$$Y \approx AB, \quad A \in R^{m \times r}, \quad B \in R^{r \times n} \quad (2.8)$$

We can fit such a model by computing the SVD of the data and truncating it. The truncating SVD is the solution to,


$$\min_{A \in R^m, B \in R^{r \times n}} \|Y - AB\|_F \text{ subject to } \|\vec{a}_1\|_2 = 1, \dots, \|\vec{a}_r\|_2 = 1 \quad (2.9)$$

```
% We don't have a right dataset for rank r model
% so we generate a random low rank matrix and implement
% our rank - r model .
% We added a slider so the rank can be set based on the data .

A = rand(2000,200)*rand(200,2000)*10 ;

% RANK - r MODEL
s = size(A);
m = s(1);
n = s(2);
mu = 0 %(1/(m*n))*sum(sum(A)) ;

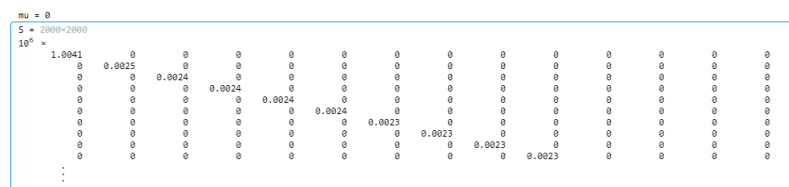
% RANK 1 MODEL
ones(s);
[U , S ,Vt] = svd(A - mu*ones(s)); % calculatling svd of centred data
S

V=Vt';
n = 1  % setting the n to abstract the features

%The features that we get form out matrix .
u_n = U(:,1:n); %a - movies / might be unintrepretable
v_n = V(1:n,:); %b - user likes / might be unintrepretable
App_Rr = mu*ones(s) + S(1,1)*(u_n)*(v_n); % reconstructed from features|

norm(App_Rr-A) % We see the norm to be small only for rank = 1 , means the random is not perfect
```

we can see the value of s which is the singular values , after n values we can see only null values will be entered in.



However, the SVD provides an estimate of the matrices A and B that constrains the columns of A and the rows of B to be orthogonal. As a result, these vectors do not necessarily correspond to interpretable factors.

# Chapter 3

## Matrix Completion

Matrix Completion is a method for recovering lost information. It originates from machine learning and usually deals with highly sparse matrices. Missing or unknown data is estimated using the low-rank matrix of the known data.

### 3.1 Missing data

The problem of predicting ratings can be recast as that of completing a matrix from some of its entries as illustrated in Figure. This problem is known as matrix completion.

The problem looks similar to this :

$$\begin{bmatrix} ? & 1 & 5 \\ 1 & 7 & ? \\ ? & 1 & 5 \end{bmatrix}$$

Figure 3.1: Missing data

The below figure depicts the movie matrix with missing rating :

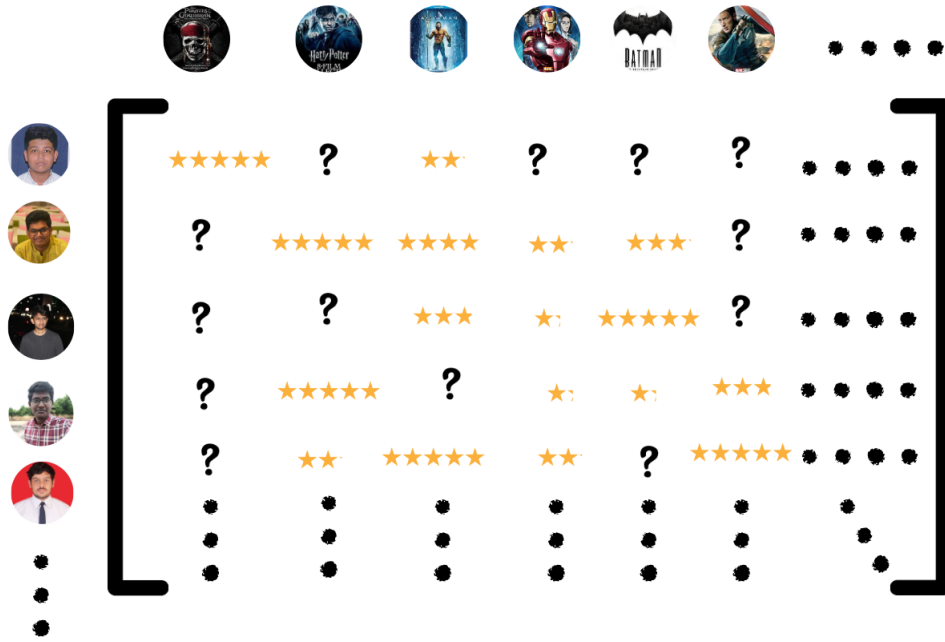


Figure 3.2: Each row corresponds to a user that ranks a subset of the movies, which correspond to the columns

In order to solve the problem, we need to make an assumption on the structure of the matrix that we aim to complete. In compressed sensing we make the assumption that the original signal is sparse. In the case of matrix completion, we make the assumption that the original matrix is low rank. This implies that there exists a high correlation between the entries of the matrix, which may make it possible to infer the missing entries from the observations.

The low-rank assumption implies that if the matrix has dimensions  $m \times n$  then it can be factorized into two matrices that have dimensions  $m \times r$  and  $r \times n$ . This factorization allows to encode the matrix using  $r(m + n)$  parameters. If the number of observed entries is larger than  $r(m + n)$  parameters then it may be possible to recover the missing entries. However, this is not enough to ensure that the problem is well posed.

In the following matrix the last row does not seem to be correlated to the rest of the rows,

$$\begin{aligned}
 &M = 5 \times 4 \\
 &\begin{matrix} 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \\ -3 & 3 & -3 & 3 \end{matrix}
 \end{aligned}$$

The singular-value decomposition of the matrix M is given by,

```
[u,s,vt] = svd(M);
v=vt';
```

The first rank-1 component of this decomposition has information that is very spread out,

```
S1 = s(1,1)*(u1)*(v1)
```

```
S1 = 5x4
      2.0000      2.0000      2.0000      2.0000
      2.0000      2.0000      2.0000      2.0000
      2.0000      2.0000      2.0000      2.0000
      2.0000      2.0000      2.0000      2.0000
      0.0000      0.0000      0.0000      0.0000
```

It is because the entries in  $V_1$  are non - zero and have same magnitude, so that each entry of  $U_1$  affects every single entry of the corresponding row. If any one of the entries is missing in the data we can recover it from the other entries

```
v1 = v(1,:)
```

```
v1 = 1x4
     -0.5000     -0.5000     -0.5000     -0.5000
```

```
u1 = u(:,1)
```

```
u1 = 5x1
     -0.5000
     -0.5000
     -0.5000
     -0.5000
     -0.0000
```

But the information in the second rank-1 component is very localized, due to the fact that the corresponding left singular vector is very sparse,

```
S2 = s(2,2)*(u2)*(v2)
```

```
S2 = 5x4
      0.0000     -0.0000      0.0000     -0.0000
     -0.0000      0.0000     -0.0000      0.0000
      0.0000     -0.0000      0.0000     -0.0000
      0.0000     -0.0000      0.0000     -0.0000
     -3.0000      3.0000     -3.0000      3.0000
```

Reconstructing the matrix completely,

```
S1+S2 % RECONSTRUCT THE MATRIX
```

```
ans = 5x4
      2.0000      2.0000      2.0000      2.0000
      2.0000      2.0000      2.0000      2.0000
      2.0000      2.0000      2.0000      2.0000
      2.0000      2.0000      2.0000      2.0000
     -3.0000      3.0000     -3.0000      3.0000
```

If we observe this carefully we can understand that each entry in the right singular vector affects only one entry of the component. This simple example shows that sparse singular vectors are problematic for matrix completion.

## 3.2 Nuclear Norm Minimizing

The nuclear norm minimization (NNM) is commonly used to approximate the matrix rank by shrinking all singular values equally. However, the singular values have clear physical meanings in many practical problems, and NNM may not be able to faithfully approximate the matrix rank.

We are interested in recovering low-rank matrices from a subset of their entries. Let  $\Omega$  be the related elements and let  $\vec{y}$  be a vector containing the revealed entries. Ideally, we would like to choose the measurement-corresponding matrix with the lowest rank.

$$\min_{X \in \mathbb{R}^{m \times n}} \text{rank}(X) \quad \text{such that} \quad X_{\Omega} = \vec{y} \quad (3.1)$$

this optimization problem is computationally hard to solve. Substituting the rank with the nuclear norm yields a tractable alternative:

$$\min_{X \in \mathbb{R}^{m \times n}} \|X\|_* \quad \text{such that} \quad X_{\Omega} = \vec{y} \quad (3.2)$$

The cost function is convex and the constraint is linear, so this is a convex program. In practice, the revealed entries are usually noisy. They do not correspond exactly to entries from a low-rank matrix. We take this into account by removing the equality constraint and adding a data-fidelity term penalizing the '2-norm error over the revealed entries in the cost function

$$\min_{X \in \mathbb{R}^{m \times n}} \frac{1}{2} \|X_{\Omega} - \vec{y}\|_2^2 + \lambda \|X\|_* \quad (3.3)$$

where  $\lambda > 0$  is a regularization parameter.

### 3.2.1 Convexity Test

we say function  $f$  is convex on a given set  $S$  only if :

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y) \quad (3.4)$$

is strictly satisfied for all  $x, y \in S$  and all  $\lambda \in [0, 1]$

#### Testing The convexity of Rank()

here we are testing weather the function  $\min_{X \in R^{m \times n}} \text{rank}(X)$  is a convex or an non-convex function with an example.

$$\begin{aligned} X &= \begin{bmatrix} 5 & 0 \\ 0 & 0 \end{bmatrix} \quad Y = \begin{bmatrix} 0 & 0 \\ 0 & 5 \end{bmatrix} \quad \lambda = 0.1 \\ \text{rank}\left(\begin{bmatrix} 0.5 & 0 \\ 0 & 4.5 \end{bmatrix}\right) &\not\leq 0.1 * \text{rank}\left(\begin{bmatrix} 5 & 0 \\ 0 & 0 \end{bmatrix}\right) + 0.9 * \text{rank}\left(\begin{bmatrix} 0 & 0 \\ 0 & 5 \end{bmatrix}\right) \\ 2 &\not\leq 0.1 * 1 + 0.9 * 1 \end{aligned}$$

we can clearly see that the  $\min_{X \in R^{m \times n}} \text{rank}(X)$  is a non-convex function cause it is not satisfying the above equation 3.4

#### Testing The convexity of Nuclear Norm

Now we are calculating weather the Nuclear Norm function  $\min_{X \in R^{m \times n}} \|X\|_*$  is convex or an Non-convex function.

$$\begin{aligned} \text{Let } x &:= \begin{bmatrix} 5 \\ 0 \end{bmatrix} \text{ and } y := \begin{bmatrix} 0 \\ 5 \end{bmatrix} \quad \text{for any } \lambda \in (0, 1) \\ \|\lambda x + (1 - \lambda)y\|_* &\leq \lambda \|x\|_* + (1 - \lambda) \|y\|_* \\ \lambda &= 0.1 \\ \begin{bmatrix} 0.707 \\ 0 \end{bmatrix} &\leq \begin{bmatrix} 5 \\ 0 \end{bmatrix} \end{aligned}$$



```

x= [5 0]'

x = 2x1
    5
    0

y = [0 5]'

y = 2x1
    0
    5

la = 0.1

la = 0.1000

c= la*x+la*y

c = 2x1
    0.5000
    0.5000

x11=norm(svd(c,"vector"))

x11 = 0.7071

x22 = la*norm(svd(x)) + (1-la)*norm(svd(y))

x22 = 5

```

The Nuclear Norm function is satisfying the convexity condition.

### 3.3 Implementation In Matlab

we are converting the constrained optimization problem into an unconstrained optimization problem. Constrained optimization problems are very difficult to solve in the real life where as the unconstrained optimization problem can be solved using iterative process .

We now apply this method to the following completion problem for a matrix X having unknown variables

```

originalMatrix = 6x4
    1    1    5    4
    2    1    4    5
    4    5    2    1
    5    4    2    1
    4    5    1    2
    1    2    5    5

```

Replacing some values with some unknown variables

$$X = \begin{pmatrix} 1 & x_1 & 5 & 4 \\ x_2 & 1 & 4 & 5 \\ 4 & 5 & 2 & x_3 \\ 5 & 4 & 2 & 1 \\ 4 & 5 & 1 & 2 \\ 1 & 2 & x_4 & 5 \end{pmatrix}$$

In order to solve the problem we follow the below steps:

1. We compute the average observed rating and subtract it from each entry in the matrix. We denote the vector of centered ratings by  $y$ .
2. We solve the optimization problem
3. We add the average observed rating to the solution of the optimization problem and round each entry to the nearest integer

We are first solving the initial equation using CVX :

$$\min_{X \in \mathbb{R}^{m \times n}} \|X\|_* \quad \text{such that} \quad X_{\Omega} = \vec{y} \quad (3.5)$$

We replace the unknown variables with zero and we will find the average rating of the matrix

```
mu = vpa(subs(f,[x1 x2 x3 x4],[0 0 0 0])/20)
mu = 3.15
```

We create a variable matrix of dimensions 6\*4 and constraints are the known part of the matrix

```

% CVX baseline
cvx_begin quiet
    variable X(m,n)
    minimize(norm_nuc(X))

    subject to
        X(1,1)==1-mu
        X(1,3)==5-mu
        X(1,4)==4-mu
        X(2,2)==1-mu
        X(2,3)==4-mu
        X(2,4)==5-mu
        X(3,1)==4-mu
        X(3,2)==5-mu
        X(3,3)==2-mu
        X(4,:) == [5,4,2,1]-mu
        X(5,:) == [4,5,1,2]-mu
        X(6,1)==1-mu
        X(6,2)==2-mu
        X(6,4)==5-mu
cvx_end

```

The final matrix is rounded off and we could observe that predicted values are almost equal to the original values

```
round(full(X)+mu)
```

```

ans = 6x4
     1     3     5     4
     2     1     4     5
     4     5     2     2
     5     4     2     1
     4     5     1     2
     1     2     5     5

```

Now we will be solving with the unconstrained equation using CVX :

$$\min_{X \in \mathbb{R}^{m \times n}} \frac{1}{2} \|X_{\Omega} - \vec{y}\|_2^2 + \lambda \|X\|_* \quad (3.6)$$

we are initializing lambda = 0.15 (hyper parameter.

```
%Solving using CVX

lamda = 0.15 % hyper parameter set after trying many

%CVX
cvx_begin quiet
    variable X(6,4)
    y = [1 ; 5 ; 4 ; 1 ; 4 ; 5 ; 4 ; 5 ; 2 ; 5 ; 4 ; 2 ; 1 ; 4 ; 5 ; 1 ; 2 ; 1 ; 2 ; 5] -mu ; % the known part of matrix
    subject to % constrains
        xx = [X(1,1);X(1,3);X(1,4);X(2,2);X(2,3);X(2,4);X(3,1);
              X(3,2);X(3,3);X(4,1);X(4,2);X(4,3);X(4,4);X(5,1);X(5,2);
              X(5,3);X(5,4);X(6,1);X(6,2);X(6,4);]
        minimize( 0.5*norm(xx - y,2) + lamda*norm_nuc(X)) ; % cost function
cvx_end

round(X+mu) % Final matrix is rounded , it almost match the orginal matches
```

we can see the matrix, where the missing entries are same when it is compared to that of the nuclear norm solution above.

```
ans = 6x4

     1     3     5     4
     2     1     4     5
     4     5     2     2
     5     4     2     1
     4     5     1     2
     1     2     5     5
```

we are almost able to match the values of the missing data which are some what much closer to that of the Original values of the user ratings.

### 3.4 Algorithms

As saw how we solve our proximal problem in cvx , lets explore how this is done as iterative solution .

This is the theorem for the proximal equations with minimizing nuclear norm .

**Theorem :** (Proximal operator of the nuclear norm) The solution to

$$\min_{X \in \mathbb{R}^{m \times n}} \frac{1}{2} \| X - Y \|_F^2 + \tau \| X \|_* \text{ is } \mathcal{D}_\tau(Y), \quad (3.7)$$

obtained by soft-thresholding the singular values of  $Y = USV^T$

$$\mathcal{D}_\tau(Y) = US_\tau(S)V^T, \quad (3.8)$$

$$S_\tau(S)_{ii} = S_{ii} - \tau \text{ if } S_{ii} > \tau, \text{ otherwise } 0 \quad (3.9)$$

But this exact theorem is not sufficient as we have a partial complete matrix and vectors instead of matrices . So we arrive at this algorithm for solving .

**Algorithm :** (Proximal-gradient method for nuclear-norm regularization) Let  $Y$  be a matrix such that  $Y_\omega = y$  and let us abuse notation by interpreting  $X_\Omega^{(k)}$  as a matrix which is zero on  $\Omega^c$ . We set the initial point  $X^{(0)}$  to  $Y$ . Then we iterate the update

$$X^{(k+1)} = \mathcal{D}_{\alpha_k \lambda}(X^{(k)} - \alpha_k(X_\Omega^{(k)} - Y)) \quad (3.10)$$

where  $\alpha_k > 0$  is the step size.

### 3.5 Implementation of algorithm in Matlab

This is the function for soft thresholding of singular values based the threshold value .

```
function D = Sft_thresh_singular_vals(Y,tou)
%Soft thresholding of singular values based the given parameter tou
[U,S,Vt] = svd(Y);
[m,n] = size(S);
for i = [1:n-1]
if S(i,i) > tou
    S(i,i) = S(i,i) - tou; % we subtract if > tou
else
    S(i,i) = 0; % else it will be set to 0
end
end
D = U*S*Vt; %Final matrix that we return
end
```

Now as we have the functions for soft threshold we directly implement our algorithm based the equation 3.10 . We also not don't know about the alpha learning rate . So we use 2 loops and a 19\*19 , thats around 361 combinations and choose the best pair hyper parameters

```
% to find an optimal value for the
% hyperparameters alpha and threshold values so we have experimented with
% several combinations of different range .

A = [1 1 5 4; 2 1 4 5; 4 5 2 1; 5 4 2 1; 4 5 1 2; 1 2 5 5] %matrix

A = 6x4
     1     1     5     4
     2     1     4     5
     4     5     2     1
     5     4     2     1
     4     5     1     2
     1     2     5     5

Y = [1 0 5 4; 0 1 4 5; 4 5 2 0; 5 4 2 1; 4 5 1 2; 1 2 0 5] - mu %the matrix missing elements zero

Y = 6x4
    -2.1500    -3.1500     1.8500     0.8500
    -3.1500    -2.1500     0.8500     1.8500
     0.8500     1.8500    -1.1500    -3.1500
     1.8500     0.8500    -1.1500    -2.1500
     0.8500     1.8500    -2.1500    -1.1500
    -2.1500    -1.1500    -3.1500     1.8500

Y(1,2)=0;
Y(2,1)=0;
Y(3,4)=0;
Y(6,3)=0;
X = Y

X = 6x4
    -2.1500         0     1.8500     0.8500
         0    -2.1500     0.8500     1.8500
     0.8500     1.8500    -1.1500         0
     1.8500     0.8500    -1.1500    -2.1500
     0.8500     1.8500    -2.1500    -1.1500
    -2.1500    -1.1500         0     1.8500
```

```
[n1,n2] = size(Y);
small = 99999999 ;% setting small value to compare results

%different combinations of hyper parametres tested
for q=[-9:9]
    for p=[-9:9]

        alpha = 10^(p);
        threshold = 10^(q) ;

        %We do a 100 itrations , as larger ones yielded bad results
        for i =[0:100]
            XK = X;

            XK(1,2)=0;
            XK(2,1)=0;
            XK(3,4)=0;
            XK(6,3)=0;
            X = Sft_thresh_singular_vals( X - alpha*( XK - Y),threshold); % update as per algorithm

        end

        if small > norm(X+mu - A)
            small = norm(X+mu - A);
            s=X;
            val = [10^(p),10^(q)]; % optimal hyperparameter

        end
    end
end

% optimal hyperparameter
val(1) % optimal alpha

ans = 1

val(2) % optimal threshold

ans = 1.0000e-09
```

```

round(s+mu)

ans = 6x4
    1    2    5    4
    2    1    4    5
    4    5    2    2
    5    4    2    1
    4    5    1    2
    1    2    4    5

```

so these are the 2 optimal values for 100 iterations and we see the output almost matches the actual matrix in our reference .

### 3.6 Alternating minimization

Our current method minimizing the nuclear norm to recover a low-rank matrix is an effective method but it requires repeatedly computing the singular-value decomposition of the matrix, which can be computationally heavy for large matrices. So a better way is to factorize the matrix as  $AB$  where  $A \in R^{m \times k}$  and  $B \in R^{k \times n}$  as now we know the rank cant exceed  $k$  .So we can use cross validation and set  $k$  .

$A$  and  $B$  can now be fit by solving this optimization problem

$$\min_{\tilde{A} \in R^{m \times k}, \tilde{B} \in R^{k \times n}} \| (\tilde{A}\tilde{B})_{\Omega} - \vec{y} \|_2 \quad (3.11)$$

This is nonconvex , so we solve by alternate minimization .That is fix  $B$  and solve

$$\min_{\tilde{A} \in R^{m \times k}} \| (\tilde{A}\tilde{B})_{\Omega} - \vec{y} \|_2 \quad (3.12)$$

and fix  $A$  and solve

$$\min_{\tilde{B} \in R^{k \times n}} \| (\tilde{A}\tilde{B})_{\Omega} - \vec{y} \|_2 \quad (3.13)$$

Iteratively solving these least-squares problems allows to find a local minimum of the cost function.

# Chapter 4

## Structured low-rank models

### 4.1 Non negative matrix factorization

PCA can be used to compute the main principal directions of a dataset, which can be interpreted as basis vectors that capture as much of the energy as possible. These vectors are constrained to be orthogonal. Unfortunately, as a result they are often not necessarily interpretable. This suggests computing a decomposition where all elements of A and B are non-negative, with the hope that this will allow us to learn a more interpretable model. A non-negative matrix factorization of the data matrix may be obtained by solving the optimization problem,

$$\text{minimize } \|X - \tilde{A}\tilde{B}\|_F^2 \quad (4.1)$$

$$\text{subject to } \tilde{A}_{i,j} \geq 0 \quad (4.2)$$

$$\tilde{B}_{i,j} \geq 0 \text{ for all } i, j \quad (4.3)$$

This is a nonconvex problem which is computationally hard, due to the non-negative constraint. This can still be solved by using high-end concepts, which makes matrix factorization interpretable.



# Chapter 5

## Conclusion

So in this project we have explored decomposing to important features in rank 1 level and rank -  $r$  level , we also explored completion matrices from cvx to iterative methods computationally , and we concluded with alternative minimization and non negative matrix factorization . This project can be used in applications of analysis of data where instead of discarding null or "nan" valued rows we can fill the missing element with a reasonable approximation .

### 5.1 Appendix

# MIS END SEMISTER PROJCT

## MATRIX FACTORIZATION

TEAM - 11

### Least-Squares Optimization Problem

```
% Model 1 with rating = 1
% Here we will visualize the rank one model with a single case , where
% one user's rating one movie is got by a*b and we take the least
% squares optimization problem .
```

$$f(a, b) := (1 - ab)^2$$

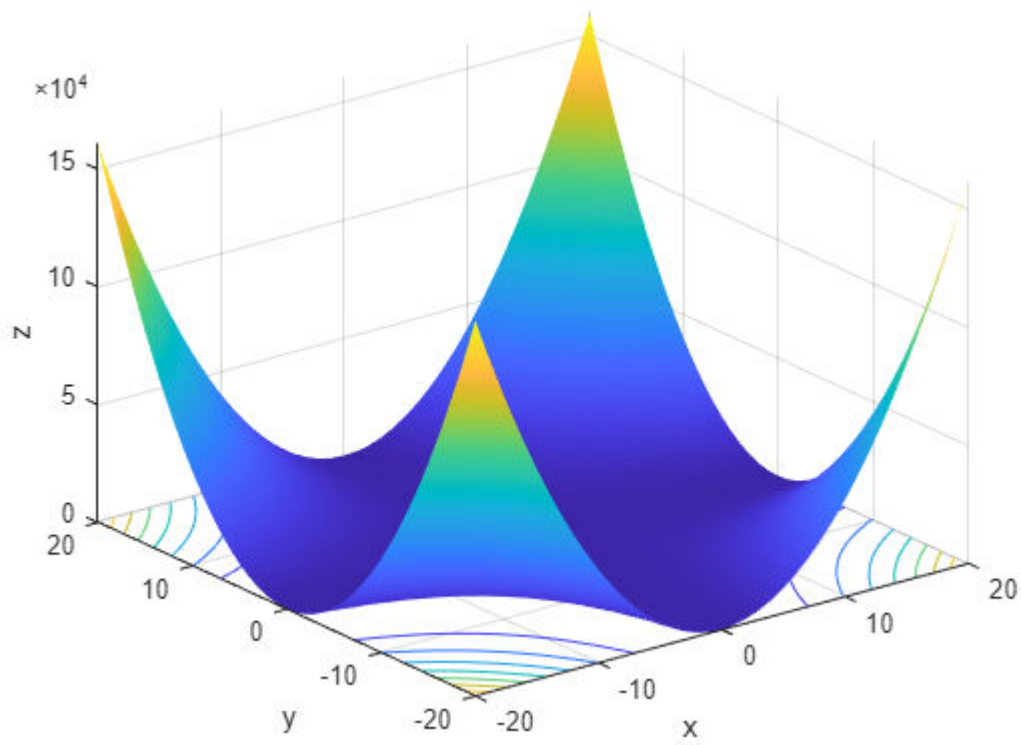
```
x = -20:20 ;
y = x ;
[x1,x2] = meshgrid(x,y) ;

Z = (1 - x1.*x2).^2 % function equation
```

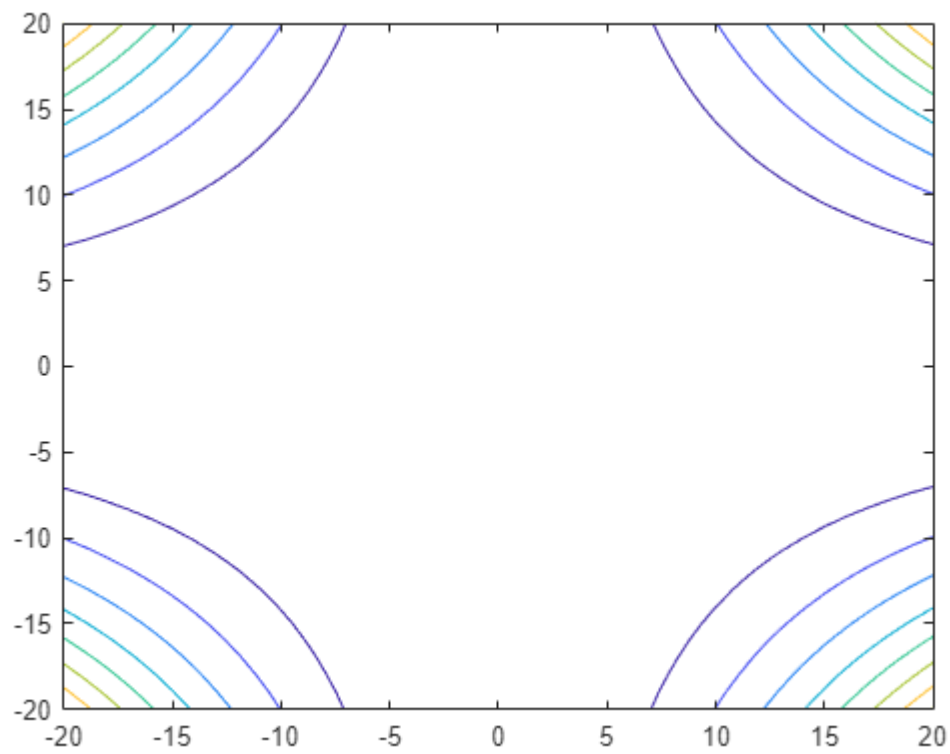
```
Z = 41x41
    159201    143641    128881    114921    101761    89401 ...
    143641    129600    116281    103684    91809    80656
    128881    116281    104329    93025    82369    72361
    114921    103684    93025    82944    73441    64516
    101761    91809    82369    73441    65025    57121
    89401    80656    72361    64516    57121    50176
    77841    70225    63001    56169    49729    43681
    67081    60516    54289    48400    42849    37636
    57121    51529    46225    41209    36481    32041
    47961    43264    38809    34596    30625    26896
    :
    :
```

```
surf(x1,x2,Z) % plotting it
shading interp
```

```
%Labels
xlabel('x')
ylabel('y')
zlabel('z')
```



```
%The contour plot of the function  
contour(x1,x2,Z)
```













## Rank-1 Model

```
% Here we extract the a,b features from the matrix using svd .
```

**Example 1.1 (movies)** MADHAV , SVS , PRASH , LOKI rate the following six movies from 1 to 5.

**A =**

	 MADHAV	 SVS	 PRASH	 LOKI		
	1	1	5	4	VIKRAM	
	2	1	4	5	VADA CHENNAI	
	4	5	2	1	THOR : LOVE AND THUNDER	
	5	4	2	1	CHARLIE 777	
	4	5	1	2	DOCTOR STRANGE	
	1	2	5	5	BEAST	

```
A = [1 1 5 4; 2 1 4 5 ;4 5 2 1 ;5 4 2 1 ;4 5 1 2 ;1 2 5 5] %Rating matrix
```

```
A = 6×4
```

```
1     1     5     4
2     1     4     5
4     5     2     1
5     4     2     1
4     5     1     2
1     2     5     5
```

```
s = size(A);
m = s(1);
n = s(2);
mu = (1/(m*n))*sum(sum(A)); %calculating the mean
```

```
ones(s); % creating a matrix of ones of the same size of matrix to centre the data
[U , S ,Vt] = svd(A - mu*ones(s)); % calculatling svd of centred data
S %Note that the 1st singular value is significantly larger .
```

```
S = 6×4
```

```
7.7851     0     0     0
0     1.6180     0     0
0     0     1.5468     0
0     0     0     0.6180
0     0     0     0
0     0     0     0
```

```
%The features that we get form out matrix .
```

```
u1 = U(:,1) %a - movies genre
```

```
u1 = 6×1
```

```
-0.4464
-0.3905
0.3905
0.3850
0.3850
-0.4464
```

```
v1 = Vt(:,1) %b - users taste/likes
```

```
v1 = 4x1
    0.4780
    0.5210
   -0.4780
   -0.5210
```

```
App_R1 = mu*ones(s) + S(1,1)*(u1)*(v1') % The features are used to reconstruct the original matrix
```

```
App_R1 = 6x4
    1.3387    1.1893    4.6613    4.8107
    1.5466    1.4160    4.4534    4.5840
    4.4534    4.5840    1.5466    1.4160
    4.4328    4.5616    1.5672    1.4384
    4.4328    4.5616    1.5672    1.4384
    1.3387    1.1893    4.6613    4.8107
```

```
norm(App_R1-A) % we see that the matrix regained is almost the same as A
```

```
ans = 1.6180
```

## Rank -R Modle

$$y[i, j] \approx \sum_{l=1}^r a_l[i] b_l[j].$$

$$Y \approx AB, \quad A \in \mathbb{R}^{m \times r}, \quad B \in \mathbb{R}^{r \times n}.$$

```
% We don't have a right dataset for rank r model
% so we generate a random low rank matrix and implement
% our rank - r model .
% We added a slider so the rank can be set based on the data .
```

```
A = rand(2000,200)*rand(200,2000)*10 ;
```

```
% RANK - r MODEL
```

```
s = size(A);
m = s(1);
n = s(2);
mu = 0 %(1/(m*n))*sum(sum(A)) ;
```

```
mu = 0
```

```
% RANK 1 MODEL
```

```
ones(s);
[U , S ,Vt] = svd(A - mu*ones(s)); % calculating svd of centred data
S
```

```
S = 2000x2000
```

```
106 x
    1.0022      0      0      0      0      0      0      0 ...
         0    0.0025      0      0      0      0      0      0
         0      0    0.0024      0      0      0      0      0
         0      0      0    0.0024      0      0      0      0
         0      0      0      0    0.0024      0      0      0
         0      0      0      0      0    0.0023      0      0
         0      0      0      0      0      0    0.0023      0
         0      0      0      0      0      0      0    0.0023
         0      0      0      0      0      0      0      0
         0      0      0      0      0      0      0      0
         :
         :
```

```
V=Vt';
n = 1 % setting the n to abstract the features
```

```
n = 1
```

```
%The features that we get form out matrix .
```

```
u_n = U(:,1:n); %a - movies / might be unintrepretable
```

```
v_n = V(1:n,:); %b - user likes / might be unintrepretable
```

```
App_Rr = mu*ones(s) + S(1,1)*(u_n)*(v_n); % reconstructed from features
```

```
norm(App_Rr-A) % We see the norm to be small only for rank = 1 , means the random is not perfect
```

```
ans = 2.4508e+03
```

## Matrix completion

```
% Matrix completing - Localized data example
```

```
% Last row is localized , hence is problematic for reconstruction
```

```
M = [2 2 2 2;
      2 2 2 2;
      2 2 2 2;
      2 2 2 2;
      -3 3 -3 3; ] %localized matrix
```

```
M = 5x4
```

```
    2    2    2    2
    2    2    2    2
```

```

2    2    2    2
2    2    2    2
-3   3   -3   3

```

```

[u,s,vt] = svd(M); % we take svd to analys
v=vt';

```

```

u1 = u(:,1) %most entries non zero , so each entry affects all elements of corresponding row

```

```

u1 = 5x1
-0.5000
-0.5000
-0.5000
-0.5000
-0.0000

```

```

v1 = v(1,:)

```

```

v1 = 1x4
-0.5000  -0.5000  -0.5000  -0.5000

```

```

u2 = u(:,2) %sparsece - so will be localized

```

```

u2 = 5x1
0.0000
-0.0000
0.0000
0.0000
-1.0000

```

```

v2 = v(2,:)

```

```

v2 = 1x4
0.5000  -0.5000   0.5000  -0.5000

```

```

S1 = s(1,1)*(u1)*(v1)

```

```

S1 = 5x4
2.0000    2.0000    2.0000    2.0000
2.0000    2.0000    2.0000    2.0000
2.0000    2.0000    2.0000    2.0000
2.0000    2.0000    2.0000    2.0000
0.0000    0.0000    0.0000    0.0000

```

```

S2 = s(2,2)*(u2)*(v2)

```

```

S2 = 5x4
0.0000  -0.0000   0.0000  -0.0000
-0.0000   0.0000  -0.0000   0.0000
0.0000  -0.0000   0.0000  -0.0000
0.0000  -0.0000   0.0000  -0.0000
-3.0000   3.0000  -3.0000   3.0000

```

```

S1+S2 % RECONSTRUCT THE MATRIX

```

```

ans = 5x4

```



2.0000	2.0000	2.0000	2.0000
2.0000	2.0000	2.0000	2.0000
2.0000	2.0000	2.0000	2.0000
2.0000	2.0000	2.0000	2.0000
-3.0000	3.0000	-3.0000	3.0000

## CONVEXITY TEST

**we say  $f$  is convex on  $S$  if :**

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y)$$

for all  $x, y \in S$  and all  $\lambda \in [0, 1]$

### 1) rank()

$$X = \begin{bmatrix} 5 & 0 \\ 0 & 0 \end{bmatrix} \quad Y = \begin{bmatrix} 0 & 0 \\ 0 & 5 \end{bmatrix} \quad \lambda = 0.1$$

$$\text{rank}\left(\begin{bmatrix} 0.5 & 0 \\ 0 & 4.5 \end{bmatrix}\right) \not\leq 0.1 * \text{rank}\left(\begin{bmatrix} 5 & 0 \\ 0 & 0 \end{bmatrix}\right) + 0.9 * \text{rank}\left(\begin{bmatrix} 0 & 0 \\ 0 & 5 \end{bmatrix}\right)$$

$$2 \not\leq 0.1 * 1 + 0.9 * 1$$

```
x = [5 0;0 0]
```

```
x = 2x2
     5     0
     0     0
```

```
y = [0 0;0 5]
```

```
y = 2x2
     0     0
     0     5
```

```
la = 0.1
```

```
la = 0.1000
```

```
aaa=rank(la*x + (1-la)*y)
```

```
aaa = 2
```

```
bbb=0.1*rank(x) + 0.9*rank(y)
```

```
bbb = 1
```

### 2) Nuclear Norm

Let  $x := \begin{bmatrix} 5 \\ 0 \end{bmatrix}$  and  $y := \begin{bmatrix} 0 \\ 5 \end{bmatrix}$  for any  $\lambda \in (0, 1)$

$$\|\lambda x + (1 - \lambda)y\|_* \leq \lambda \|x\|_* + (1 - \lambda) \|y\|_*$$

$$\lambda = 0.1$$

$$\begin{bmatrix} 0.707 \\ 0 \end{bmatrix} \leq \begin{bmatrix} 5 \\ 0 \end{bmatrix}$$

```
x= [5 0]';
y = [0 5]';
la = 0.1;
c= la*x+la*y;
x11=norm(svd(c,"vector"));

x22 = la*norm(svd(x)) + (1-la)*norm(svd(y))
```

```
x22 = 5
```

## Using CVX to solve the constrained Opt Problem

$$\min_{X \in \mathbb{R}^{m \times n}} \|X\|_* \quad \text{such that } X_{\Omega} = \vec{y}.$$

```
%Solving using CVX
```

```
syms x1 x2 x3 x4
```

```
X = [1 x1 5 4 ; x2 1 4 5 ; 4 5 2 x3 ; 5 4 2 1 ; 4 5 1 2 ; 1 2 x4 5] %matrix with unknown as var
```

```
X =
```

$$\begin{pmatrix} 1 & x_1 & 5 & 4 \\ x_2 & 1 & 4 & 5 \\ 4 & 5 & 2 & x_3 \\ 5 & 4 & 2 & 1 \\ 4 & 5 & 1 & 2 \\ 1 & 2 & x_4 & 5 \end{pmatrix}$$

```
f = sum(sum(X)) %summing for caculating average
```

```
f = x1 + x2 + x3 + x4 + 63
```

```
mu = vpa(subs(f,[x1 x2 x3 x4],[0 0 0 0])/20) % mean is calculated
```

```
mu = 3.15
```

```
mu = double(mu)
```

```
mu = 3.1500
```

```
m = 6;  
n = 4;  
lambda = 1;
```

```
% CVX baseline
```

```
cvx_begin quiet
```

```
variable X(m,n)
```

```
minimize(norm_nuc(X)) % minimizing nuclear norm
```

```
subject to % constrains: the given values are the known part of matrix
```

```
X(1,1)==1-mu
```

```
X(1,3)==5-mu
```

```
X(1,4)==4-mu
```

```
X(2,2)==1-mu
```

```
X(2,3)==4-mu
```

```
X(2,4)==5-mu
```

```
X(3,1)==4-mu
```

```
X(3,2)==5-mu
```

```
X(3,3)==2-mu
```

```
X(4,:)==[5,4,2,1]-mu
```

```
X(5,:)==[4,5,1,2]-mu
```

```
X(6,1)==1-mu
```

```
X(6,2)==2-mu
```

```
X(6,4)==5-mu
```

```
cvx_end
```

```
round(full(X)+mu) % Final matrix is rounded , it almost match the original matches
```

```
ans = 6x4
```

```
1 3 5 4  
2 1 4 5  
4 5 2 2  
5 4 2 1  
4 5 1 2  
1 2 5 5
```

$$\begin{pmatrix} 1 & 2(1) & 5 & 4 \\ 2(2) & 1 & 4 & 5 \\ 4 & 5 & 2 & 2(1) \\ 5 & 4 & 2 & 1 \\ 4 & 5 & 1 & 2 \\ 1 & 2 & 5(5) & 5 \end{pmatrix}$$

## Using cvx to solve the unconstrained Opt Problem.

$$\min_{X \in \mathbb{R}^{m \times n}} \frac{1}{2} \|X_{\Omega} - \vec{y}\|_2^2 + \lambda \|X\|_*$$

```
%Solving using CVX
```

```
lamda = 0.15 % hyper parameter set after trying many
```

```
lamda = 0.1500
```

```
%CVX
```

```
cvx_begin quiet
```

```
    variable X(6,4)
```

```
    y = [1 ; 5 ; 4 ; 1 ; 4 ; 5 ; 4 ; 5 ; 2 ; 5 ; 4 ; 2 ; 1 ; 4 ; 5 ; 1 ; 2 ; 1 ; 2 ; 5] -mu ; % the known part of y
```

```
    subject to % constraints
```

```
    xx = [X(1,1);X(1,3);X(1,4);X(2,2);X(2,3);X(2,4);X(3,1);  
X(3,2);X(3,3);X(4,1);X(4,2);X(4,3);X(4,4);X(5,1);X(5,2);  
X(5,3);X(5,4);X(6,1);X(6,2);X(6,4);]
```

```
xx =
```

```
cvx real affine expression (20x1 vector)
```

```
    minimize( 0.5*norm(xx - y,2) + lamda*norm_nuc(X)) ; % cost function  
cvx_end
```

```
    round(X+mu) % Final matrix is rounded , it almost match the original matches
```

```
ans = 6x4
```

1	3	5	4
2	1	4	5
4	5	2	2
5	4	2	1
4	5	1	2
1	2	5	5

$$\begin{pmatrix} 1 & \textcolor{red}{2} (1) & 5 & 4 \\ \textcolor{red}{2} (2) & 1 & 4 & 5 \\ 4 & 5 & 2 & \textcolor{red}{2} (1) \\ 5 & 4 & 2 & 1 \\ 4 & 5 & 1 & 2 \\ 1 & 2 & \textcolor{red}{5} (5) & 5 \end{pmatrix}$$

## Solving Proximal Based on Algorithm

**Algorithm 2.4** (Proximal-gradient method for nuclear-norm regularization). *Let  $Y$  be a matrix such that  $Y_{\Omega} = y$  and let us abuse notation by interpreting  $X_{\Omega}^{(k)}$  as a matrix which is zero on  $\Omega^c$ . We set the initial point  $X^{(0)}$  to  $Y$ . Then we iterate the update*

$$X^{(k+1)} = \mathcal{D}_{\alpha_k \lambda} \left( X^{(k)} - \alpha_k \left( X_{\Omega}^{(k)} - Y \right) \right),$$

where  $\alpha_k > 0$  is the step size.

```
% to find an optimal value for the
% hyperparameters alpha and threshold values so we have experimented with
% several combinations of different range .
```

```
A = [1 1 5 4; 2 1 4 5 ;4 5 2 1 ;5 4 2 1 ;4 5 1 2 ;1 2 5 5] %matrix
```

```
A = 6x4
    1    1    5    4
    2    1    4    5
    4    5    2    1
    5    4    2    1
    4    5    1    2
    1    2    5    5
```

```
Y = [1 0 5 4 ; 0 1 4 5 ; 4 5 2 0 ; 5 4 2 1 ; 4 5 1 2 ; 1 2 0 5] - mu %the matrix missing elements
```

```
Y = 6x4
-2.1500 -3.1500  1.8500  0.8500
-3.1500 -2.1500  0.8500  1.8500
 0.8500  1.8500 -1.1500 -3.1500
 1.8500  0.8500 -1.1500 -2.1500
 0.8500  1.8500 -2.1500 -1.1500
-2.1500 -1.1500 -3.1500  1.8500
```

```
Y(1,2)=0;
Y(2,1)=0;
Y(3,4)=0;
Y(6,3)=0;
```

```
X = Y
```

```
X = 6x4
    -2.1500         0     1.8500     0.8500
         0    -2.1500     0.8500     1.8500
     0.8500     1.8500    -1.1500         0
     1.8500     0.8500    -1.1500    -2.1500
     0.8500     1.8500    -2.1500    -1.1500
    -2.1500    -1.1500         0     1.8500
```

```
[n1,n2] = size(Y);
small = 99999999 ;% setting small value to compare results

%different combinations of hyper parametres tested
for q=[-9:9]
    for p=[-9:9]

        alpha = 10^(p);
        threshold = 10^(q) ;

        %We do a 100 iterations , as larger ones yielded bad results
        for i =[0:100]
            XK = X;

            XK(1,2)=0;
            XK(2,1)=0;
            XK(3,4)=0;
            XK(6,3)=0;
            X = Sft_thresh_singular_vals( X - alpha*( XK - Y),threshold); % update as per algo

        end

        if small > norm(X+mu - A)
            small = norm(X+mu - A);
            s=X;
            val = [10^(p),10^(q)]; % optimal hyperparameter

        end
    end
end

% optimal hyperparameter
val(1) % optimal alpha
```

```
ans = 1
```

```
val(2) % optimal threshold
```

```
ans = 1.0000e-09
```

```
round(s+mu)
```

```
ans = 6x4
     1     2     5     4
```

2	1	4	5
4	5	2	2
5	4	2	1
4	5	1	2
1	2	4	5

$$\begin{pmatrix} 1 & \textcolor{red}{2} (1) & 5 & 4 \\ \textcolor{red}{2} (2) & 1 & 4 & 5 \\ 4 & 5 & 2 & \textcolor{red}{2} (1) \\ 5 & 4 & 2 & 1 \\ 4 & 5 & 1 & 2 \\ 1 & 2 & \textcolor{red}{5} (5) & 5 \end{pmatrix}$$

```
function D = Sft_thresh_singular_vals(Y,tou)
    %Soft thresholding of singular values based the given parameter tou
    [U,S,Vt] = svd(Y);
    [m,n] = size(S);
    for i = [1:n-1]
        if S(i,i) > tou
            S(i,i) = S(i,i) - tou;    % we subtract if > tou
        else
            S(i,i) = 0;                % else it will be set to 0
        end
    end
    D = U*S*Vt;                       %Final matrix that we return
end
```