Summer 8-31-2017

Matrix completion algorithms with applications in biomedicine, e-commerce and social science

Yiran Wang
New Jersey Institute of Technology

Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

MATRIX COMPLETION ALGORITHMS WITH APPLICATIONS IN BIOMEDICINE, E-COMMERCE AND SOCIAL SCIENCE

by
Yiran Wang

This thesis investigates matrix completion algorithms with applications in biomedicine, e-commerce and social science. In general, matrix completion algorithms work well for low rank matrices. Such matrices find many applications in recommender systems and social network analysis. On the other hand, biological networks often yield high rank matrices. For example, the adjacency matrix representing interactions between transcription factors and target genes in the cell is a highly sparse matrix, in which most entries correspond to absent interactions and only a few entries correspond to present interactions. This sparse matrix is a high rank or even full rank matrix. Matrix completion algorithms do not work well for high rank matrices. In this thesis, several experiments are conducted to evaluate the performance of matrix completion algorithms for both low rank and high rank matrices. A new high rank matrix completion method is proposed,which is designed to process adjacency matrices representing interactions between transcription factors and target genes in cells.

MATRIX COMPLETION ALGORITHMS WITH APPLICATIONS IN BIOMEDICINE, E-COMMERCE AND SOCIAL SCIENCE

by
Yiran Wang

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Sciences

Department of Computer Sciences

May 2017

APPROVAL PAGE

MATRIX COMPLETION ALGORITHMS WITH APPLICATIONS IN BIOMEDICINE, E-COMMERCE AND SOCIAL SCIENCE

Yiran Wang

Dr. Jason T.L. Wang, Thesis Advisor — Date
Professor of Bioinformatics and Computer Science, NJIT

Dr. Xiaoning Ding, Committee Member — Date
Assistant Professor of Computer Science , NJIT

Dr. Chase Qishi Wu, Committee Member — Date
Associate Professor of Computer Science , NJIT

BIOGRAPHICAL SKETCH

Author: Yiran Wang

Degree: Master of Science

Date: May 2017

Undergraduate and Graduate Education:

- Bachelor of Science in Computer Science,
  University of Macau, Macau, China, 2015

Major: Computer Sciences

Presentations and Publications:

Hua, Zhongyun and Wang, Yiran and Zhou, Yicong, "Image Cipher Using a New Interactive Two-Dimensional Chaotic Map," *Systems, Man, and Cybernetics (SMC), 2015 IEEE International Conference on*, pp 1804-1808, 2015.

This thesis is dedicated to my parents.

ACKNOWLEDGMENT

I would like to thank my advisor Dr.Wang for his patience and encourage on me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Matrix completion has been widely used in many fields of biomedicine, e-Commerce and social Science, such as recommendation system [2], analyzing social networks [4], analyzing biomedicine datasets [10–12] and image processing [5].Those problems can be solved by matrix are always based on an assumption that the final matrix should be low rank. This is quite reasonable for some problems. Let's take Netflix problem as an example [2]. Generally, preference of different users can be clustered into several cliques. A certain user's preference usually can be classified into few cliques. That will give a matrix a low rank attribute. In this thesis, we want to see the performance of low rank matrix working on dataset which can be represented in low rank matrix and the feasibility of using high rank matrix completion to predict Interactions between transcription factors and target genes. Two low rank matrix completion method were introduced. Some experiments were performed on synthetic data to see their performance. We also proposed a high rank matrix completion algorithm and apply it on predicting unknown interactions between transcription factors and target genes. Different from those problems based on an assumption that the final matrix should be low rank, our final matrix are always high rank or even full rank. Although several researches have been made on high rank matrix completion [3], some works are still needed to make it practical and efficient. In our model, we use -1 to represent no edges between a transcription factor and a certain target gene and 1 to represent there is an edge between a transcription factor and a certain target gene. The absolute value of an entry is not as important as its sign in this scenario, therefore fully recovery is not a must in this problem. Instead of finding one correct subspace for

the matrix, we prefer doing exhaustive search on all the possible subspaces. This may be not as efficient as the original algorithm, but it can improve accuracy and reliability of final output.

CHAPTER 2

LOW RANK MATRIX COMPLETIONS

2.1 GROUSE Algorithm

2.1.1 Introduction of GROUSE algorithm and Implementation

GROUSE (Grassmannian Rank-One Update Subspace Estimation) [1] is an efficient online algorithm for tracking subspaces from highly incomplete observations. This algorithm can also be applied on low matrix completion. The algorithm is showed as follow,

Algorithm 1 Grassmannian Rank-One Update Subspace Estimation

Input: An$n \times d$ orthogonal matrix U_0. A sequence of vectors v_t, each observed in entries Ω_t. A set of stepsizes η_t

Output: output (σ)

1: **for** $t = 1, 2, 3, ..., T$ **do**

2: Estimate weights $w = argmin_a \| \triangle_{\Omega_t}(U_t a - v_t) \|$

3: Predict full vectors $p = U_t w$

4: Compute residual $r = \triangle_{\Omega_t}(v_t - p)$

5: Update subspace $U_{t+1} = U_t + ((cos(\sigma\eta_t) - 1)\frac{p}{\|p\|} + (sin(\sigma\eta_t) - 1)\frac{r}{\|r\|})\frac{w^T}{\|w\|}$;

6: **end for**

For this algorithm, matrix completion can be thought of a subspace identification problem where only aim to identify the column space of the unknown low-rank matrix. Once the column space is identified, we can project the incomplete columns onto that subspace in order to complete the matrix [1].

2.1.2 Test and Result

Here we generate 4 different low rank matrix to simulate e-commerce data and social network data. The results are showed as follow, The comparison

Table 2.1: Synthetic data of low rank matrix 1

Columns	Rows	Rank
10	10	5

Table 2.2: Synthetic data of low rank matrix 2

Columns	Rows	Rank
20	20	5

Table 2.3: Synthetic data of low rank matrix 3

Columns	Rows	Rank
10	10	7

Table 2.4: Synthetic data of low rank matrix 4

Columns	Rows	Rank
20	20	7

are showed as follow, where $'o','v','*','.'$ represent data 1, data 2, data 3 and data 4 respectively It can be seen from those figures, that the rank doesn't affect BER so much when the matrix is low rank.

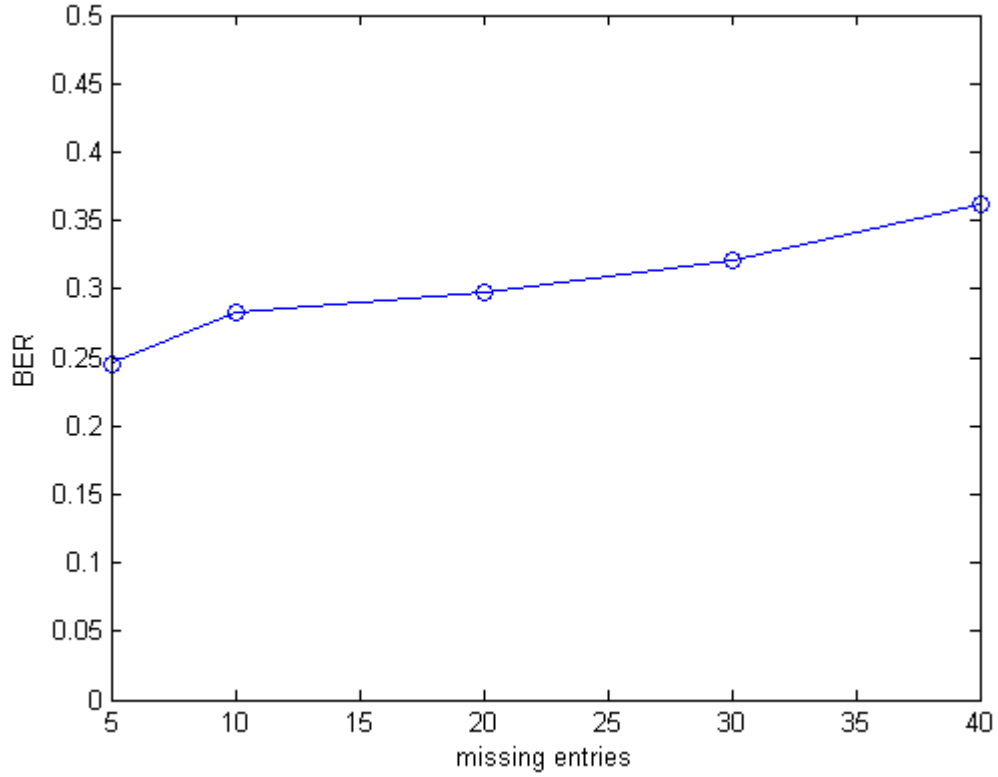

Figure 2.1: GROUSE result on synthetic data 1.

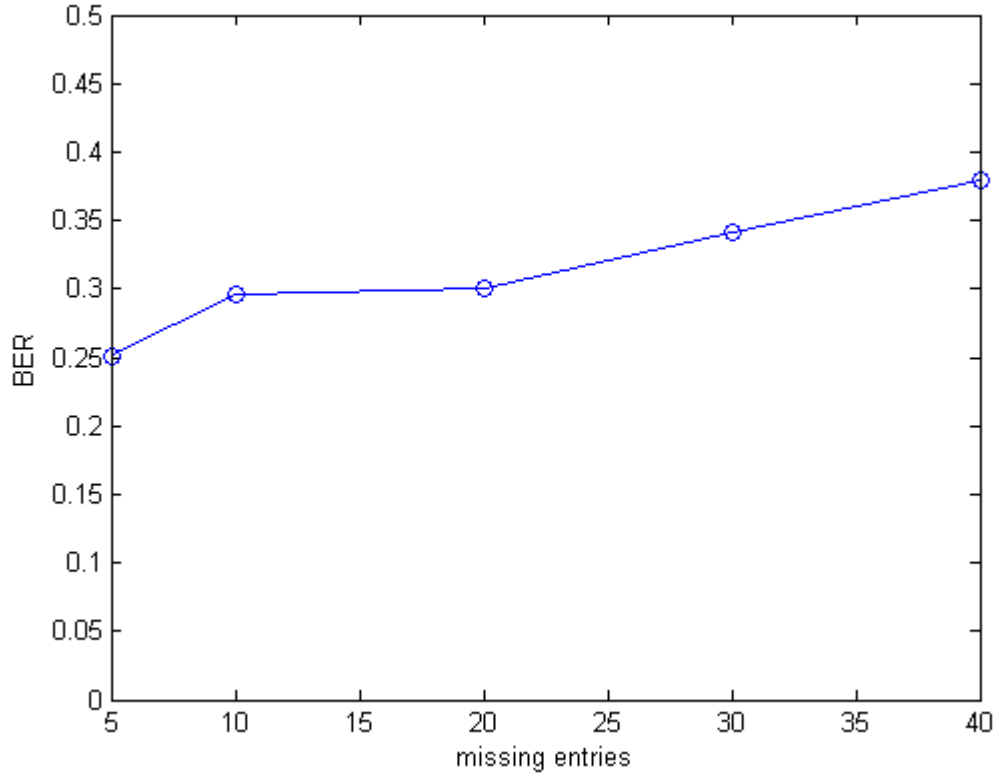

Figure 2.2: GROUSE result on synthetic data 2.

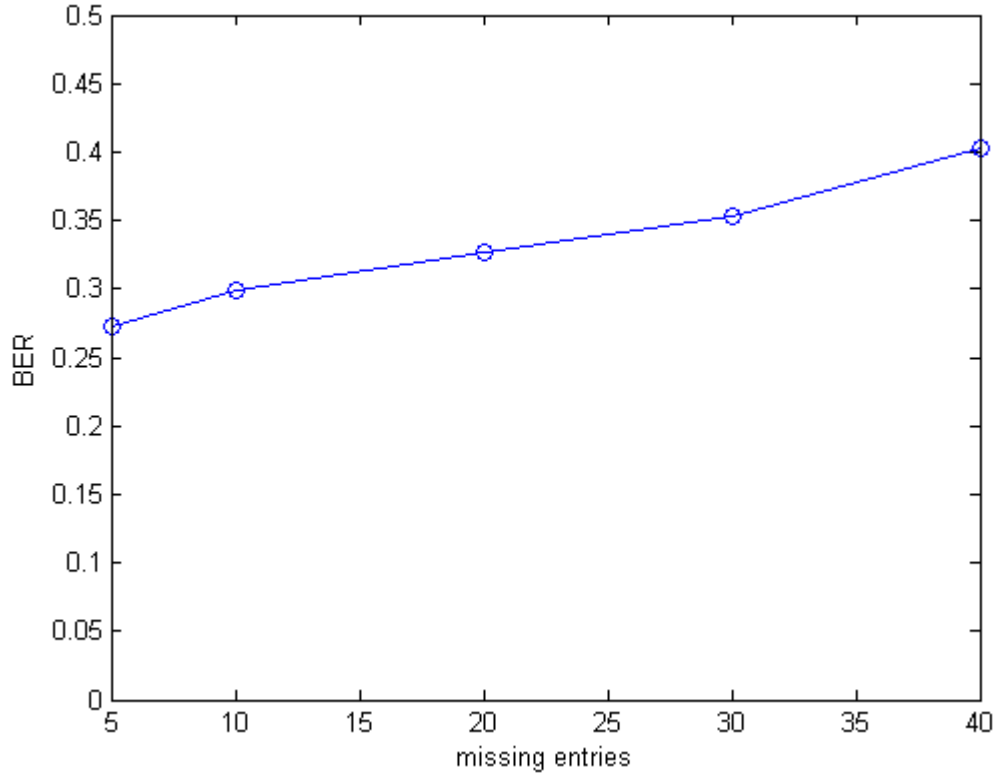

Figure 2.3: GROUSE result on synthetic data 3.

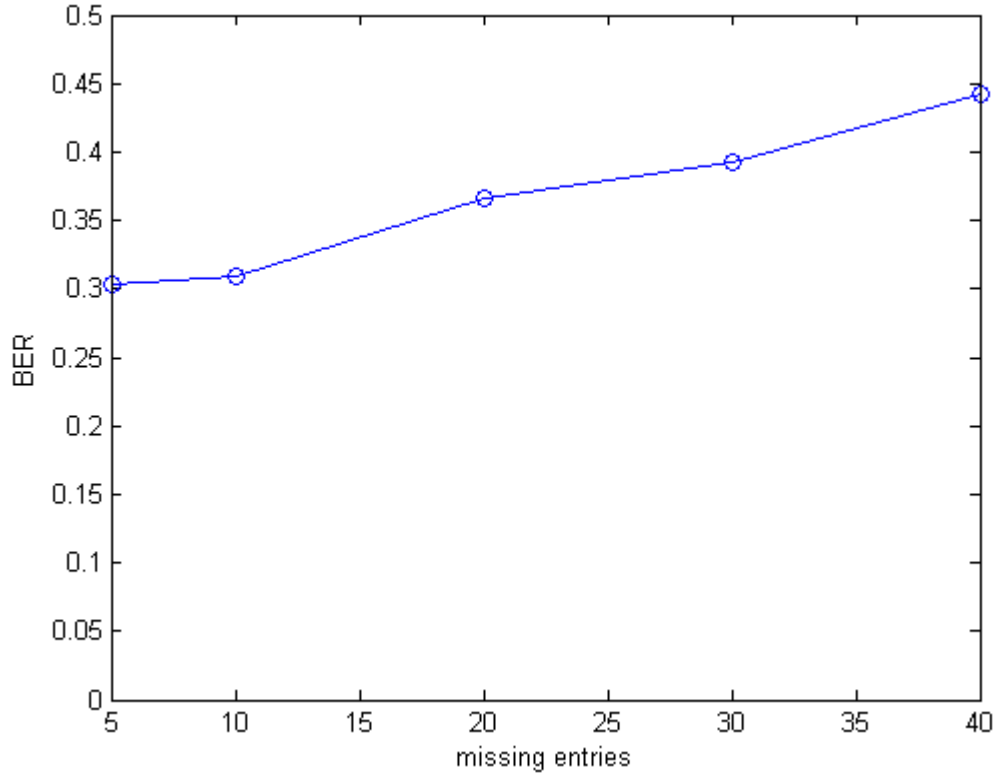

Figure 2.4: GROUSE result on synthetic data 4.

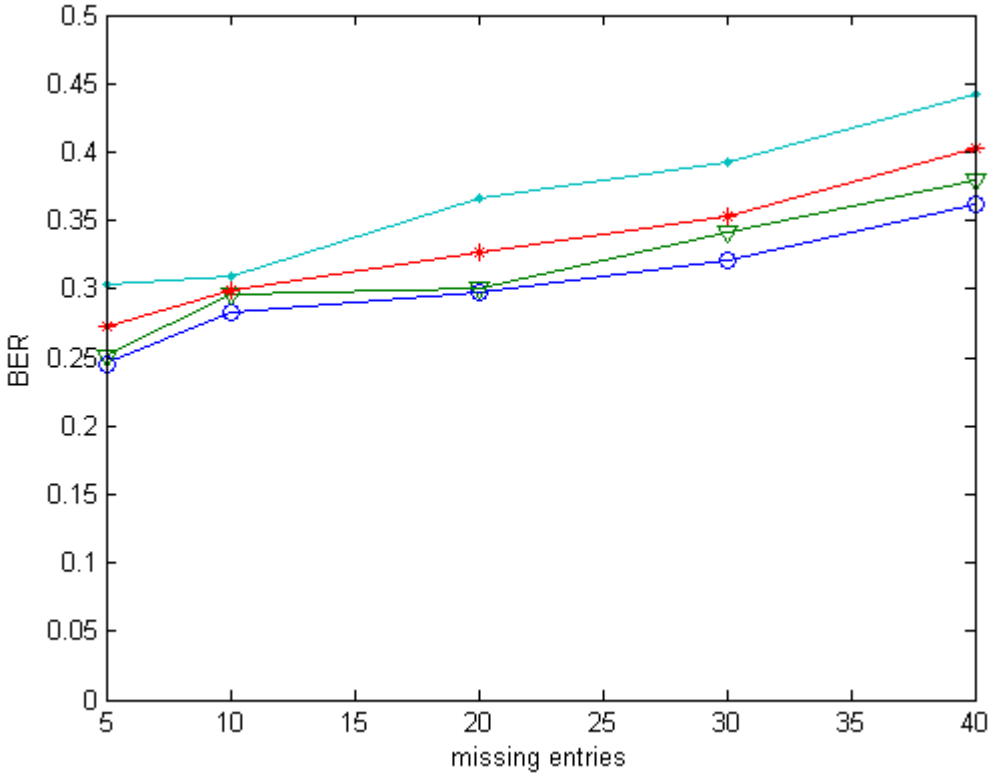

Figure 2.5: GROUSE comparison result of low rank data.

2.2 ALM Method

2.2.1 Introduction of ALM Method

The ALM (Augmented Lagrange Multipliers) method is a scalable and fast algorithms for solving the Robust PCA problem, namely recovering a low-rank matrix with an unknown fraction of its entries being arbitrarily corrupted [8]. The algorithm of exact ALM metohd are showed as follow,

Algorithm 2 RPCA via the exact ALM Method

Input: Observation Matrix $D \in R^{m \times n}$, λ

Output: output (A_k, E_k)

$Y_0 = sgn(D)/J(sgn(D)); \mu_0 = 0; \rho > 1; K = 0$

while not converged **do**

  $A^0_{k+1} = A^*_k, E^0_{k+1} = E^*_k, j = 0;$

  while not converged **do**

    $(U, S, V) = svd(D - E^j_{k+1} + \mu_k^{-1} Y^*_k)$

    $A^{j+1}_{k+1} = US_{\mu_k^{-1}}[S]V^T$

    $E^{j+1}_{k+1} = S_{\lambda\mu_k^{-1}}[D - A^{j+1}_{k+1} + \mu_k^{-1} Y^*_k]$

  end while

  $Y_{k+1} = Y_k + \mu_k(N_j - A_{k+1} - E_{k+1}); \mu_{k+1} = \rho\mu_k$

  $k = k + 1$

end while

return result

2.2.2 Test and Result

Here we use the same synthetic data from GROUSE, the results are shown as follow,

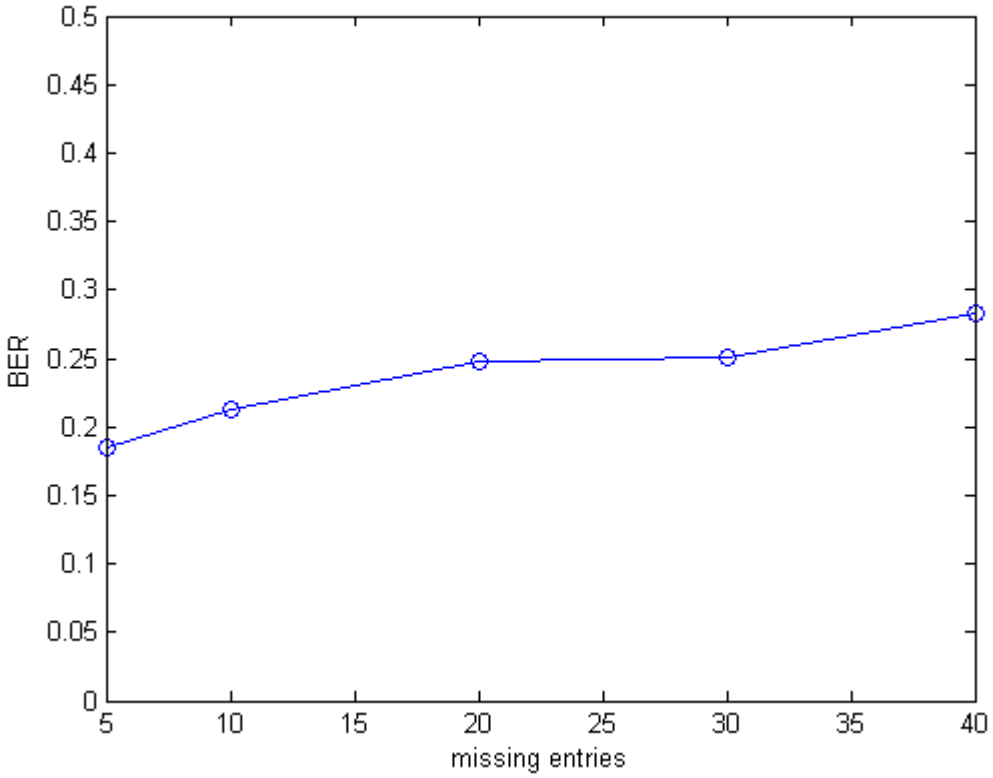

Figure 2.6: Exact ALM result on synthetic data 1.

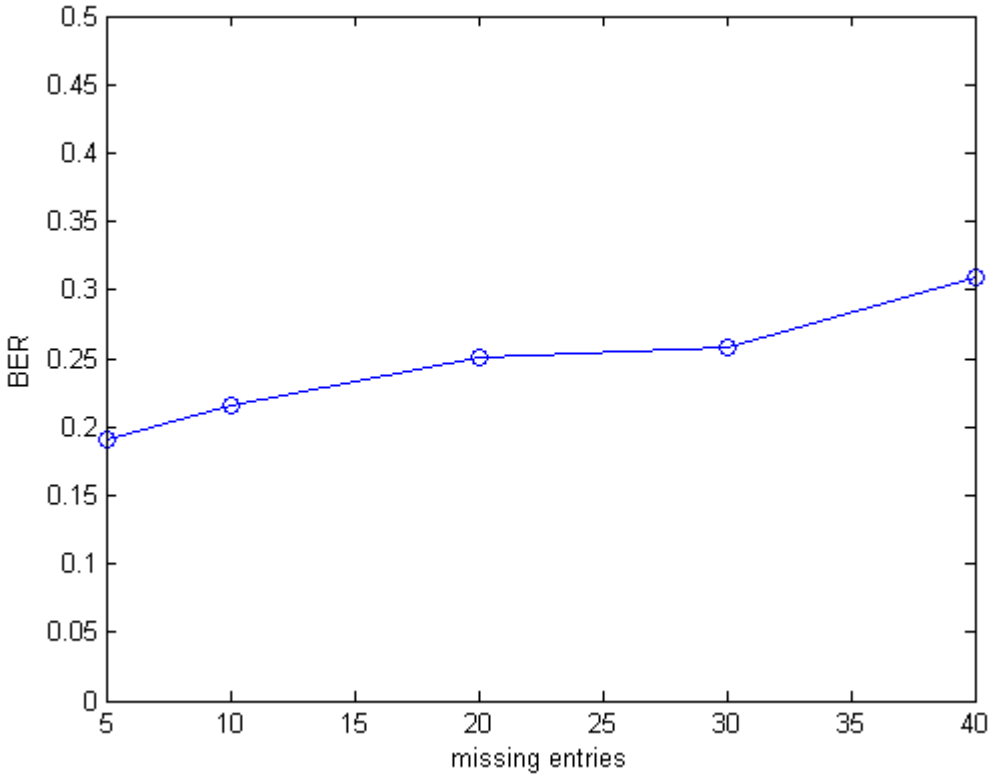

Figure 2.7: Exact ALM result on synthetic data 2.

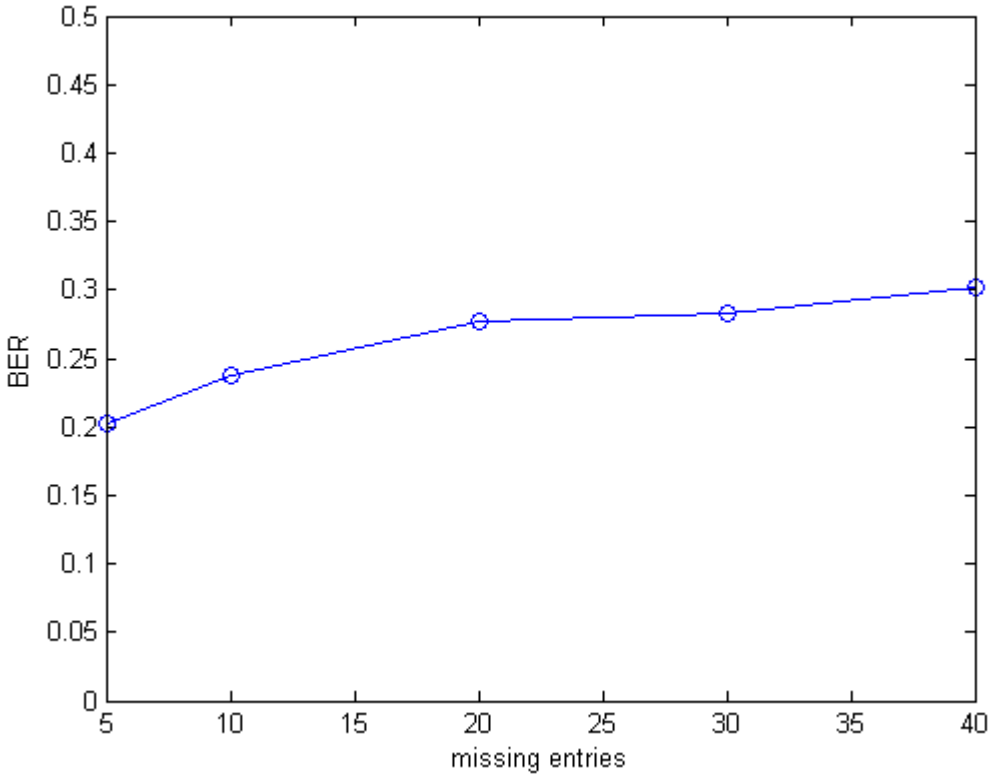

Figure 2.8: Exact ALM result on synthetic data 3.

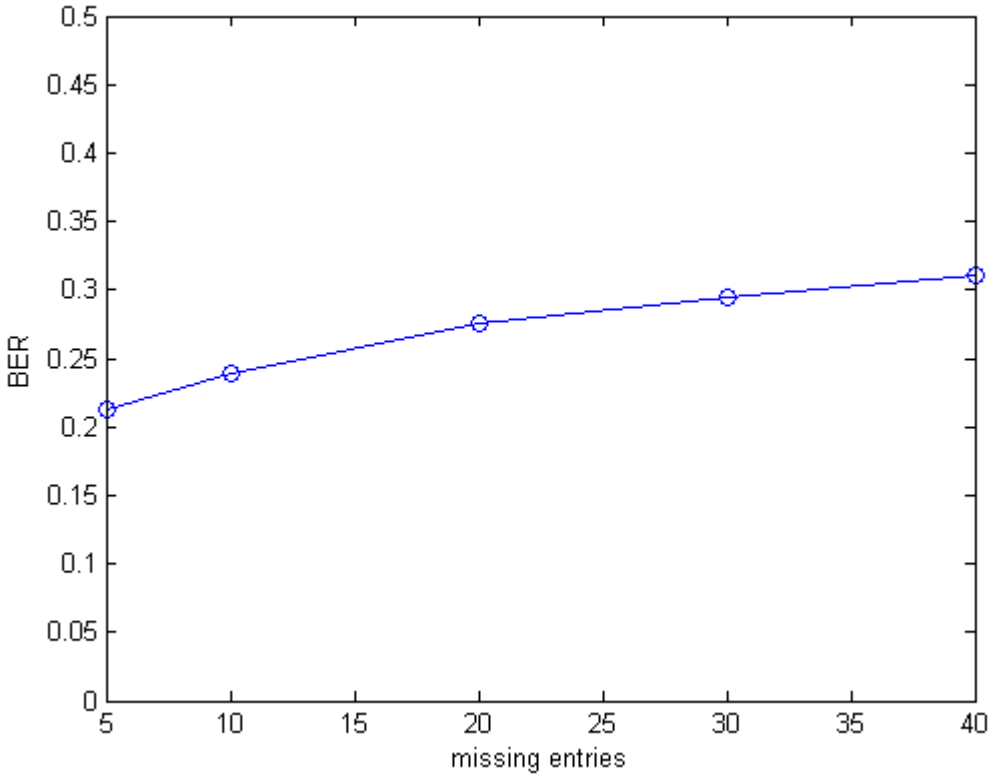

Figure 2.9: Exact ALM result on synthetic data 4.

The comparison results are,

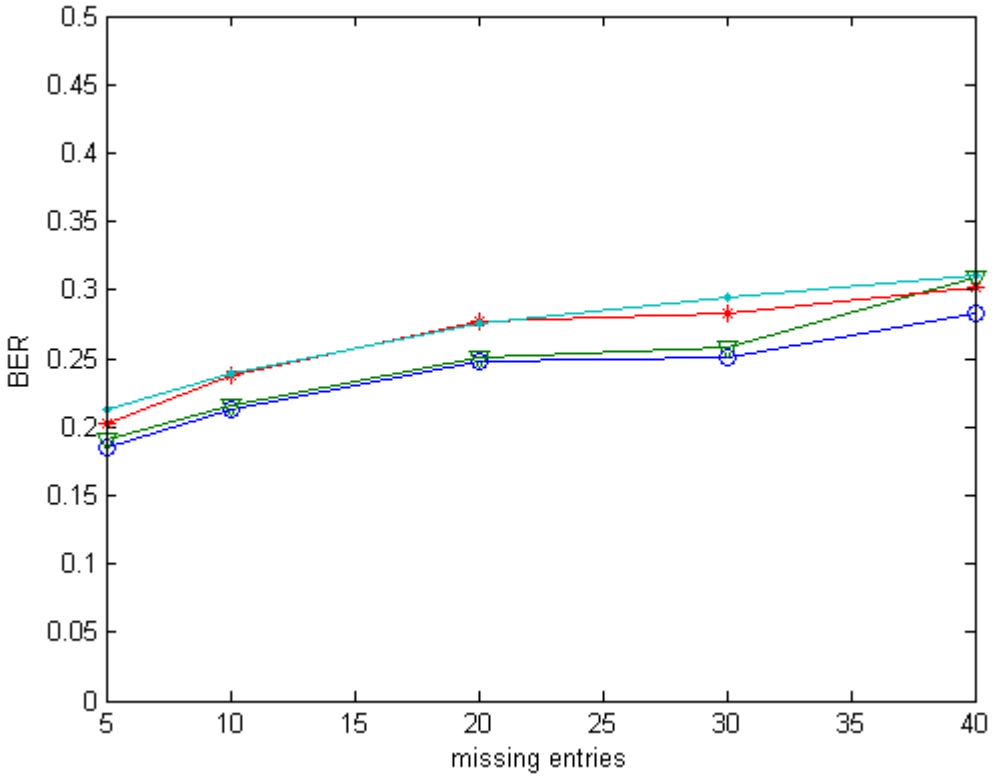

Figure 2.10: Exact ALM comparison result of low rank data.

where $'o','v','*','.'$ represent data 1, data 2, data 3 and data 4 respectively. The result is quite similar with the result of GROUSE algorithm.The rank doesn't affect BER so much when the matrix is low rank.

CHAPTER 3

FEASIBILITY OF HIGH RANK MATRIX COMPLETION APPLIED ON BIOMEDICINE

3.1 Materials and Methods

3.1.1 Interactions between Transcription Factors and Target Genes

The original ChEA 2016 dataset has 645 transcription factors and 49238 unique target genes. In the original data, the first entry of each rows is a transcription factor, then followed by a list of related target genes.Thus, to apply our prediction based on high rank matrix completion on factor-target gene interactions data, pre-process on original data set is required. We need to represent data set in the form of an adjacency matrix. First, define a matrix $X \in R^{N \times n}$ including M transcription factors and N target genes. Set $\{T_1, T_2, T_3 ... T_M\}$ is transcription factor set, and set $\{G_1, G_2, G_3 ... G_M\}$ is target gene set [7]. In order to represent the edges between transcription factors and target genes, we use 3 values $-1, 0, 1$ to presents these different relationships. Value 1 means there exists an edge from transcription factor to a target gene, while the -1 means there is no interaction between a transcription factor and a gene. Value 0 represents the relationship between a transcription factor and a target gene is unknown.

$$X(r,c) = \begin{cases} -1 & \text{No edge} \\ 0 & \text{Unknown} \\ 1 & \text{has a edge} \end{cases} \tag{3.1}$$

Where $0 < r \leq N, 0 < c \leq n$.

3.1.2 Basic Assumptions

After getting the adjacency matrix $X \in R^{N \times n}$, we can consider this matrix as a set of N incomplete vectors.

$$\begin{pmatrix} | & | & \cdots & | & | \\ v_1 & v_2 & \cdots & v_{N-1} & v_N \\ | & | & \cdots & | & | \end{pmatrix} \tag{3.2}$$

We assume that each complete vector lies in a d-dimensional subspace, where $d <= n$. If correct subspace for each vector is found, then we can project incomplete vector onto their corresponding subspace to get the complete vector accurately. However, the absolute value of each entry is meaningless in this scenario. Only the sign of the value determines if there is an edge between a transcription factor and a gene. Thus, instead of getting the correct subspaces, we prefer to project incomplete vectors onto all the possible subspaces we get. The sign will be determined by the occurrence of positive values and negative values.

$$X_{predict}(r, c) = \begin{cases} -1 & \text{if more negative values} \\ 1 & \text{if more positive values} \end{cases} \tag{3.3}$$

Based on the assumption that the columns of a high-rank matrix belong to a union of multiple low rank subspaces. A high rank matrix completion problem can be regarded as multiple low-rank matrix completion problems on those subspaces.Therefore, we can formulate this problem into a high rank matrix completion problem. The relationship between different genes can be represented with an adjacency matrix X with size $n \times N$. Assume that the columns of X lie in a union of at most k subspaces of R^n, $k = o(n^d)$ for some $d > 1$. Each subspace has dimension at most $r < n$ and $N > kn$. A "high-rank" situation may have a total

rank with n. The coherence of an r-dimensional subspace $S \subseteq R^n$ is defined as,

$$\mu(S) := \frac{n}{r} \max_j \|P_S e_j\|_2^2 \tag{3.4}$$

where P_S is the projection operator onto S and e_j are the canonical vectors for R^n. A vector of the complete n-dimensional real vector space is called a canonical unit vector [9]. With the definition, $1 \leq \mu(S) \leq \frac{n}{r}$. Main assumptions about X can be made [3].

- The columns of X lie in the union of at most k subspaces, with $k = o(n^d)$ for some $d > 1$. Each subspace has rank at most $r < n$.
- The coherence of each subspace is bounded above by μ_0.
- If $rank(S_i) = r_i$, then any subset of r_i columns from S_i can spans S_i. Let $0 < \epsilon < 1$ and S_{i,ϵ_0} denote the subset of points in S_i at least ϵ_0 distance away from any other space. There exist a constant $0 < V_0 \leq 1$, depending on ϵ_0, such that
    1. The probability that a column selected uniformly at random belongs to S_{i,ϵ_0} is at least $0/k$.
    2. If $x \in S_{i,\epsilon_0}$, then the probability that a column selected uniformly at random belongs to the ball of radius ϵ_0 centered at x is at least ${}_0\epsilon_0^r/k$

3.1.3 The Proposed Algorithm

Generate Local Neighborhood In the step, S_0 columns will be select randomly as seeds. $S0$ is determined as [3],

$$S_0 \geq \left\lceil \frac{k(\log k + 1/\delta^0)}{1 - e^{-4} v_0} \right\rceil \tag{3.5}$$

$$\delta^0 = n^{2-2\beta^{1/2}} \log n \tag{3.6}$$

for some $\beta > 1$ and $0 < v_0 \leq 1$. This S makes we can have enough number of observed entries with a relatively high probability. For each seed, find all columns with t_0 observations in common. t_0 is defined as,

$$t_0 = \left\lceil 2\mu_0^2 \log(2S_0 l_0 n/\delta_0) \right\rceil \tag{3.7}$$

$$l_0 = \left\lceil \max\left\{ \frac{2kn}{v_0(\frac{\epsilon_0}{\sqrt{3}})^r}, \frac{8k\log(s/\delta_0)}{v_0(\frac{\epsilon_0}{\sqrt{3}})^r} \right\} \right\rceil \tag{3.8}$$

Where $0 < \epsilon < 1$. Now we have S_0 seeds with their corresponding set. Randomly select $l_0 n$ columns from each such set. Finally, for each seed a set of n columns from the same subspace as the seed must be found. This can be accomplished by identifying columns that are ϵ close to the seeds. The local neighborhood will be generated by random selecting n columns with partial distance less than $\epsilon/\sqrt{2}$ from the seed. This also implies that the column number N should be large enough so that n or more such columns could be found [3].

Subspace Completion For each of our local neighbor sets,we do matrix completion on those incompletely $n \times n$ using IALM method [8]. If matrix completion fails, then the seed together with its neighborhood will be discarded.

Algorithm 3 RPCA via the Inexact ALM Method

Input: Local neighborhood N_j, λ

Output: output (A_k, E_k)

1: $Y_0 = N_j / J(N_j); E_0 = 0; \mu_0 > 0; \rho > 1; K = 0$

2: **while** not converged **do**

3: $(U, S, V) = svd(N_j - E_k + \mu_k^{-1} Y_k)$

4: $A_{k+1} = US_{\mu_k^{-1}}[S]V^T$

5: $E_{k+1} = S_{\lambda\mu_k^{-1}}[N_j - A_{k+1} + \mu_k^{-1} Y_k]$

6: $Y_{k+1} = Y_k + \mu_k(N_j - A_{k+1} - E_{k+1}); \mu_{k+1} = \rho\mu_k$

7: **end while**

8: **return** result

Projection and Exhaustive Search Now assume we got S subspaces, $N_1, N_2, ..., N_j$. $S < S_0$, since some seeds and their corresponding neighbor were discarded in subspace completion. The first step is project incomplete columns onto those subspaces. Let $\bar{x}$ denotes complete columns, and x denotes incomplete columns.The projection operation is defined as,

$$\bar{x} = N(N^T N)^{\dagger} N^T x \tag{3.9}$$

† represents psuedo-inverse. We use pseudo-inverse to prevent $(N^T N)$ is not invertible. Psuedo-inverse is defined as,

$$\text{SVD of A:} \quad A = U \begin{pmatrix} S & 0 \\ 0 & 0 \end{pmatrix} V^T \tag{3.10}$$

$$\text{pseudo-inverse of A:} \quad A = V \begin{pmatrix} S^{-1} & 0 \\ 0 & 0 \end{pmatrix} U^T \tag{3.11}$$

After we finish projection on all possible subspaces, we will receive S complete matrix. For the relationship of genes are represented in by $+1$ as related and -1 as not related, the sign of values matters more compared to the absolute value. We simply generate results by counting occurrence of positive and negative on a position. The sign with a higher occurrence will be regarded as the correct one.

3.2 Result

In the experiments of high rank matrix completion,both synthetic data and real world data was tested to see the feasibility of high rank matrix completion can be applied on biomedicine. Our synthetic simulates a biomedicine dataset can be represented in high rank matrix. The ratio between the number of real gene interactions (edges) and the number of genes (nodes) falls between 1.5 and 2.75 regardless of the differences in phylogeny, phenotypic complexity, life history, and the total number of genes in an organism [6]. Here we chose 2.4 as our ratio to generate synthetic data. The ratio of transcription target between target gene are

Table 3.1: Synthetic data of high rank matrix

Gene	related factors	number of interactions
50	5	120

usually smaller than 2. We choose 5 transcription factors together with 50 different target genes in human and mouse as test dataset Here are the results of high rank

Table 3.2: human data

Gene	transcription factors	number of interactions
50	5	78

matrix completion applied on those data. In those figures, x-axis represents missing entries and y-axis represents balanced error rate. Among all those 3 results, only

the experiment on synthetic data has an acceptable result when missing entries is less than 50.

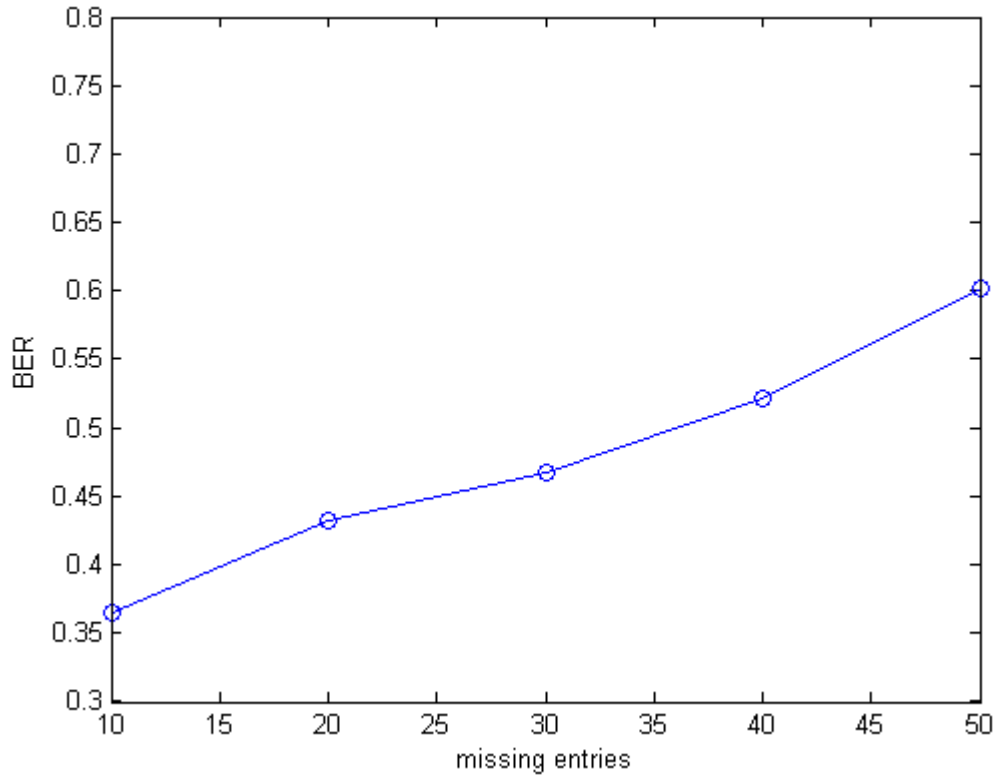

Figure 3.1: Result on synthetic data.

Table 3.3: mouse data

Gene	transcription factors	number of interactions
50	5	68

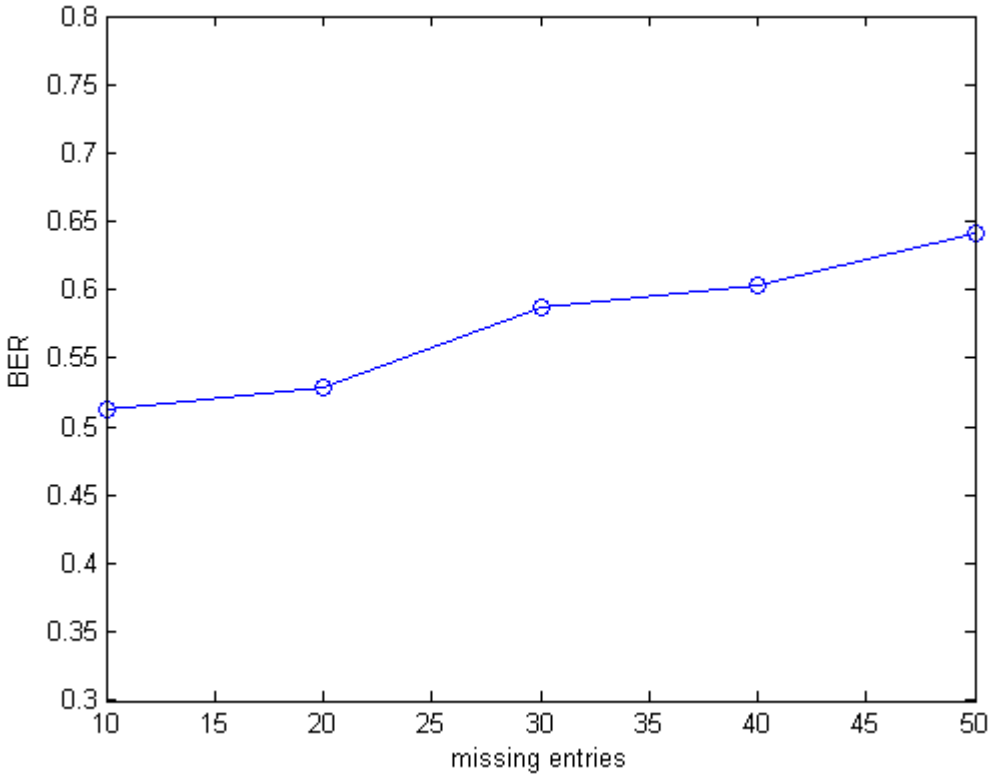

Figure 3.2: Result on mouse data.

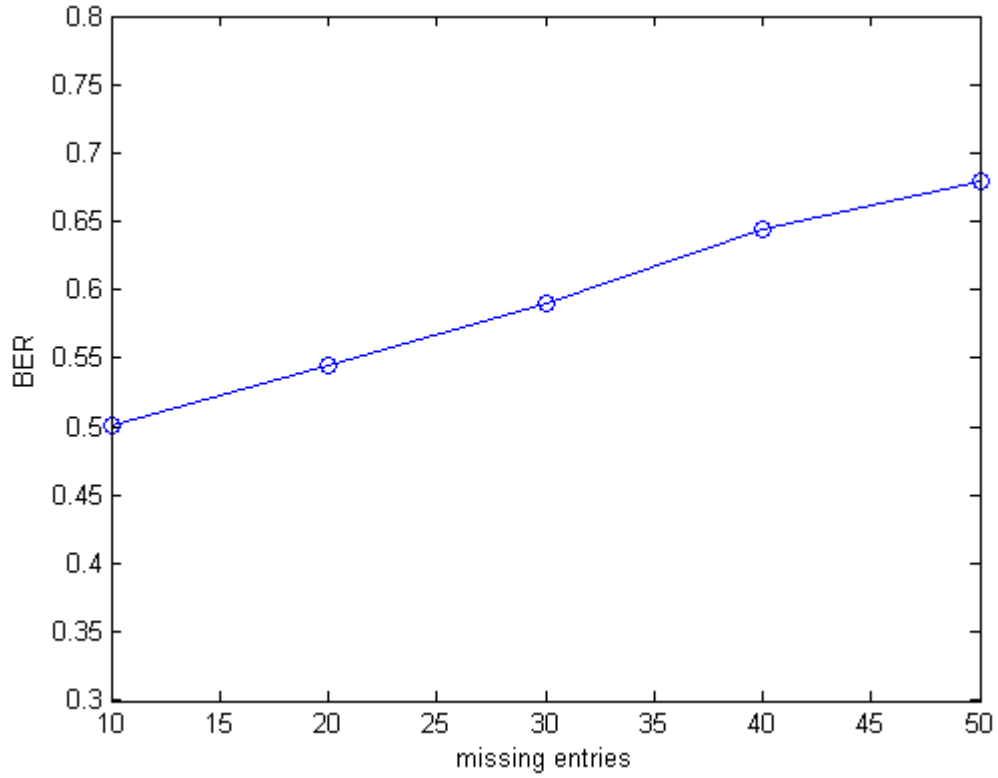

Figure 3.3: Result on human data.

The comparison of these 3 results are showed as follow, where $'o', 'v', '*'$ represent synthetic data, mouse data, human data respectively. The BER error rate increased dramatically while missing entries increasing. These results show that our algorithm may not be feasible in predicting unknown relationships between transcription factors and target genes, because there are not so many edges between transcription factors and target genes. Our basic assumption is that a high matrix may lay in multiple low rank subspaces. The lack of relationships between transcription factors and target genes may lead to a result that the high rank matrix representing this data set may not lay in any low rank subspaces.

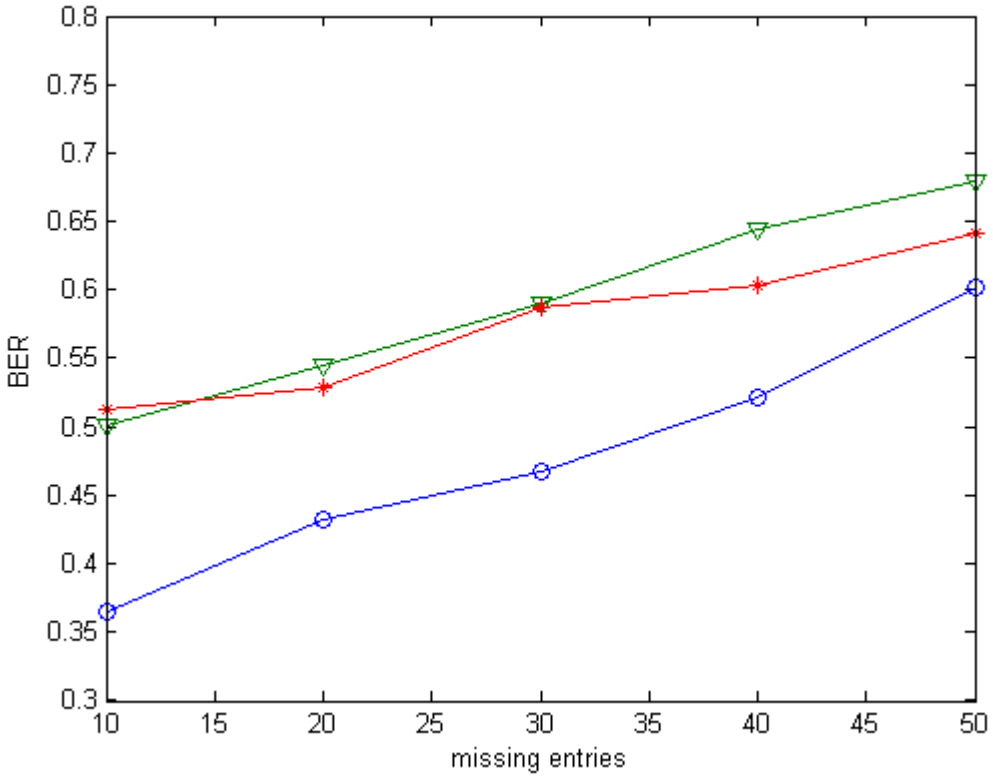

Figure 3.4: Comparison of 3 results.

CHAPTER 4

CONCLUSION

The experiments' results reflect the performance of low rank matrix completion and high rank matrix completion. Low rank matrix completion algorithms have good performance on predicting unknown entries of low rank matrix. However, our high rank matrix completion algorithm doesn't work well. The reason of this difference between low rank matrix completion and high rank matrix completion is that compared to high rank matrix, it is much easier for us to find the pattern of a low rank matrix. Only few rows of a low rank matrix is independent, while other rows are dependent to those rows. Once we find those independent rows or the pattern of those independent rows, we are able to recover other rows who are dependent to those rows. In this thesis, we try to solve a high rank matrix problem using a divide and conquer method. We tried to divide a high rank matrix completion problem into multiple low rank matrix completion problem based on an assumption that a high rank matrix completion may lay in multiple low rank subspaces. The results shows that this method are highly limited by data to be implemented. Some dataset may lay in multiple low rank subspaces, some may not. There are still works need to do in order to make high rank matrix completion feasible in biomedicine.

APPENDIX A

PROGRAM & DATASETS

```
function [X,err_reg] = test_alg(I,J,S,numr,numc,maxrank,
                                step_size,maxCycles,Uinit)

values = sparse(I,J,S,numr,numc);
Indicator = sparse(I,J,1,numr,numc);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%Main Algorithm
%

if (nargin<9)
    % initialize U to a random r-dimensional subspace
    U = orth(randn(numr,maxrank));
else
    U = Uinit;
end

err_reg = zeros(maxCycles*numc*100,1);


for outiter = 1:maxCycles*100,
```

```
    col_order = randperm(numc);

for k=1:numc,

    idx = find(Indicator(:,col_order(k)));
    v_Omega = values(idx,col_order(k));
    U_Omega = U(idx,:);

    weights = U_Omega\v_Omega;
    norm_weights = norm(weights);

    residual = v_Omega - U_Omega*weights;
    norm_residual = norm(residual);


    sG = norm_residual*norm_weights;
    err_reg((outiter-1)*numc + k) = norm_residual/norm(v_Omega);
    t = step_size*sG/( (outiter-1)*numc + k );

    if t<pi/2,
        alpha = (cos(t)-1)/norm_weights^2;
        beta = sin(t)/sG;

        step = U*(alpha*weights);
        step(idx) = step(idx) + beta*residual;
```

```
        U = U + step*weights';
    end
end

end
R = zeros(numc,maxrank);

for k=1:numc,
    idx = find(Indicator(:,k));
    v_Omega = values(idx,k);
    U_Omega = U(idx,:);
    R(k,:) = (U_Omega\v_Omega)';
end
 X = U*R';

function [A iter svp] = fast_alm_mc(D, tol, maxIter)
% This matlab code implements the inexact augmented Lagrange multiplie
% method for Matrix Completion.
%
% D - m x n matrix of observations/data (required input)
%
% tol - tolerance for stopping criterion.
%     - DEFAULT 1e-7 if omitted or -1.
%
% maxIter - maximum number of iterations
%         - DEFAULT 1000, if omitted or -1.
%
% Model:
```

```
%       min |A|_*
%       subj A + E = D, ProjectionOnOmega(E) =
clear global;
global A Sparse_Z;

addpath PROPACK;

[m n] = size(D);

if nargin < 2
    tol = 1e-4;
elseif tol == -1
    tol = 1e-4;
end

if nargin < 3
    maxIter = 1000;
elseif maxIter == -1
    maxIter = 1000;
end

% read sparse matrix D
[m n] = size(D);
[I J V] = find(D);
p = length(I);
col = [0; find(diff(J)); p];
Sparse_Z = D;
```

```
clear D;

% initialize
Y = zeros(p, 1);
Z = zeros(p, 1);
A.U = zeros(m, 5);
A.V = zeros(n, 5);
d_norm = norm(V, 'fro');
mu = 0.3/(lansvd('Axz','Atxz',m,n,1,'L'));
rho_s = p / (m * n);
rho = 1.1 + 2.5 * rho_s;

sv = 5;
svp = sv;

% Iteration
iter = 0;
converged = false;
stopCriterion = 1;
while ~converged
    %% alternative projection

    iter = iter + 1;
    if iter == 1
        Z = V;
    else
        Z = Z + 1/mu * Y;
```

```
    Sparse_Z = spconvert([I,J,Z; m,n,0]);
end
if stopCriterion > 10 * tol
    options.tol = 10*tol;
else
    options.tol = min(0.1*tol, 0.01/mu);
end
[A.U,S,A.V] = lansvd('Axz','Atxz',m,n,sv,'L',options);
%% predict the rank of A.
diagS = diag(S);
diagS = diagS(1:sv);
svn = length(find(diagS > 1/mu));
svp = svn;

ratio = diagS(1:end-1)./diagS(2:end);
[max_ratio, max_idx] = max(ratio);
if max_ratio > 2
    svp = min(svn, max_idx);
end
if svp < sv %|| iter < 10
    sv = min(svp + 1, n);
else
    sv = min(svp + 10, n);
end

%% update A Y Z mu
sqrtds = sqrt(diagS(1:svp) - 1/mu);
```

```
A.U = A.U(:, 1:svp) * diag(sqrtds);
A.V = A.V(:, 1:svp) * diag(sqrtds);

Z = UVtOmega(A.U,A.V,I,J,col);
Z = V - Z;
Y = Y + mu*Z;

%% stop Criterion
stopCriterion = norm(Z, 'fro') / d_norm;
if stopCriterion < tol
    converged = true;
end

%% display
if mod( iter, 10) == 0
    disp(['#svd ' num2str(iter) ' r(A) ' num2str(svp)...
        ' svn ' num2str(svn) ' max_idx ' num2str(max_idx) ' sv ' n
        ' stopCriterion ' num2str(stopCriterion) ' mu ' num2str(mu
end

%% Maximum iterations reached
if ~converged && iter >= maxIter
    disp('Maximum iterations reached') ;
    converged = 1 ;
end

%% update mu
```

```
    mu = mu*rho;

end

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <iostream>
#include <fstream>
using namespace std;


const int M = 5;
const int N = 50;
const double PE = 1.0 / 2.4;

void main()
{
        int a[M][N];
        int count = 0;

        srand(time(0));



        for (int i = 0; i < M; i++)
                for (int j = 0; j < N; j++)
```

```
        {
                a[i][j] = rand() % 2;
                if (a[i][j] == 1) count++;
        }


if (count > PE*M*N)
{
        for (int n = count, i, j; n - PE*M*N > 0; )
        {
                i = rand() % M;
                j = rand() % N;
                if (1 == a[i][j])
                {
                        a[i][j] = 0;
                        n--;
                }
        }
}
else if (count < PE*M*N)
{
        for (int n = count, i, j; PE*M*N - n > 0; )
        {
                i = rand() % M;
                j = rand() % N;
                if (0 == a[i][j])
                {
```

```
                                a[i][j] = 1;
                                n++;
                        }
                }
        }

        ofstream myfile("data.txt", ios::out);
        for (int i = 0; i < M; i++) {
        for (int j = 0; j<N; j++)
                {
                        printf("%d ", a[i][j]);
                        myfile<<(a[i][j])<<' ';
                }
                myfile <<"\n";
                printf("\n");
        }

        printf("\nDone.\n");
        system("pause");
}

function[e] = balance_error(X, correct, input)

[rows, cols] = size(X);
a = 0;
b = 0;
c = 0;
```

```
d = 0;

for i = 1:rows
    for j = 1:cols
        if (input(i,j) == 0 && X(i,j) > 0 && correct(i,j) == 1)
            a = a+1;
        end
        if (input(i,j) == 0 && X(i,j) < 0 && correct(i,j) == 1)
            b = b+1;
        end
        if (input(i,j) == 0 && X(i,j) < 0 && correct(i,j) == -1)
            d = d+1;
        end
        if (input(i,j) == 0 && X(i,j) > 0 && correct(i,j) == -1)
            c = c+1;
        end
    end
end
[a,b,c,d]
e_overall = ((b+c)/(a+b+c+d))
e = (b/(a+b) + c/(c+d))/2;

function y = UVtOmega(U,V,I,J,col);

y = zeros(length(I), 1);
for k = 1:length(col)-1
    j = J(col(k)+1);
```

```
    Xj = U * V(j,:)';
    idx = [col(k)+1:col(k+1)];
    y(idx) = Xj(I(idx));
end
newDataFlag = 1;

if newDataFlag == 1

    clc;
    close all;
    m = 1000;
    pdr = 6;
    n = m;
    r = 10;
    ML = (randn(m,r)); MR = (randn(n,r));
    p = min(round(r * (2 * n - r) * pdr), m*n);
    rho_s = p / (m * n);

    [I J col omega] = myRandsample(m, n, p);
    V = UVtOmega(ML, MR, I, J, col);

    D = spconvert([I,J,V; m,n,0]);
%     clear I J col;

end

disp('m');
```

```
disp(m);
disp('n');
disp(n);
disp('r');
disp(r);
disp('p');
disp(p);
disp('pdr');
disp(pdr);
disp('rho_s');
disp(rho_s);

% inexact alm mc
tic;
[A iter svp] = inexact_alm_mc(D, 1e-4);
tElapsed = toc;

trAA = sum(sum((A.U'*A.U) .* (A.V'*A.V)));
trMM = sum(sum((ML'*ML) .* (MR'*MR)));
trAM = sum(sum((A.U'*ML) .* (A.V'*MR)));
error = sqrt((trAA + trMM -2*trAM)/trMM);

disp('Iteration');
disp(iter);
disp('Time');
disp(tElapsed);
disp('rank of A');
```

```
disp(svp);
disp('|A-M| _F _/ _ |M| _F');
disp(error);

global LANBPRO_TRUTH
LANBPRO_TRUTH=0;

if LANBPRO_TRUTH==1
   global MU NU MUTRUE NUTRUE
   global MU_AFTER NU_AFTER MUTRUE_AFTER NUTRUE_AFTER
end

if nargin<1 | length(varargin)<2
   error('Not enough input arguments.');
end
narg=length(varargin);

A = varargin{1};
if isnumeric(A) | isstruct(A)
   if isnumeric(A)
      if ~isreal(A)
         error('A must be real')
      end
      [m n] = size(A);
   elseif isstruct(A)
      [m n] = size(A.R);
   end
```

```
  k=varargin{2};
  if narg >= 3 & ~isempty(varargin{3});
    p = varargin{3};
  else
    p = rand(m,1)-0.5;
  end
  if narg < 4, options = []; else options=varargin{4}; end
  if narg > 4
    if narg<7
      error('All or none of U_old, B_old and V_old must be provided.')
    else
      U = varargin{5}; B_k = varargin{6}; V = varargin{7};
    end
  else
    U = []; B_k = []; V = [];
  end
  if narg > 7, anorm=varargin{8}; else anorm = []; end
else
  if narg<5
    error('Not enough input arguments.');
  end
  Atrans = varargin{2};
  if ~isstr(Atrans)
    error('Afunc and Atransfunc must be names of m-files')
  end
  m = varargin{3};
  n = varargin{4};
```

```
  if ~isreal(n) | abs(fix(n)) ~= n | ~isreal(m) | abs(fix(m)) ~= m
    error('M and N must be positive integers.')
  end
  k=varargin{5};
  if narg < 6, p = rand(m,1)-0.5; else p=varargin{6}; end
  if narg < 7, options = []; else options=varargin{7}; end
  if narg > 7
    if  narg < 10
      error('All or none of U_old, B_old and V_old must be provided.')
    else
      U = varargin{8}; B_k = varargin{9}; V = varargin{10};
    end
  else
    U = []; B_k = []; V=[];
  end
  if narg > 10, anorm=varargin{11}; else anorm = [];  end
end

% Quick return for min(m,n) equal to 0 or 1.
if min(m,n) == 0
   U = [];  B_k = [];  V = [];  p = [];  ierr = 0;  work = zeros(2,2);
   return
elseif  min(m,n) == 1
  if isnumeric(A)
    U = 1;  B_k = A;  V = 1;  p = 0; ierr = 0; work = zeros(2,2);
  else
    U = 1;  B_k = feval(A,1); V = 1; p = 0; ierr = 0; work = zeros(2,2
```

```
  end
  if nargout<3
    U = B_k;
  end
  return
end

% Set options.
%m2 = 3/2*(sqrt(m)+1);
%n2 = 3/2*(sqrt(n)+1);
m2 = 3/2;
n2 = 3/2;
delta = sqrt(eps/k); % Desired level of orthogonality.
eta = eps^(3/4)/sqrt(k);     % Level of orth. after reorthogonalization
cgs = 0;                % Flag for switching between iterated MGS and CGS
elr = 2;                % Flag for switching extended local
                        % reorthogonalization on and off.
gamma = 1/sqrt(2);    % Tolerance for iterated Gram-Schmidt.
onesided = 0; t = 0; waitb = 0;

% Parse options struct
if ~isempty(options) & isstruct(options)
  c = fieldnames(options);
  for i=1:length(c)
    if strmatch(c(i),'delta'), delta = getfield(options,'delta');
end
    if strmatch(c(i),'eta'), eta = getfield(options,'eta'); end
```

```
    if strmatch(c(i),'cgs'), cgs = getfield(options,'cgs'); end
    if strmatch(c(i),'elr'), elr = getfield(options,'elr'); end
    if strmatch(c(i),'gamma'), gamma = getfield(options,'gamma'); end
    if strmatch(c(i),'onesided'), onesided = getfield(options,'onesided
    if strmatch(c(i),'waitbar'), waitb=1; end
  end
end

if waitb
  waitbarh = waitbar(0,'Lanczos bidiagonalization in progress...');
end

if isempty(anorm)
  anorm = []; est_anorm=1;
else
  est_anorm=0;
end

% Conservative statistical estimate on the size of round-off terms.
% Notice that {\bf u} == eps/2.
FUDGE = 1.01; % Fudge factor for ||A||_2 estimate.

npu = 0; npv = 0; ierr = 0;
p = p(:);
% Prepare for Lanczos iteration.
if isempty(U)
  V = zeros(n,k); U = zeros(m,k);
```

```
  beta = zeros(k+1,1); alpha = zeros(k,1);
  beta(1) = norm(p);
  % Initialize MU/NU-recurrences for monitoring loss of orthogonality.
  nu = zeros(k,1); mu = zeros(k+1,1);
  mu(1)=1; nu(1)=1;

  numax = zeros(k,1); mumax = zeros(k,1);
  force_reorth = 0;  nreorthu = 0; nreorthv = 0;
  j0 = 1;
else
  j = size(U,2); % Size of existing factorization
  % Allocate space for Lanczos vectors
  U = [U, zeros(m,k-j)];
  V = [V, zeros(n,k-j)];
  alpha = zeros(k+1,1);  beta = zeros(k+1,1);
  alpha(1:j) = diag(B_k); if j>1 beta(2:j) = diag(B_k,-1); end
  beta(j+1) = norm(p);
  % Reorthogonalize p.
  if j<k & beta(j+1)*delta < anorm*eps,
    fro = 1;
    ierr = j;
  end
  int = [1:j]';
  [p,beta(j+1),rr] = reorth(U,p,beta(j+1),int,gamma,cgs);
  npu =  rr*j;  nreorthu = 1;  force_reorth= 1;

  % Compute Gerscgorin bound on ||B_k||_2
```

```
    if est_anorm
      anorm = FUDGE*sqrt(norm(B_k'*B_k,1));
    end
    mu = m2*eps*ones(k+1,1); nu = zeros(k,1);
    numax = zeros(k,1); mumax = zeros(k,1);
    force_reorth = 1;  nreorthu = 0; nreorthv = 0;
    j0 = j+1;
end



if isnumeric(A)
    At = A';
end

if delta==0
    fro = 1; % The user has requested full reorthogonalization.
else
    fro = 0;
end

if LANBPRO_TRUTH==1
    MUTRUE = zeros(k,k); NUTRUE = zeros(k-1,k-1);
    MU = zeros(k,k); NU = zeros(k-1,k-1);

    MUTRUE_AFTER = zeros(k,k); NUTRUE_AFTER = zeros(k-1,k-1);
    MU_AFTER = zeros(k,k); NU_AFTER = zeros(k-1,k-1);
end
```

```
% Perform Lanczos bidiagonalization with partial reorthogonalization.
for j=j0:k
  if waitb
    waitbar(j/k,waitbarh)
  end

  if beta(j) ~= 0
    U(:,j) = p/beta(j);
  else
    U(:,j) = p;
  end

  % Replace norm estimate with largest Ritz value.
  if j==6
    B = [[diag(alpha(1:j-1))+diag(beta(2:j-1),-1)]; ...
      [zeros(1,j-2),beta(j)]];
    anorm = FUDGE*norm(B);
    est_anorm = 0;
  end

  %%%%%%%%%% Lanczos step to generate v_j. %%%%%%%%%%%%%%
  if j==1
    if isnumeric(A)
      r = At*U(:,1);
    elseif isstruct(A)
      r = A.R\U(:,1);
```

```
    else
      r = feval(Atrans,U(:,1));
    end
    alpha(1) = norm(r);
    if est_anorm
      anorm = FUDGE*alpha(1);
    end
  else
    if isnumeric(A)
      r = At*U(:,j) - beta(j)*V(:,j-1);
    elseif isstruct(A)
      r = A.R\U(:,j) - beta(j)*V(:,j-1);
    else
      r = feval(Atrans,U(:,j))  - beta(j)*V(:,j-1);
    end
    alpha(j) = norm(r);

    % Extended local reorthogonalization
    if alpha(j)<gamma*beta(j) & elr & ~fro
      normold = alpha(j);
      stop = 0;
      while ~stop
        t = V(:,j-1)'*r;
        r = r - V(:,j-1)*t;
        alpha(j) = norm(r);
        if beta(j) ~= 0
          beta(j) = beta(j) + t;
```

```
      end
      if alpha(j)>=gamma*normold
        stop = 1;
      else
        normold = alpha(j);
      end
    end
  end

  if est_anorm
    if j==2
      anorm = max(anorm,FUDGE*sqrt(alpha(1)^2+beta(2)^2+alpha(2)*beta
    else
      anorm = max(anorm,FUDGE*sqrt(alpha(j-1)^2+beta(j)^2+alpha(j-1)
          beta(j-1) + alpha(j)*beta(j)));
    end
  end

  if ~fro & alpha(j) ~= 0
    % Update estimates of the level of orthogonality for the
    %  columns 1 through j-1 in V.
    nu = update_nu(nu,mu,j,alpha,beta,anorm);
    numax(j) = max(abs(nu(1:j-1)));
  end

  if j>1 & LANBPRO_TRUTH
    NU(1:j-1,j-1) = nu(1:j-1);
```

```
  NUTRUE(1:j-1,j-1) = V(:,1:j-1)'*r/alpha(j);
end

if elr>0
  nu(j-1) = n2*eps;
end

% IF level of orthogonality is worse than delta THEN
%    Reorthogonalize v_j against some previous  v_i's, 0<=i<j.
if onesided~=-1 & ( fro | numax(j) > delta | force_reorth ) & alpha
  % Decide which vectors to orthogonalize against:
  if fro | eta==0
    int = [1:j-1]';
  elseif force_reorth==0
    int = compute_int(nu,j-1,delta,eta,0,0,0);
  end
  % Else use int from last reorth. to avoid spillover from mu_{j-1
  % to nu_j.

  % Reorthogonalize v_j
  [r,alpha(j),rr] = reorth(V,r,alpha(j),int,gamma,cgs);
  npv = npv + rr*length(int); % number of inner products.
  nu(int) = n2*eps;  % Reset nu for orthogonalized vectors.

  % If necessary force reorthogonalization of u_{j+1}
  % to avoid spillover
  if force_reorth==0
```

```
      force_reorth = 1;
    else
      force_reorth = 0;
    end
    nreorthv = nreorthv + 1;
  end
end


% Check for convergence or failure to maintain semiorthogonality
if alpha(j) < max(n,m)*anorm*eps & j<k,
  % If alpha is "small" we deflate by setting it
  % to 0 and attempt to restart with a basis for a new
  % invariant subspace by replacing r with a random starting vector:
  %j
  %disp('restarting, alpha = 0')
  alpha(j) = 0;
  bailout = 1;
  for attempt=1:3
    r = rand(m,1)-0.5;
    if isnumeric(A)
      r = At*r;
    elseif isstruct(A)
      r = A.R\r;
    else
      r = feval(Atrans,r);
    end
```

```
      nrm=sqrt(r'*r); % not necessary to compute the norm accurately h
      int = [1:j-1]';
      [r,nrmnew,rr] = reorth(V,r,nrm,int,gamma,cgs);
      npv = npv + rr*length(int(:));          nreorthv = nreorthv + 1;
      nu(int) = n2*eps;
      if nrmnew > 0
        % A vector numerically orthogonal to span(Q_k(:,1:j)) was found
        % Continue iteration.
        bailout=0;
        break;
      end
    end
    if bailout
      j = j-1;
      ierr = -j;
      break;
    else
      r=r/nrmnew; % Continue with new normalized r as starting vector.
      force_reorth = 1;
      if delta>0
        fro = 0;    % Turn off full reorthogonalization.
      end
    end
  elseif  j<k & ~fro & anorm*eps > delta*alpha(j)
%    fro = 1;
    ierr = j;
  end
```

```
if j>1 & LANBPRO_TRUTH
  NU_AFTER(1:j-1,j-1) = nu(1:j-1);
  NUTRUE_AFTER(1:j-1,j-1) = V(:,1:j-1)'*r/alpha(j);
end


if alpha(j) ~= 0
  V(:,j) = r/alpha(j);
else
  V(:,j) = r;
end

%%%%%%%%% Lanczos step to generate u_{j+1}. %%%%%%%%%%%%%
if waitb
   waitbar((2*j+1)/(2*k),waitbarh)
end

if isnumeric(A)
  p = A*V(:,j) - alpha(j)*U(:,j);
elseif isstruct(A)
  p = A.Rt\V(:,j) - alpha(j)*U(:,j);
else
  p = feval(A,V(:,j)) - alpha(j)*U(:,j);
end
beta(j+1) = norm(p);
% Extended local reorthogonalization
```

```
if beta(j+1)<gamma*alpha(j) & elr & ~fro
  normold = beta(j+1);
  stop = 0;
  while ~stop
    t = U(:,j)'*p;
    p = p - U(:,j)*t;
    beta(j+1) = norm(p);
    if alpha(j) ~= 0
      alpha(j) = alpha(j) + t;
    end
    if beta(j+1) >= gamma*normold
      stop = 1;
    else
      normold = beta(j+1);
    end
  end
end

if est_anorm
  % We should update estimate of ||A||  before updating mu - especia
  % important in the first step for problems with large norm since a
  % may be a severe underestimate!
  if j==1
    anorm = max(anorm,FUDGE*pythag(alpha(1),beta(2)));
  else
    anorm = max(anorm,FUDGE*sqrt(alpha(j)^2+beta(j+1)^2 + alpha(j)*b
  end
```

```
end


if ~fro & beta(j+1) ~= 0
  % Update estimates of the level of orthogonality for the columns o
  mu = update_mu(mu,nu,j,alpha,beta,anorm);
  mumax(j) = max(abs(mu(1:j)));
end

if LANBPRO_TRUTH==1
  MU(1:j,j) = mu(1:j);
  MUTRUE(1:j,j) = U(:,1:j)'*p/beta(j+1);
end

if elr>0
  mu(j) = m2*eps;
end

% IF level of orthogonality is worse than delta THEN
%    Reorthogonalize u_{j+1} against some previous  u_i's, 0<=i<=j.
if onesided~=1 & (fro | mumax(j) > delta | force_reorth) & beta(j+1)
  % Decide which vectors to orthogonalize against.
  if fro | eta==0
    int = [1:j]';
  elseif force_reorth==0
    int = compute_int(mu,j,delta,eta,0,0,0);
  else
```

```
      int = [int; max(int)+1];
    end
    % Else use int from last reorth. to avoid spillover from nu to mu.

%     if onesided~=0
%       fprintf('i = %i, nr = %i, fro = %i\n',j,size(int(:),1),fro)
%     end
    % Reorthogonalize u_{j+1}
    [p,beta(j+1),rr] = reorth(U,p,beta(j+1),int,gamma,cgs);
    npu = npu + rr*length(int);  nreorthu = nreorthu + 1;

    % Reset mu to epsilon.
    mu(int) = m2*eps;

    if force_reorth==0
      force_reorth = 1; % Force reorthogonalization of v_{j+1}.
    else
      force_reorth = 0;
    end
  end

  % Check for convergence or failure to maintain semiorthogonality
  if beta(j+1) < max(m,n)*anorm*eps  & j<k,
    % If beta is "small" we deflate by setting it
    % to 0 and attempt to restart with a basis for a new
    % invariant subspace by replacing p with a random starting vector:
    %j
```

```
%disp('restarting, beta = 0')
beta(j+1) = 0;
bailout = 1;
for attempt=1:3
  p = rand(n,1)-0.5;
  if isnumeric(A)
    p = A*p;
  elseif isstruct(A)
    p = A.Rt\p;
  else
    p = feval(A,p);
  end
  nrm=sqrt(p'*p); % not necessary to compute the norm accurately h
  int = [1:j]';
  [p,nrmnew,rr] = reorth(U,p,nrm,int,gamma,cgs);
  npu = npu + rr*length(int(:));  nreorthu = nreorthu + 1;
  mu(int) = m2*eps;
  if nrmnew > 0
    % A vector numerically orthogonal to span(Q_k(:,1:j)) was found
    % Continue iteration.
    bailout=0;
    break;
  end
end
if bailout
  ierr = -j;
  break;
```

```
      else
        p=p/nrmnew; % Continue with new normalized p as starting vector.
        force_reorth = 1;
        if delta>0
          fro = 0;     % Turn off full reorthogonalization.
        end
      end
    elseif  j<k & ~fro & anorm*eps > delta*beta(j+1)
%     fro = 1;
      ierr = j;
    end

    if LANBPRO_TRUTH==1
      MU_AFTER(1:j,j) = mu(1:j);
      MUTRUE_AFTER(1:j,j) = U(:,1:j)'*p/beta(j+1);
    end
  end
  if waitb
    close(waitbarh)
  end

  if j<k
    k = j;
  end

  B_k = spdiags([alpha(1:k) [beta(2:k);0]],[0 -1],k,k);
  if nargout==1
```

```
  U = B_k;
elseif k~=size(U,2) | k~=size(V,2)
  U = U(:,1:k);
  V = V(:,1:k);
end
if nargout>5
  work = [[nreorthu,npu];[nreorthv,npv]];
end



function mu = update_mu(muold,nu,j,alpha,beta,anorm)

% UPDATE_MU:  Update the mu-recurrence for the u-vectors.
%
%    mu_new = update_mu(mu,nu,j,alpha,beta,anorm)

binv = 1/beta(j+1);
mu = muold;
eps1 = 100*eps/2;
if j==1
  T = eps1*(pythag(alpha(1),beta(2)) + pythag(alpha(1),beta(1)));
  T = T + eps1*anorm;
  mu(1) = T / beta(2);
else
  mu(1) = alpha(1)*nu(1) - alpha(j)*mu(1);
%  T = eps1*(pythag(alpha(j),beta(j+1)) + pythag(alpha(1),beta(1)));
```

```
    T = eps1*(sqrt(alpha(j).^2+beta(j+1).^2) + sqrt(alpha(1).^2+beta(1).
    T = T + eps1*anorm;
    mu(1) = (mu(1) + sign(mu(1))*T) / beta(j+1);
    % Vectorized version of loop:
    if j>2
      k=2:j-1;
      mu(k) = alpha(k).*nu(k) + beta(k).*nu(k-1) - alpha(j)*mu(k);
      %T = eps1*(pythag(alpha(j),beta(j+1)) + pythag(alpha(k),beta(k)));
      T = eps1*(sqrt(alpha(j).^2+beta(j+1).^2) + sqrt(alpha(k).^2+beta(k
      T = T + eps1*anorm;
      mu(k) = binv*(mu(k) + sign(mu(k)).*T);
    end
%   T = eps1*(pythag(alpha(j),beta(j+1)) + pythag(alpha(j),beta(j)));
    T = eps1*(sqrt(alpha(j).^2+beta(j+1).^2) + sqrt(alpha(j).^2+beta(j).
    T = T + eps1*anorm;
    mu(j) = beta(j)*nu(j-1);
    mu(j) = (mu(j) + sign(mu(j))*T) / beta(j+1);
end
mu(j+1) = 1;


function nu = update_nu(nuold,mu,j,alpha,beta,anorm)

% UPDATE_MU:  Update the nu-recurrence for the v-vectors.
%
%  nu_new = update_nu(nu,mu,j,alpha,beta,anorm)
```

```
nu = nuold;
ainv = 1/alpha(j);
eps1 = 100*eps/2;
if j>1
  k = 1:(j-1);
%  T = eps1*(pythag(alpha(k),beta(k+1)) + pythag(alpha(j),beta(j)));
  T = eps1*(sqrt(alpha(k).^2+beta(k+1).^2) + sqrt(alpha(j).^2+beta(j).
  T = T + eps1*anorm;
  nu(k) = beta(k+1).*mu(k+1) + alpha(k).*mu(k) - beta(j)*nu(k);
  nu(k) = ainv*(nu(k) + sign(nu(k)).*T);
end
nu(j) = 1;

function x = pythag(y,z)
%PYTHAG Computes sqrt( y^2 + z^2 ).
%
% x = pythag(y,z)
%
% Returns sqrt(y^2 + z^2) but is careful to scale to avoid overflow.

[m n] = size(y);
if m>1 | n>1
  y = y(:); z=z(:);
  rmax = max(abs([y z]'))';
  id=find(rmax==0);
  if length(id)>0
    rmax(id) = 1;
```

```
      x = rmax.*sqrt((y./rmax).^2 + (z./rmax).^2);
      x(id)=0;
    else
      x = rmax.*sqrt((y./rmax).^2 + (z./rmax).^2);
    end
    x = reshape(x,m,n);
  else
    rmax = max(abs([y;z]));
    if (rmax==0)
      x = 0;
    else
      x = rmax*sqrt((y/rmax)^2 + (z/rmax)^2);
    end
  end
```

BIBLIOGRAPHY

[1] Laura Balzano, Robert Nowak, and Benjamin Recht. Online identification and tracking of subspaces from highly incomplete information. In *Communication, Control, and Computing (Allerton), 2010 48th Annual Allerton Conference on*, pages 704–711. IEEE, 2010.

[2] James Bennett, Stan Lanning, et al. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35. New York, NY, USA, 2007.

[3] Brian Eriksson, Laura Balzano, and Robert D Nowak. High-rank matrix completion. In *AISTATS*, pages 373–381, 2012.

[4] Nathalie Henry and Jean-Daniel Fekete. Matlink: Enhanced matrix visualization for analyzing social networks. *Human-Computer Interaction–INTERACT 2007*, pages 288–302, 2007.

[5] Hui Ji, Chaoqiang Liu, Zuowei Shen, and Yuhong Xu. Robust video denoising using low rank matrix completion. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1791–1798. IEEE, 2010.

[6] Robert D Leclerc. Survival of the sparsest: robust gene networks are parsimonious. *Molecular systems biology*, 4(1):213, 2008.

[7] Qing Liao, Naiyang Guan, Chengkun Wu, and Qian Zhang. Predicting unknown interactions between known drugs and targets via matrix completion. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 591–604. Springer, 2016.

[8] Zhouchen Lin, Minming Chen, and Yi Ma. The augmented lagrange multiplier method for exact recovery of corrupted low-rank matrices. *arXiv preprint arXiv:1009.5055*, 2010.

[9] Peter J Pahl and Rudolf Damrath. *Mathematical foundations of computational engineering: a handbook*. Springer Science & Business Media, 2001.

[10] Turki Turki, William Bassett, and Jason TL Wang. A learning framework to improve unsupervised gene network inference. In *Machine Learning and Data Mining in Pattern Recognition*, pages 28–42. Springer, 2016.

[11] Turki Turki and Jason TL Wang. A new approach to link prediction in gene regulatory networks. In *International Conference on Intelligent Data Engineering and Automated Learning*, pages 404–415. Springer, 2015.

[12] Turki Turki, Jason TL Wang, and Ibrahim Rajikhan. Inferring gene regulatory networks by combining supervised and unsupervised methods. In *Machine*

Learning and Applications (ICMLA), 2016 15th IEEE International Conference on, pages 140–145. IEEE, 2016.