Summer 2017

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

Yiran Wang
New Jersey Institute of Technology

Follow this and additional works at: https://digitalcommons.njit.edu/theses
Part of the Computer Sciences Commons

Recommended Citation
Wang, Yiran, "Matrix completion algorithms with applications in biomedicine, e-commerce and social science" (2017). Theses. 37.
https://digitalcommons.njit.edu/theses/37

This Thesis is brought to you for free and open access by the Theses and Dissertations at Digital Commons @ NJIT. It has been accepted for inclusion in Theses by an authorized administrator of Digital Commons @ NJIT. For more information, please contact digitalcommons@njit.edu.
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.
The Van Houten library has removed some of the personal information and all signatures from the approval page and biographical sketches of theses and dissertations in order to protect the identity of NJIT graduates and faculty.
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
Professor of Bioinformatics and Computer Science, NJIT

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

Dr. Chase Qishi Wu, Committee Member
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:

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

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>2 LOW RANK MATRIX COMPLETIONS</td>
<td>3</td>
</tr>
<tr>
<td>2.1 GROUSE Algorithm</td>
<td>3</td>
</tr>
<tr>
<td>2.1.1 Introduction of GROUSE algorithm and Implementation</td>
<td>3</td>
</tr>
<tr>
<td>2.1.2 Test and Result</td>
<td>4</td>
</tr>
<tr>
<td>2.2 ALM Method</td>
<td>10</td>
</tr>
<tr>
<td>2.2.1 Introduction of ALM Method</td>
<td>10</td>
</tr>
<tr>
<td>2.2.2 Test and Result</td>
<td>10</td>
</tr>
<tr>
<td>3 FEASIBILITY OF HIGH RANK MATRIX COMPLETION APPLIED ON BIOMEDICINE</td>
<td>16</td>
</tr>
<tr>
<td>3.1 Materials and Methods</td>
<td>16</td>
</tr>
<tr>
<td>3.1.1 Interactions between Transcription Factors and Target Genes</td>
<td>16</td>
</tr>
<tr>
<td>3.1.2 Basic Assumptions</td>
<td>17</td>
</tr>
<tr>
<td>3.1.3 The Proposed Algorithm</td>
<td>18</td>
</tr>
<tr>
<td>3.2 Result</td>
<td>21</td>
</tr>
<tr>
<td>4 CONCLUSION</td>
<td>26</td>
</tr>
<tr>
<td>APPENDIX A PROGRAM &amp; DATASETS</td>
<td>27</td>
</tr>
<tr>
<td>BIBLIOGRAPHY</td>
<td>63</td>
</tr>
<tr>
<td>Table</td>
<td>Page</td>
</tr>
<tr>
<td>---------------</td>
<td>------</td>
</tr>
<tr>
<td>2.1</td>
<td>4</td>
</tr>
<tr>
<td>2.2</td>
<td>4</td>
</tr>
<tr>
<td>2.3</td>
<td>4</td>
</tr>
<tr>
<td>2.4</td>
<td>4</td>
</tr>
<tr>
<td>3.1</td>
<td>21</td>
</tr>
<tr>
<td>3.2</td>
<td>21</td>
</tr>
<tr>
<td>3.3</td>
<td>22</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>GROUSE result on synthetic data 1.</td>
<td>5</td>
</tr>
<tr>
<td>2.2</td>
<td>GROUSE result on synthetic data 2.</td>
<td>6</td>
</tr>
<tr>
<td>2.3</td>
<td>GROUSE result on synthetic data 3.</td>
<td>7</td>
</tr>
<tr>
<td>2.4</td>
<td>GROUSE result on synthetic data 4.</td>
<td>8</td>
</tr>
<tr>
<td>2.5</td>
<td>GROUSE comparison result of low rank data.</td>
<td>9</td>
</tr>
<tr>
<td>2.6</td>
<td>Exact ALM result on synthetic data 1.</td>
<td>11</td>
</tr>
<tr>
<td>2.7</td>
<td>Exact ALM result on synthetic data 2.</td>
<td>12</td>
</tr>
<tr>
<td>2.8</td>
<td>Exact ALM result on synthetic data 3.</td>
<td>13</td>
</tr>
<tr>
<td>2.9</td>
<td>Exact ALM result on synthetic data 4.</td>
<td>14</td>
</tr>
<tr>
<td>2.10</td>
<td>Exact ALM comparison result of low rank data.</td>
<td>15</td>
</tr>
<tr>
<td>3.1</td>
<td>Result on synthetic data.</td>
<td>22</td>
</tr>
<tr>
<td>3.2</td>
<td>Result on mouse data.</td>
<td>23</td>
</tr>
<tr>
<td>3.3</td>
<td>Result on human data.</td>
<td>24</td>
</tr>
<tr>
<td>3.4</td>
<td>Comparison of 3 results.</td>
<td>25</td>
</tr>
</tbody>
</table>
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 $\Omega_t$. A set of stepsizes $\eta_t$

Output: output ($\sigma$)

1. for $t = 1, 2, 3, ..., T$ do
2. Estimate weights $w = \arg\min_a \| \Delta_{\Omega_t}(U_t a - v_t) \|
3. Predict full vectors $p = U_t w$
4. Compute residual $r = \Delta_{\Omega_t}(v_t - p)$
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

<table>
<thead>
<tr>
<th>Columns</th>
<th>Rows</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 2.2: Synthetic data of low rank matrix 2

<table>
<thead>
<tr>
<th>Columns</th>
<th>Rows</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>20</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 2.3: Synthetic data of low rank matrix 3

<table>
<thead>
<tr>
<th>Columns</th>
<th>Rows</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
<td>7</td>
</tr>
</tbody>
</table>

Table 2.4: Synthetic data of low rank matrix 4

<table>
<thead>
<tr>
<th>Columns</th>
<th>Rows</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>20</td>
<td>7</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Input: Observation Matrix $D \in \mathbb{R}^{m \times n}$, $\lambda$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output: output $(A_k, E_k)$</td>
</tr>
</tbody>
</table>

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

2. **while** not converged **do**
   
3. $A^0_{k+1} = A^*_k, E^0_{k+1} = E^*_k, j = 0$
   
4. **while** not converged **do**
   
5. $(U, S, V) = \text{svd}(D - E^{j}_{k+1} + \mu^{-1}_k Y^*_k)$
   
6. $A^{j+1}_{k+1} = US_{\mu^{-1}_k}[S]V^T$
   
7. $E^{j+1}_{k+1} = S_{A_{\mu^{-1}_k}}[D - A^{j+1}_{k+1} + \mu^{-1}_k Y^*_k]$  
   
8. **end while**
   
9. $Y_{k+1} = Y_k + \mu_k(N_j - A_{k+1} - E_{k+1}); \mu_{k+1} = \rho \mu_k$
   
10. $k = k + 1$

11. **end while**

12. **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 \( o', v', \ast', \cdot' \) 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 row 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 $X \in \mathbb{R}^{N \times n}$ 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.

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

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

16
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.

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

We assume that each complete vector lies in a d-dimensional subspace, where $d \leq n$. 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.

$$
X_{predict}(r, c) = \begin{cases} 
-1 & \text{if more negative values} \\
1 & \text{if more positive values}
\end{cases}
$$ (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 $S \subseteq \mathbb{R}^n$ is defined as,

$$\mu(S) := \frac{n}{r} \max_j \|P_se_j\|_2^2$$  \hspace{1cm} (3.4)$$

where $P_s$ is the projection operator onto $S$ and $e_j$ are the canonical vectors for $\mathbb{R}^n$. 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 $\mu_0$.

- If $\text{rank}(S_i) = r_i$, then any subset of $r_i$ columns from $S_i$ can spans $S_i$. Let $0 < \epsilon < 1$ and $S_{i,\epsilon_0}$ denote the subset of points in $S_i$ at least $\epsilon_0$ distance away from any other space. There exist a constant $0 < V_0 \leq 1$, depending on $\epsilon_0$, such that

  1. The probability that a column selected uniformly at random belongs to $S_{i,\epsilon_0}$ 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 $\epsilon_0$ centered at $x$ is at least $\delta \epsilon_0^r / k$.

### 3.1.3 The Proposed Algorithm

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

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

$$\delta^0 = n^{2-2\beta/2} \log n$$  \hspace{1cm} (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,

$$t_0 = \left\lceil 2\mu_0^2 \log(2S_0l_0n/\delta_0) \right\rceil$$

(3.7)

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

(3.8)

Where $0 < \epsilon < 1$. Now we have $S_0$ seeds with their corresponding set. Randomly select $l_0n$ 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 $\epsilon$ 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 $N_j$, $\lambda$

**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,

$$\bar{x} = N(N^TN)^\dagger N^Tx$$  \hspace{1cm} (3.9)

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

SVD of A:  \quad A = U \begin{pmatrix} S & 0 \\ 0 & 0 \end{pmatrix} V^T  \hspace{1cm} (3.10)

pseudo-inverse of A:  \quad A = V \begin{pmatrix} S^{-1} & 0 \\ 0 & 0 \end{pmatrix} U^T  \hspace{1cm} (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>
<thead>
<tr>
<th>Gene related factors</th>
<th>number of interactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>5</td>
</tr>
</tbody>
</table>

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 matrix completion applied on those data.

<table>
<thead>
<tr>
<th>Gene</th>
<th>transcription factors</th>
<th>number of interactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>5</td>
<td>78</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Gene</th>
<th>transcription factors</th>
<th>number of interactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>5</td>
<td>68</td>
</tr>
</tbody>
</table>
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.
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 outer = 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;
\[ U = U + \text{step} \times \text{weights} \];

end

end

end

\[ R = \text{zeros}(\text{numc}, \text{maxrank}); \]

for \( k = 1: \text{numc} \),

\[ \text{idx} = \text{find}(\text{Indicator}(:, k)); \]
\[ v_{\text{Omega}} = \text{values} (\text{idx}, k); \]
\[ U_{\text{Omega}} = U(\text{idx}, :); \]
\[ R(k, :) = (U_{\text{Omega}} \backslash v_{\text{Omega}})' ; \]
end

\[ X = U \times R'; \]

function [A iter svp] = fast_alm_mc(D, tol, maxIter)
% This matlab code implements the inexact augmented Lagrange multiplier
% method for Matrix Completion.
%
% \( D - m \times n \) matrix of observations/data (required input)
%
% \( tol \) - tolerance for stopping criterion.
% \( - \text{DEFAULT} \, 1e^{-7} \) if omitted or \(-1\).
%
% \( \text{maxIter} \) - maximum number of iterations
% \( - \text{DEFAULT} \, 1000 \), if omitted or \(-1\).
%
% Model:
\[
\begin{align*}
\text{clear global;} \\
global A \text{ Sparse } Z; \\
\text{addpath PROPACK;} \\
[m \ n] = \text{size}(D); \\
\text{if nargin < 2} \\
\text{tol} = 1e^{-4}; \\
\text{elseif} \ \text{tol} == -1 \\
\text{tol} = 1e^{-4}; \\
\text{end} \\
\text{if nargin < 3} \\
\text{maxIter = 1000;} \\
\text{elseif maxIter == -1} \\
\text{maxIter = 1000;} \\
\text{end} \\
\text{read sparse matrix D} \\
[m \ n] = \text{size}(D); \\
[I \ J \ V] = \text{find}(D); \\
p = \text{length}(I); \\
\text{col} = [0; \text{find(diff}(J)); \ p]; \\
\text{Sparse.Z} = D;
\end{align*}
\]
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;
    end
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(sqrt(ds));
A.V = A.V(:, 1:svp) * diag(sqrt(ds));

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) 'svn' num2str(stopCriterion) '
    mu' num2str(mu]);
end

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

%% update mu
\[ \mu = \mu \cdot \rho ; \]

\textbf{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++)

```c
{
    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])<<'\n';
    }
    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;
\[ d = 0 ; \]

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

\[
\text{function } y = \text{UVtOmega}(U, V, I, J, \text{col}) ;
\]
\[
y = \text{zeros}(\text{length}(I), 1) ;
\]
\[
\text{for } k = 1 : \text{length}(\text{col})-1
\]
\[
j = J(\text{col}(k)+1) ;
\]
\[ X_j = U \ast V(j,:)'; \]
\[ \text{idx} = [\text{col}(k)+1:\text{col}(k+1)]; \]
\[ y(\text{idx}) = X_j(I(\text{idx})); \]

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 = \text{min}(\text{round}(r \ast (2 \ast n - r) \ast \text{pdr}), m\ast n);
rho_s = p / (m \ast 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/ω|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
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 = varargin{6}; V = varargin{7};
    end
else
    U = []; B = []; 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
    end
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'); end
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 $\|b\|_u = \text{eps}/2$.
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);
\begin{verbatim}
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;
end

% 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);
umax = 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 = A*U(:,1);
        elseif isstruct(A)
            r = A.R\U(:,1);
        end
    else
        % Lanczos step code
    end
else
    \( r = \text{feval}(A_{\text{trans}}, U(:,1)) \);
end

alpha(1) = \text{norm}(r);
if est_anorm
    anorm = \text{FUDGE} \cdot \alpha(1);
end

else
    if isnumeric(A)
        \( r = A^* U(:,j) - \beta(j) \cdot V(:,j-1) \);
    elseif isstruct(A)
        \( r = A \cdot R \backslash U(:,j) - \beta(j) \cdot V(:,j-1) \);
    else
        \( r = \text{feval}(A_{\text{trans}}, U(:,j)) - \beta(j) \cdot V(:,j-1) \);
    end

alpha(j) = \text{norm}(r);

% Extended local reorthogonalization
if alpha(j) < \text{gamma} \cdot beta(j) \& elr \& ~fro
    normold = alpha(j);
    stop = 0;
    while ~stop
        t = V(:,j-1)' \cdot r;
        \( r = r - V(:,j-1) \cdot t \);
        alpha(j) = \text{norm}(r);
        if beta(j) \neq 0
            beta(j) = beta(j) + t;
        end
    end

end
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(2)));
    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, nu, 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);
\[
\text{NUTRUE}(1:j-1,j-1) = V(:,1:j-1)' \ast r / \alpha(j);
\]

end

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

% IF level of orthogonality is worse than \text{delta} THEN
% Reorthogonalize \text{v}_j against some previous \text{v}_i's, 0 \leq i < j.
if onesided \sim = -1 & ( \text{fro} | \text{numax}(j) > \text{delta} | \text{force_reorth} ) & \alpha(j) \sim = 0
    % Decide which vectors to orthogonalize against:
    if \text{fro} | \text{eta} = 0
        int = [1:j-1]';
    elseif \text{force_reorth} = 0
        int = \text{compute_int}(\text{nu}, j-1, \text{delta}, \text{eta}, 0, 0, 0);
    end
    % Else use int from last reorth. to avoid spillover from \text{mu}_{j-1} to \text{nu}_j.
    % Reorthogonalize \text{v}_j
    [r, \alpha(j), rr] = \text{reorth}(V, r, \alpha(j), \text{int}, \text{gamma}, \text{cgs});
    npv = npv + rr*\text{length}(\text{int}); % number of inner products.
    nu(int) = n2*eps; % Reset \text{nu} for orthogonalized vectors.

% If necessary force reorthogonalization of \text{u}_{j+1}
% to avoid spillover
if force_reorth = 0

50
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
\[ nrm = \sqrt{r' \cdot r}; \]  % not necessary to compute the norm accurately here.

\[ \text{int} = [1:j-1]'; \]

\[ [r, nrmnew, rr] = \text{reorth}(V, r, nrm, \text{int}, \gamma, \text{cgs}); \]

\[ \text{nrv} = \text{nrv} + rr \cdot \text{length(int(:))}; \]

\[ \text{nreorthv} = \text{nreorthv} + 1; \]

\[ \text{nu(int)} = n2 \cdot \text{eps}; \]

\[ \text{if } nrmnew > 0 \]

\[ % A vector numerically orthogonal to \text{span}(Q_{k(:1:j)}) \text{ was found.} \]

\[ % \text{Continue iteration.} \]

\[ \text{bailout}=0; \]

\[ \text{break;} \]

\[ \text{end} \]

\[ \text{end} \]

\[ \text{if } \text{bailout} \]

\[ j = j-1; \]

\[ \text{ierr} = -j; \]

\[ \text{break;} \]

\[ \text{else} \]

\[ r = r/nrmnew; % \text{Continue with new normalized } r \text{ as starting vector.} \]

\[ \text{force_reorth} = 1; \]

\[ \text{if } \text{delta}>0 \]

\[ \text{fro} = 0; % \text{Turn off full reorthogonalization.} \]

\[ \text{end} \]

\[ \text{end} \]

\[ \text{elseif } j<k \& \sim \text{fro} \& \text{anorm*eps} > \text{delta*alpha(j)} \]

\[ % \text{fro} = 1; \]

\[ \text{ierr} = j; \]

\[ \text{end} \]
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 \& \text{fro}

\text{normold} = \text{beta}(j+1);
\text{stop} = 0;
\text{while } \sim \text{stop}
\text{t} = U(:,j)^*p;
\text{p} = p - U(:,j)*t;
\text{beta}(j+1) = \text{norm}(p);
\text{if } \text{alpha}(j) \sim 0
\text{alpha}(j) = \text{alpha}(j) + t;
\text{end}
\text{if } \text{beta}(j+1) \geq \gamma*\text{normold}
\text{stop} = 1;
\text{else}
\text{normold} = \text{beta}(j+1);
\text{end}
\text{end}
\text{end}

if \text{est_anorm}
\% We should update estimate of \|A\| before updating mu – especially
\% important in the first step for problems with large norm since a
\% may be a severe underestimate!
\text{if } j==1
\text{anorm} = \text{max}(\text{anorm},FUDGE*\text{pythag} (\text{alpha}(1), \text{beta}(2)));
\text{else}
\text{anorm} = \text{max}(\text{anorm},FUDGE*\text{sqrt}(\text{alpha}(j)^2+\text{beta}(j+1)^2 + \text{alpha}(j)*\text{beta}(j)));
\text{end}
if ~fro & beta(j+1) ~= 0

% Update estimates of the level of orthogonality for the columns of V.
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
end

55
\begin{verbatim}
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
',j, size(int(:,1),size(int))
% 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
\end{verbatim}
%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 here
    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;
end
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
\[ U = B_k; \]
\[
\text{elseif } k^* = \text{size}(U,2) \quad \text{or} \quad k^* = \text{size}(V,2) \\
U = U(:,1:k); \\
V = V(:,1:k); \\
\]
\[
\text{end} \\
\text{if nargout}>5 \\
\text{work} = [[\text{nreorthu},\text{npu}];[\text{nreorthv},\text{npv}]]; \\
\text{end} \\
\]

\[
\text{function } \text{mu} = \text{update}\_\text{mu}(\text{muold},\text{nu},j,\text{alpha},\text{beta},\text{anorm}) \\
\]
\[
\textbf{function} \quad \mu = \text{update}\_\text{mu}(\mu, \nu, j, \alpha, \beta, \text{anorm}) \\
\]
\[
\% \text{UPDATE}\_\text{MU}: \quad \text{Update the } \mu-\text{recurrence for the } u-\text{vectors}. \\
\%
\%
\mu_\text{new} = \text{update}\_\text{mu}(\mu, \nu, j, \alpha, \beta, \text{anorm}) \\
\]
\[
\quad \text{binv} = 1/\beta(j+1); \\
\quad \mu = \mu_{\text{old}}; \\
\quad \eps1 = 100*\text{eps}/2; \\
\text{if } j==1 \\
\qquad T = \eps1*(\text{pythag}(\alpha(1), \beta(2)) + \text{pythag}(\alpha(1), \beta(1))); \\
\qquad T = T + \eps1*\text{anorm}; \\
\qquad \mu(1) = T / \beta(2); \\
\text{else} \\
\qquad \mu(1) = \alpha(1)*\nu(1) - \alpha(j)*\mu(1); \\
\qquad \% \quad T = \eps1*(\text{pythag}(\alpha(j), \beta(j+1)) + \text{pythag}(\alpha(1), \beta(1)))); \\
\]
\begin{align*}
T &= \epsilon_1 \cdot (\sqrt{(\alpha(j)^2 + \beta(j+1)^2}) + \sqrt{(\alpha(1)^2 + \beta(1)^2)}; \\
T &= T + \epsilon_1 \cdot \text{anorm}; \\
\mu(1) &= (\mu(1) + \text{sign}(\mu(1)) \cdot T) / \beta(j+1); \\
\text{% Vectorized version of loop:} \\
\text{if } j > 2 \\
\quad k &= 2:j-1; \\
\quad \mu(k) &= \alpha(k) \cdot \nu(k) + \beta(k) \cdot \nu(k-1) - \alpha(j) \cdot \mu(k); \\
\quad %T &= \epsilon_1 \cdot (\text{pythag}(\alpha(j), \beta(j+1)) + \text{pythag}(\alpha(k), \beta(k))); \\
\quad T &= \epsilon_1 \cdot (\sqrt{(\alpha(j)^2 + \beta(j+1)^2}) + \sqrt{(\alpha(k)^2 + \beta(k)^2)}; \\
T &= T + \epsilon_1 \cdot \text{anorm}; \\
\quad \mu(k) &= \text{binv} \cdot (\mu(k) + \text{sign}(\mu(k)) \cdot T); \\
\end{align*}

\begin{align*}
\text{mu}(j+1) &= 1; \\
\text{function } \nu = \text{update_nu}(\nuold, \mu, j, \alpha, \beta, \text{anorm}) \\
\text{% UPDATE\_MU: Update the \nu-recurrence for the \nu-vectors.} \\
\text{%} \\
\text{% } \nu_{\text{new}} = \text{update_nu}(\nu, \mu, j, \alpha, \beta, \text{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).^2));
    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;
    end
end
\[
x = \text{rmax} \times \text{sqrt} \left( \frac{y}{\text{rmax}} \right)^2 + \left( \frac{z}{\text{rmax}} \right)^2 \right);
\]
\[
x(id) = 0;
\]

else

\[
x = \text{rmax} \times \text{sqrt} \left( \frac{y}{\text{rmax}} \right)^2 + \left( \frac{z}{\text{rmax}} \right)^2 \right);
\]
end

\[
x = \text{reshape} (x, m, n);
\]

else

\[
rmax = \text{max} (\text{abs} ([y ; z]));
\]

if (rmax == 0)

\[
x = 0;
\]

else

\[
x = \text{rmax} \times \text{sqrt} \left( \frac{y}{\text{rmax}} \right)^2 + \left( \frac{z}{\text{rmax}} \right)^2 \right);
\]
end
end


