Modified proof, new algorithm simulation and ordinal tensor simulation

Chanwoo Lee

1 Accuracy proof for approximated SVD

Our goal is to recover tensor $\mathcal{A} = \mathcal{C} \times_1 M^{(1)} \times_2 M^{(2)} \times_3 M^{(3)}$ from a tensor $\mathcal{D} = \mathcal{A} + \mathcal{E}$ with a noise whose element is drawn from $N(0, \sigma^2)$ using following algorithms. To formulate the problem clear, we set dimension of $\mathcal{A} \in R^{d_1 \times d_2 \times d_3}$ and $\mathcal{C} \in R^{r_1 \times r_2 \times r_3}$

Algorithm 1 Approx tensor SVD 1

1: **procedure** SVD($\mathcal{A}$)
2: **Step A: Approximate SVD of $\mathcal{A}$**
3: **for** $n \leftarrow 1 : N$ **do**
4: Unfold $\mathcal{A}$ as $A_{(n)}$
5: Generate an $d_1 \cdots d_{n-1} d_{n+1} \cdots d_N \times (r_i)$ Gaussian test matrix $\Omega^{(n)}$ from $N(0,1)$
6: For $\mathbf{Y^{(n)}} = \mathbf{A_{(n)}} \Omega^{(n)}$
7: Construct a matrix $\hat{M}^{(n)}$ whose columns form an orthonormal basis for the range of $\mathbf{Y^{(n)}}$
8: get $\hat{\mathcal{C}} = \mathcal{A} \times_1 \hat{M}^{(1)} \times_2 \hat{M}^{(2)} \cdots \times_N \hat{M}^{(N)}$
9: **Step B: Get approximated SVD**
10: **return** $(\hat{\mathcal{C}}, \hat{M}^{(1)}, \cdots, \hat{M}^{(N)})$

First, I am going to show convergences of matrices can guarantee convergence in a tensor in Theorem 1. To make it clear, if we find matrices such that $\hat{M}_i \to M_i$ we can get $\hat{\mathcal{A}} \to \mathcal{A}$.

Theorem 1. *Let $\mathcal{A} = C \times_1 M_1 \times_2 M_2 \times_3 M_3 \in \mathcal{R}^{d_1 \times d_2 \times d_3}$ where $C \in \mathcal{R}^{r_1 \times r_2 \times r_3}$ and $M_i \in \mathcal{R}^{d_i \times r_i}$ orthonormal matrix for each $i \in [3]$.*

Suppose we have estimation $\hat{M}_1, \hat{M}_2, \hat{M}_3$ *such that* $\|M_i - \hat{M}_i\| \leq \epsilon$ *for each* $i \in [3]$ *Let* $\hat{C} = \mathcal{A} \times_1 \hat{M}_1^{\,t} \times_2 \hat{M}_2^{\,t} \times_3 \hat{M}_3^{\,t}$, *and* $\hat{A} = \hat{C} \times_1 \hat{M}_1 \times_2 \hat{M}_2 \times_3 \hat{M}_3$ *Then,*

$$\|\hat{A} - \mathcal{A}\| \leq \|\mathcal{A}\|(2\|M_1\| + 2\|M_2\| + 2\|M_3\| + 3\epsilon)\epsilon$$

Proof. First, notice that for each i,

$$\begin{aligned}\|M_i M_i^t - \hat{M}_i \hat{M}_i^{\,t}\| &= \|M_i M_i^t - M_i \hat{M}_i^{\,t} + M_i \hat{M}_i^{\,t} - \hat{M}_i \hat{M}_i^{\,t}\| = \|M_i(M_i^t - \hat{M}_i^{\,t}) + (M_i - \hat{M}_i)\hat{M}_i^{\,t}\| \\ &\leq (2\|M_i\| + \epsilon)\epsilon\end{aligned}$$

Main proof is as follows

$$\begin{aligned}\|\mathcal{A} - \hat{\mathcal{A}}\| &= \|\mathcal{A} - \hat{C} \times_1 \hat{M}_1 \times_2 \hat{M}_2 \times_3 \hat{M}_3\| = \|\mathcal{A} - \mathcal{A} \times_1 \hat{M}_1^{\,t} \times_2 \hat{M}_2^{\,t} \times_3 \hat{M}_3^{\,t} \times_1 \hat{M}_1 \times_2 \hat{M}_2 \times_3 \hat{M}_3\| \\
&= \|\mathcal{A} - \mathcal{A} \times_1 \hat{M}_1 \hat{M}_1^{\,t} \times_2 \hat{M}_2 \hat{M}_2^{\,t} \times_2 \hat{M}_2 \hat{M}_2^{\,t}\| \\
&= \|A_{(1)} - \hat{M}_1 \hat{M}_1^{\,t} A_{(1)} (\hat{M}_2 \hat{M}_2^{\,t} \otimes \hat{M}_3 \hat{M}_3^{\,t})\| \\
&\text{By orthonormality of } M_1, M_2 \text{ and } M_3 \\
&= \|M_1 M_1^t A_{(1)} (M_2 M_2^t \otimes M_3 M_3^t) - \hat{M}_1 \hat{M}_1^{\,t} A_{(1)} (\hat{M}_2 \hat{M}_2^{\,t} \otimes \hat{M}_3 \hat{M}_3^{\,t})\| \\
&= \|(M_1 M_1^t - \hat{M}_1 \hat{M}_1^{\,t}) A_{(1)} (M_2 M_2^t \otimes M_3 M_3^t) + \hat{M}_1 \hat{M}_1^{\,t} A_{(1)} (M_2 M_2^t \otimes M_3 M_3^t - \hat{M}_2 \hat{M}_2^{\,t} \otimes \hat{M}_3 \hat{M}_3^{\,t})\| \\
&\leq \|(M_1 M_1^t - \hat{M}_1 \hat{M}_1^{\,t}) A_{(1)} (M_2 M_2^t \otimes M_3 M_3^t)\| \\
&\quad + \|\hat{M}_1 \hat{M}_1^{\,t} A_{(1)} (M_2 M_2^t \otimes M_3 M_3^t - \hat{M}_2 \hat{M}_2^{\,t} \otimes \hat{M}_3 \hat{M}_3^{\,t})\| \\
&\text{Using projection matrix norm property and } \|(M_2 M_2^t \otimes M_3 M_3^t)\| = \|M_2 M_2^t\| \|M_3 M_3^t\| \\
&\leq \|M_1 M_1^t - \hat{M}_1 \hat{M}_1^{\,t}\| \|A_{(1)}\| \\
&\quad + \|A_{(1)}\| \|M_2 M_2^t \otimes M_3 M_3^t - \hat{M}_2 \hat{M}_2^{\,t} \otimes \hat{M}_3 \hat{M}_3^{\,t}\| \\
&= (\|M_1 M_1^t - \hat{M}_1 \hat{M}_1^{\,t}\| + \|M_2 M_2^t \otimes M_3 M_3^t - \hat{M}_2 \hat{M}_2^{\,t} \otimes \hat{M}_3 \hat{M}_3^{\,t}\|) \|\mathcal{A}\| \\
&\leq (\|M_1 M_1^t - \hat{M}_1 \hat{M}_1^{\,t}\| + \|(M_2 M_2^t - \hat{M}_2 \hat{M}_2^{\,t}) \otimes M_3 M_3^t\| + \|\hat{M}_2 \hat{M}_2^{\,t} \otimes (M_3 M_3^t - \hat{M}_3 \hat{M}_3^{\,t})\|) \|\mathcal{A}\| \\
&\leq (\|M_1 M_1^t - \hat{M}_1 \hat{M}_1^{\,t}\| + \|M_2 M_2^t - \hat{M}_2 \hat{M}_2^{\,t}\| + \|M_3 M_3^t - \hat{M}_3 \hat{M}_3^{\,t}\|) \|\mathcal{A}\| \\
&\leq \|\mathcal{A}\| (2\|M_1\| + 2\|M_2\| + 2\|M_3\| + 3\epsilon)\epsilon\end{aligned}$$

□

Furthermore, we can improve Theorem 1 based on the definition of angle between matrices which is given in Definition 1.

Definition 1. *For nonzero subspaces $\mathcal{R}, \mathcal{N} \subset \mathbb{R}^n$, the minimal angle between $\mathcal{R}$ and $\mathcal{N}$ is defined to be the number $0 \leq \theta \leq \pi/2$ that satisfies*

$$\cos\theta(\mathcal{R}, \mathcal{N}) = \max_{u \in \mathcal{R}, v \in \mathcal{N} \|u\|=\|v\|=1} v^t u.$$

Our improved version of error bound is given in Theorem 2.

Theorem 2. *Under the same condition in Theorem 1 with additional condition that*

$$\sin(\theta(span(M_i), span(\hat{M}_i^t))) = \sin(\Theta(M_i, \hat{M}_i^t)) = \sqrt{1 - \cos^2(\Theta(M_i, \hat{M}_i^t))} < \epsilon$$

We can get a following bound.

$$\|\mathcal{A} - \hat{\mathcal{A}}\| \leq 6\epsilon \|\mathcal{A}\|$$

Proof. It suffices to show $\|M_i M_i^t - \hat{M}_i \hat{M}_i^t\| \leq 2\epsilon$ because we can apply this last inequality in the proof of Theorem 1.

$$\|\mathcal{A} - \hat{\mathcal{A}}\| \leq (\|M_1 M_1^t - \hat{M}_1 \hat{M}_1{}^t\| + \|M_2 M_2^t - \hat{M}_2 \hat{M}_2{}^t\| + \|M_3 M_3^t - \hat{M}_3 \hat{M}_3{}^t\|)\|\mathcal{A}\| \leq 6\epsilon \|\mathcal{A}\|$$

Then we are done.

Proof of the above inequality is as follows.

$$\begin{aligned}
\|M_i M_i^t - \hat{M}_i \hat{M}_i^t\| &= \|M_i M_i^t - \hat{M}_i \hat{M}_i^t\| \\
&= \|M_i M_i^t - M_i M_i^t \hat{M}_i \hat{M}_i^t + M_i M_i^t \hat{M}_i \hat{M}_i^t - \hat{M}_i \hat{M}_i^t a\| \\
&\leq \|M_i M_i^t - M_i M_i^t \hat{M}_i \hat{M}_i^t\| + \|M_i M_i^t \hat{M}_i \hat{M}_i^t - \hat{M}_i \hat{M}_i^t\| \\
&= \|M_i M_i^t (I - \hat{M}_i \hat{M}_i^t)\| + \|(M_i M_i^t - I) \hat{M}_i \hat{M}_i^t\| \\
&\leq \sin(\theta) + \sin(\theta) \leq 2\epsilon
\end{aligned}$$

Last inequality follows from combining following 2 lemmas. $\square$

Lemma 1. *If $P_R$and P_N are the orthogonal projectors onto $R$and N , respectively, then*

$$\cos\theta = \|P_N P_R\| = \|P_R P_N\|.$$

Proof. For vectors x and y such that $\|x\| = \|y\| = 1$, we have $P_R x \in \mathcal{R}$ and $P_N y \in \mathcal{N}$ Then

$$\cos\theta = \max_{u\in\mathcal{R}, v\in\mathcal{N} \|u\|=\|v\|=1} v^t u = \max_{u\in\mathcal{R}, v\in\mathcal{N}, \|u\|\le 1 \|v\|\le 1} v^t u = \max_{\|x\|\le 1 \|y\|\le 1} y^t P_{\mathcal{N}} P_{\mathcal{R}} x = \|P_R P_N\|$$

.

$\square$

Lemma 2. *Under the same condition on Lemma 1,*

$$\|P_N(I - P_R)\| \le \sin(\theta)$$

Proof.

$$\begin{aligned}\|P_N(I - P_R)\|^2 &= \max_{u\in\mathcal{R}, \|u\|=1} u^t P_N (I - P_R) u = \max_{u\in\mathcal{R}, \|u\|=1} u^t P_N u - u^t P_N P_R u \\ &\le 1 - \|P_N P_R\|^2 = \sin^2(\theta)\end{aligned}$$

$\square$

Now, I can show that under some good conditions, our estimation $\hat{M}^{(i)}$ in Algorithm 1 is converging to our parameter $M^{(i)}$ with regards to principle angle. In addition, we can guarantee our final tensor estimation $\hat{\mathcal{A}}$ converges to $\mathcal{A}$. Theorem 3 describes this argument.

Theorem 3. *Let $\mathcal{A} = \mathcal{C} \times_1 M^{(1)} \times_2 M^{(2)} \times_3 M^{(3)}$ be a traget tensor where each $M^{(i)} \in R^{d_i \times r_i}$ is orthonormal matrix for each $i \in [3]$ and $\mathcal{D} = \mathcal{A} + \mathcal{E}$ be a give tensor where noise elemens are drawn from $N(0, \sigma^2)$.*
Suppose, $s_{min}(C_{(i)}) >> \sigma\sqrt{\max(d_i, \frac{d_1 d_2 d_3}{d_i})\frac{d_1 d_2 d_3}{d_i r_i}}$ as $d_1, d_2, d_3 \to \infty$, where $s_{min}(C_{(i)})$ is the smallest singular value of $C_{(i)}$.
If we implement Algorithm 1 with an input $\mathcal{D}$, we can get output $(\hat{\mathcal{C}}, \hat{M}^{(1)}, \hat{M}^{(2)}, \hat{M}^{(3)})$ whose angle $\cos\Theta(M^{(i)}, \hat{M}^{(i)}) \to 1$ in probability. Furthermore, $\hat{\mathcal{A}} = \hat{\mathcal{C}} \times_1 \hat{M}^{(1)} \times_2 \hat{M}^{(2)} \times_3 \hat{M}^{(3)}$ converges to $\mathcal{A}$ which means $\|\mathcal{A} - \hat{\mathcal{A}}\| \to 0$ in probability.

Proof. It suffices to show $M^{(1)}$ case.

$$
\begin{aligned}
A_{(1)} &= M^{(1)}(\mathcal{C} \times_2 M^{(2)} \times_3 M^{(3)})_{(1)} \\
&= M^{(1)} C_{(1)} (M^{(3)} \otimes M^{(2)})^T
\end{aligned}
$$

Let's define $B = (M^{(3)} \otimes M^{(2)})^T = \begin{pmatrix} M_1^{(3)} \otimes M_1^{(2)}, & M_1^{(3)} \otimes M_2^{(2)}, & \cdots, & M_{r_3}^{(3)} \otimes M_{r_2}^{(2)} \end{pmatrix}$ where $M_j^{(i)}$ is the j-th column of $M^{(i)}$
Notice that B^T is again orthonormal matrix by orthonormality assumption on each $M^{(i)}$.

$$
\begin{aligned}
A_{(1)}\Omega &= M^{(1)} C_{(1)} (M^{(3)} \otimes M^{(2)})^T \Omega \\
&= M^{(1)} C_{(1)} B \Omega \quad \text{where } \Omega \in R^{d_2 d_3 \times r_1} \text{ whose elements from } i.i.d. N(0,1) \\
&= M^{(1)} C_{(1)} \begin{pmatrix} Z_1 & Z_2, & \cdots & , Z_{r_1} \end{pmatrix} \quad \text{where } Z_i^T = \begin{pmatrix} \sum_{k=1}^{d_2 d_3} B_{1,k}\Omega_{k,i}, & \cdots, & \sum_{k=1}^{d_2 d_3} B_{r_2 r_3,k}\Omega_{k,i} \end{pmatrix} \\
&= M^{(1)} C_{(1)} \mathbf{Z} \quad \text{where } \mathbf{Z} = \begin{pmatrix} Z_1 & Z_2, & \cdots & , Z_{r_1} \end{pmatrix}
\end{aligned} \tag{1}
$$

I am going to use the fact that elements of $\mathbf{Z} \in R^{r_2 r_3 \times r_1}$ are independent $N(0, 1)$ later and I proved this in Lemma 1.
Our estimator $\hat{M^{(1)}}$ for $M^{(1)}$ can be calculated replacing $A_{(1)}$ by $A_{(1)} + E_{(1)}$

$$
\begin{aligned}
(A_{(1)} + E_{(1)})\Omega &= M^{(1)} C_{(1)} \mathbf{Z} + E_{(1)}\Omega \\
&= \hat{M^{(1)}} R \quad \text{(QR decomposition)}
\end{aligned} \tag{2}
$$

Note that $Im(A_{(1)}\Omega) \subset Im(M^{(1)})$ and $Im(A_{(1)}\Omega + E_{(1)}\Omega) = Im(\hat{M^{(1)}})$ Therefore,

$$\begin{aligned}
\cos\Theta(M^{(1)}, \hat{M^{(1)}}) &= \max_{u \in Im(M^{(1)}), v \in Im(\hat{M^{(1)}})} \cos(u, v) \\
&\geq \max_{u \in Im(A_{(1)}\Omega), v \in Im((A_{(1)}+E_{(1)})\Omega)} \cos(u, v) \qquad (3) \\
&= \max_{x \in R^{r_1}, y \in R^{r_1}, \|x\|_2 = \|y\|_2 = 1} \cos(A_{(1)}\Omega x, (A_{(1)} + E_{(1)})\Omega y)
\end{aligned}$$

To show

$$\max_{x \in R^{r_1}, y \in R^{r_1}, \|x\|_2 = \|y\|_2 = 1} \cos(A_{(1)}\Omega x, (A_{(1)} + E_{(1)})\Omega y) \to 1$$

It suffices to show that $\cot(A_{(1)}\Omega x, (A_{(1)} + E_{(1)})\Omega y) \to \infty$ for some x and y such that $\|x\| = \|y\| = 1$. Because if $\cot(A_{(1)}\Omega x, (A_{(1)} + E_{(1)})\Omega y) \to \infty$ for some x and y, then $\cos(A_{(1)}\Omega x, (A_{(1)} + E_{(1)})\Omega y) \to 1$ for some x and y. Therefore, maximum of cos between two subspaces becomes 1. So let's focus on proving $\cot(A_{(1)}\Omega x, (A_{(1)} + E_{(1)})\Omega y) \to \infty$ for fixed x and y. Consider following inequality.

$$\cot(A_{(1)}\Omega x, (A_{(1)} + E_{(1)})\Omega y) \geq \frac{\|A_{(1)}\Omega x\|_2}{\|E_{(1)}\Omega y\|_2} \geq \frac{s_{min}(C_{(1)})\sqrt{\chi^2_{r_1}}}{\|E\|_2 \|\Omega y\|_2} \qquad (4)$$

To get numerator part in equation (4),

$$\|A_{(1)}\Omega x\|_2 \overset{(1)}{=} \|M^{(1)} C_{(1)} \mathbf{Z} x\|_2 = \|C_{(1)} \sum_{i=1}^{r_1} Z_i x_i\|_2 \quad \text{by orthonormality of } M^{(1)} \qquad (5)$$

Combining Lemma 3 which says that all elements of $\mathbf{Z}$ are $i.i.d. N(0,1)$ and the fact that $\|x\|_2 = 1$, we can check all elements of $\sum_{i=1}^{r_1} Z_i x_i$ from $i.i.d. N(0,1)$. Finally, we can have following equations.

$$
\begin{aligned}
\|A_{(1)}\Omega x\|_2 = \|C_{(1)}\sum_{i=1}^{r_1} Z_i x_i\|_2 &= \|\tilde{Z}\|_2 \text{ where } \tilde{Z} \sim N_{r_1}(0, C_{(1)}C_{(1)}^T) \\
&\text{Based on Eigen-Value Decomposition, } C_{(1)}C_{(1)}^T = P\Lambda P^T \\
&\text{where } P \text{ is an orthonormal matrix and } \Lambda = diag(\lambda_{r_1}\cdots\lambda_1) \text{ s.t. } \lambda_{r_1} \geq \cdots \geq \lambda_1 \geq 0 \\
&= \|P^T\tilde{Z}\|_2 \quad \text{where } P^T\tilde{Z} \sim N_{r_1}(0, \Lambda) \\
&= \sqrt{\lambda_1^2 V_1 + \cdots \lambda_{r_1}^2 V_{r_1}} \text{ where } V_i \sim \chi_1^2 \quad \text{for } i \in [r_1] \\
&\geq |\lambda_1|\sqrt{V_1 + \cdots + V_{r_1}} \\
&= |\lambda_1|\sqrt{\chi_{r_1}^2} \\
&= s_{min}(C_{(1)})\sqrt{\chi_{r_1}^2}
\end{aligned}
\tag{6}
$$

Finally, we can make numerator part in (4). Also, you note that $\|\Omega y\|_2^2 \sim \chi^2(d_2d_3)$ because $\|y\|_2 = 1$. Therefore, we can get $\|\Omega y\|_2 \asymp (1 + o(1))\sqrt{d_2d_3}$. Furthermore, we have that $\|E\| \asymp (2 + o(1))\sigma\sqrt{\max(d_1, d_2d_3)}$ by Lemma 4. Finally, for any fixed $L > 0$ we can have following inequality.

$$
\begin{aligned}
P(\cot(A_{(1)}\Omega x, (A_{(1)} + E_{(1)})\Omega y) \geq L) &\geq P(\frac{s_{min}(C_{(1)})\sqrt{\chi_{r_1}^2}}{\|E\|_2\|\Omega y\|_2} \geq L) \\
&\geq P(\sqrt{\chi_{r_1}^2} \geq \frac{2L\sigma\sqrt{d_2d_3\max(d1, d_2d_3)}}{s_{min}(C_{(1)})}) \\
&= P(\chi_{r_1}^2 \geq \frac{4L^2\sigma^2 d_2d_3\max(d1, d_2d_3)}{s_{min}(C_{(1)})^2}) \\
&\geq 1 - \left(4\lambda e^{1-4\lambda}\right)^{\frac{r_1}{2}}
\end{aligned}
\tag{7}
$$

where $\lambda \overset{def}{=} \frac{L^2\sigma^2 d_2d_3\max(d1,d_2d_3)}{r_1 s_{min}(C_{(1)})^2}$. In the last line of equation (7), we used Chernoff bounds which states that $P(\chi_r^2 \geq t) \geq 1 - \left(\frac{t}{r}e^{1-\frac{t}{r}}\right)^{\frac{r}{2}}$ for any $t \geq 0$. Based on the main condition of our theorem, $\lambda \to 0$ for fixed L. Therefore, we conclude that $\cot(A_{(1)}\Omega x, (A_{(1)}+E_{(1)})\Omega y) \geq L$ with high probability. Finally, we can get desired result sending $L \to \infty$.

The last argument that $\|\mathcal{A}-\hat{\mathcal{A}}\| \to 0$ is directly derived from Theorem 2 and Theorem 3 $\square$

Lemma 3. *In the proof of the Theorem 1, elements of* $\mathbf{Z} = \begin{pmatrix} Z_1 & Z_2, & \cdots & , Z_{r_1} \end{pmatrix}$ *is from* $i.i.d N(0,1)$ *where* $Z_i^T = \begin{pmatrix} \sum_{k=1}^{d_2 d_3} B_{1,k}\Omega_{k,i}, & \cdots, & \sum_{k=1}^{d_2 d_3} B_{r_2 r_3,k}\Omega_{k,i} \end{pmatrix}^T = (z_{1,i}, \cdots, z_{r_1 r_2,i})^T$ *and* $B = (M^{(3)} \otimes M^{(2)})^T = \begin{pmatrix} M_1^{(3)} \otimes M_1^{(2)}, & M_1^{(3)} \otimes M_2^{(2)}, & \cdots, & M_{r_3}^{(3)} \otimes M_{r_2}^{(2)} \end{pmatrix}$

Proof. It's easy to check that Z_i and Z_j are independent where $i \neq j$ because all elements of Z_i are made of Ω_i = i-th column of Ω. Therefore, it suffices to show all elements of Z_1 are independent and from $N(0,1)$.

1. $z_{1,1} \sim N(0,1)$
   Note $z_{1,1} = \sum_{k=1}^{d_2 d_3} B_{1,k}\Omega_{k,1} = [B^1]^T \Omega_1$.
   Since $\|B^1\| = 1$ and $\Omega_1 \overset{i.i.d}{\sim} N(0,1)$, $z_{1,1}$ is from $N(0,1)$

2. $z_{1,1}, \cdots, z_{r_2 r_3,1}$ are independent.
   Let's define a function $(ind_1, ind_2) : N \to N \times N$ which satisfies $B_i = M_{ind_1(i)}^{(3)} \otimes M_{ind_2(i)}^{(2)}$.
   To give a simple example, $(ind_1(1), ind_2(1)) = (1,1)$ because $B_1 = M_1^{(3)} \otimes M_1^{(2)}$.
   For $i \neq j$,

$$\begin{aligned} Cov(z_{i,1}, z_{j,1}) &= Cov(\sum_{k=1}^{d_2 d_3} B_{i,k}\Omega_{k,1}, \sum_{k=1}^{d_2 d_3} B_{j,k}\Omega_{k,1}) \\ &= (B_i^T)^T (B_j^T) \\ &= 0 \end{aligned}$$

Therefore, all elements of $\mathbf{Z}$ is from $i.i.d.N(0,1)$ by 1,2 □

Lemma 4 (Spectral norm of Gaussin matrix)**.** *Let* $E \in R^{m \times n}$ *be a random matrix with i.i.d.* $N(0,1)$ *entries. Then, we have, with very high probability,*

$$\|E\|_\sigma \asymp (2 + o(1))\sqrt{\max(m,n)}$$

We can apply Theorem 1 to the lower dimension case.

Corollary 1. *Consider a noisy rank 1 matrix model* $D = \lambda a \otimes b + E$*, where* $\lambda \in R_+$ *is a scalar,* $a \in R^{d_1}, b \in R^{d_2}$ *are unit-1 vectors, and* $E \in R^{d_1 \times d_2}$ *is a Gaussian matrix with* $i.i.d.N(0, \sigma^2)$ *entires. Define a random projection*

$$\hat{a} = D\Omega, \ \ where \ \Omega = (z_1, \cdots, z_{d_2})^T \overset{i.i.d.}{\sim} N(0, 1)$$

Suppose $\lambda >> \sigma\sqrt{d_2 \max(d_1, d_2)}$ *as* $d_1, d_2 \to \infty$*. Then,* $\cos\Theta(a, \hat{a}) \to 1$ *in probability.*

Proof. Put $\mathcal{C} = \lambda, M^{(1)} = a, M^{(2)} = b$ and $M^{(3)} = 1$ into Theorem 1. Then you can get the result of Corollary 1. $\square$

2 New randomized tucker decomposition algorithm and simulations

2.1 Algorithm 2.

A weak point of the Algorithm 1 is the time cost generating large dimension of random test matrices. So algorithm 2 suggests another way to this weak point in algorithm 1. Algorithm 2 generates $N-1$ Gaussian random matrices whose sizes are $d_1 \times r_n, \cdots d_{n-1} \times r_n, d_{n+1} \times r_n, \cdots d_N \times r_n$ for each mode n. Algorithm 2 has an advantage for generating random matrices considering algorithm 1 drawing one Gaussian test matrix whose size is $d_1 \cdots d_{n-1} d_{n+1} \cdots d_N \times r_n$. Algorithm 2 is as follows.

Algorithm 2 Approx tensor SVD 2

procedure SVD($\mathcal{A}$)

Step A: Approximate SVD of $\mathcal{A}$

for $n \leftarrow 1 : N$ **do**

Unfold $\mathcal{A}$ as $A_{(n)}$

Generate Gaussian test matrices $P_1 \in R^{d_1 \times r_n} \cdots P_{n-1} \in R^{d_{n-1} \times r_n} P_{n+1} \in R^{d_{n+1} \times r_n} \cdots P_N \in R^{d_N \times r_n}$ from $N(0,1)$

Get $\Omega^{(n)} = P_1 \odot \cdots \odot P_{n-1} \odot P_{n+1} \odot \cdot \odot P_N$

For $\mathbf{Y^{(n)}} = \mathbf{A_{(n)}} \Omega^{(n)}$

Construct a matrix $\hat{M}^{(n)}$ whose columns form an orthonormal basis for the range of $\mathbf{Y^{(n)}}$

get $\hat{\mathcal{C}} = \mathcal{A} \times_1 \hat{M}^{(1)} \times_2 \hat{M}^{(2)} \cdots \times_N \hat{M}^{(N)}$

Step B: Get approximated SVD

return $(\hat{\mathcal{C}}, \hat{M}^{(1)}, \cdots, \hat{M}^{(N)})$

2.2 Simulation.

I compared 3 randomized tucker decomposition algorithms: algorithm 1, algorithm 2 above and the algorithm we covered 2 weeks ago. I denoted algorithm 1 as first method, algorithm

2 as third method and the last one as second method. My first simulation is for comparing accuracy from a signal among different methods. Simulation procedure is as follows.

1. Make signal tensor $B \in R^{100\times100\times100}$
2. Make noise Gaussian E tensors according to different standard deviations from 0.01 to 0.5
3. Estimate $\hat{B}$ for the signal tensor B from given tensor $B+E$
4. Calculate Frobenius norm of $\|\hat{B}-B\|_F$

Figure 1 shows how far our estimates are from original signal tensor.

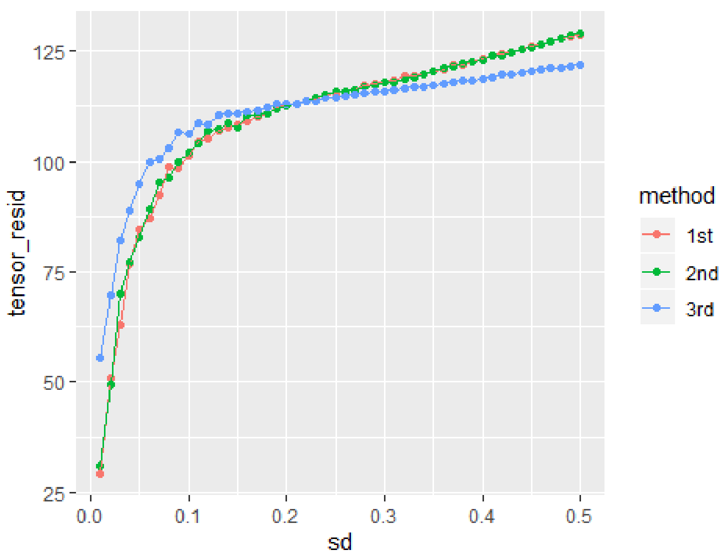


Figure 1: tensor resid in y axis means Frobenius norm between signal and an estimate. 3rd method has about 20 greater tensor resid when sd is less than 0.1. But 3rd method's norm difference from the signal becomes smaller than other methods as sd is greater than 0.3

The second simulation is for calculating angles between estimates and signal tensor. Simulation procedure is as follows.

1. Make signal tensor with $B = a \otimes b \otimes c$ where $a \in R^{100}, b \in R^{100}$ and $c \in R^{100}$
2. Make noise Gaussian E tensors according to different standard deviations from 0.01 to 0.5

3. Estimate $\hat{a}\hat{b}$ and $\hat{c}$ for the signal tensor a, b and c from given tensor $B + E$

4. Calculate $\cos\Theta(a, \hat{a})$, $\cos\Theta(b, \hat{b})$ and $\cos\Theta(c, \hat{c})$

5. Repeat 100 times and average outputs.

Figure 2 shows that the first method has the best accuracy among 3 methods in respect to both angles and MSE.

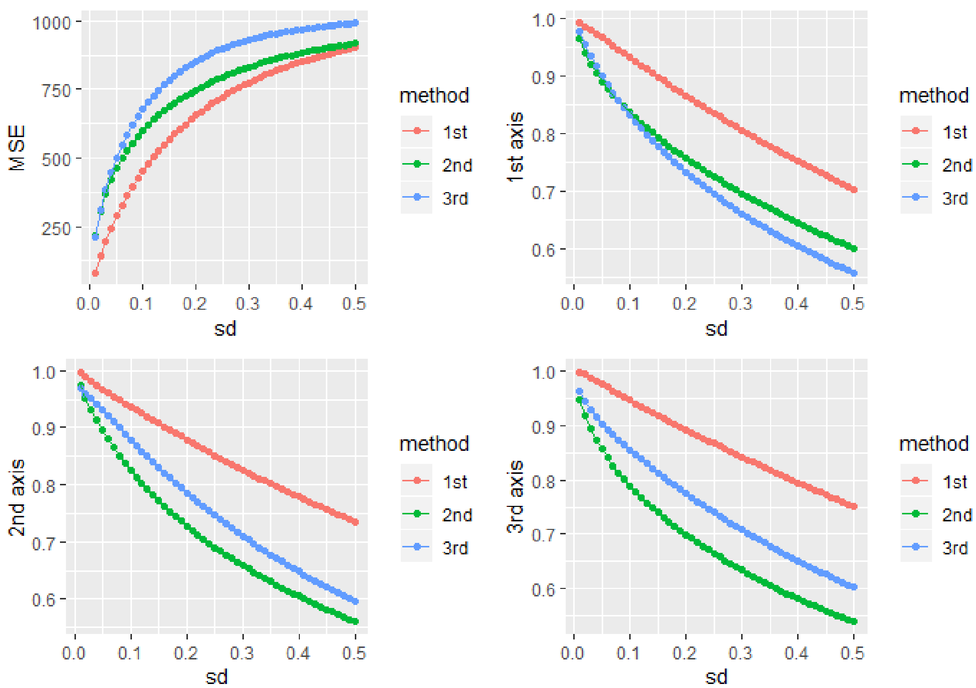


Figure 2: The first method shows better performance in all aspects. The second and Third method has similar performance with respect to angles between the signal and estimate

3 Ordinal tensor model simulation

3.1 Algorithm construction

In this section, we describe the algorithm which can be used to solve above optimization problem. We utilized a formulation of tucker decomposition, and turn the optimization into a block wise convex problem. We will divide cases into 2, when we know bin boundary $\alpha_1, \cdots, \alpha_K$and when we don't have any information about bin boundary.

3.1.1 Known Bin Boundary

From previous data or experience in the past we may know bin boundary parameter α. When we know this bin boundary, finding an estimator becomes optimization problem of

$$\mathcal{L}_{\mathcal{Y}}(\Theta,\alpha) = -\sum_{i_1,\cdots,i_N}\Big[\sum_{l=1}^{K}\mathbb{1}(y_{i_1,\cdots,i_N}=l)\log(\pi_l(\theta,\alpha))\Big]$$
$$\pi_l = P(Y_{i_1,\cdots,i_N}=l|\theta_{i_1,\cdots,i_N},\alpha) = \text{logit}(\alpha_l+\theta_{i_1,\cdots,i_N}) - \text{logit}(\alpha_{l-1}+\theta_{i_1,\cdots,i_N}) \text{ for } l<K$$
$$\pi_K = P(Y_{i_1,\cdots,i_N}=K|\theta_{i_1,\cdots,i_N},\alpha) = 1-\text{logit}(\alpha_{K-1}+\theta_{i_1,\cdots,i_N}) \text{ for } l=K$$

where $\Theta = \mathcal{C}\times_1 A_1\cdots\times_N A_N$. Our strategy for this optimization is updating each block by fixing other blocks. To give you an simple example, let's assume that our tensor data has mode 3 i.e. $\Theta = \mathcal{C}\times_1 A_1\times_2 A_2\times_3 A_3$.
First, let's update A_1 fixing A_2, A_3 and $\mathcal{C}$. If you do mode 1 metricize Θ and vectorize it , you can express this as an linear operation of vectorized A_1 as follows.

$$Vec(\Theta_{(1)}) = \big((A_3\otimes A_2)\mathcal{C}_{(1)}^T\otimes I_{d_1}\big)Vec(A_1)$$

Then we can estimate A_1 that minimizes $L_{\mathcal{Y}}(Vec(A_1))$. To make our algorithm speed-up, we decomposed the equation into a total of d_1 equations

$$\Theta_{(1)}[i,:] = A_1[i,:]\big((A_3\otimes A_2)C_{(1)}^T\big) \text{ where } i\in[d_1]$$

Based on this we can perform sub-optimizations and $L_{\mathcal{Y}}(\Theta)$ becomes $\sum_{i=1}^{d_1} L_{\mathcal{Y}}(A_1[i,:])$ which is a simple low dimensional convex optimization problem.
Likewise, you can update A_2 fixing A_1, A_3 and $\mathcal{C}$ using following formula, make it a convex optimization problem again.

$$\Theta_{(2)}[i,:] = A_2[i,:]\big((A_3\otimes A_1)\mathcal{C}_{(2)}^T\big) \text{ where } i\in[d_2]$$

You can repeat this on A_3 fixing A_2, A_3 and $\mathcal{C}$ using following formula.

$$\Theta_{(3)}[i,:] = A_3[i,:]\big((A_2 \otimes A_1)\mathcal{C}_{(3)}^T\big) \text{ where } i \in [d_2]$$

Finally, you use a formula below to update core tensor $\mathcal{C}$ with fixed A_1, A_2 and A_3

$$Vec(\Theta_{(1)}) = \big(A_3 \otimes A_2 \otimes A_1\big)Vec(\mathcal{C}_{(1)})$$

By iterating this until it converges, you can get an estimator of $\arg\min_\Theta L_{\mathcal{Y}}(\Theta)$. I used method "BFGS", quasi-Newton method to find each axis optimizer. The full algorithm is described in Algorithm 2.

Algorithm 3 Ordinal tensor optimization with known boundary α

Input: $\mathcal{C}^0 \in \mathbf{R}^{r_1\times\cdots\times r_N}, A_1^0 \in \mathbf{R}^{d_1\times r_1}, \cdots, A_N^0 \in \mathbf{R}^{d_N\times r_N}$

Output: Optimizor of $\mathcal{L}_Z(\alpha,\Theta)$ given α

for t = 1,2,$\cdots$, **do** until convergence,

Update A_n

for n = 1,2,$\cdots$,N **do**

$\Theta_{(n)} = A_n^t\mathcal{C}_{(n)}^t\big(A_{n+1}^t \otimes \cdots \otimes A_N^t \otimes A_1^{t+1} \otimes \cdots \otimes A_{n-1}^{t+1}\big)^T$

$Vec(\Theta_{(n)}) = \Bigg(\big(A_{n+1}^t \otimes \cdots \otimes A_N^t \otimes A_1^{t+1} \otimes \cdots \otimes A_{n-1}^{t+1}\big)(C_{(n)}^t)^T \otimes I_{d_n}\Bigg)Vec(A_n^t)$

$Vec(A_n^{t+1}) = \arg\max(\mathcal{L}_{\mathcal{Y}}(\alpha, Vec(\Theta_{(n)}))$

Get A_n^{t+1}

Update $\mathcal{C}$

$\Theta_{(1)} = A_1^{t+1}\mathcal{C}_{(1)}^t\big(A_N^{t+1} \otimes \cdots \otimes A_2^{t+1}\big)^T$

$Vec(\Theta_{(1)}) = \big(A_N^{t+1} \otimes \cdots \otimes A_1^{t+1}\big)Vec(C_{(1)}^t)$

$Vec(C_{(1)}^t) = \arg\max(\mathcal{L}_{\mathcal{Y}}(\alpha, Vec(\Theta_{(1)}))$

Get $\mathcal{C}^{t+1}$

return α, Θ

3.1.2 Unknown Bin Boundary

In real world, knowing bin boundary rarely happens so α also becomes parameter we have to estimate. In this case, we can add one more block of α to Algorithm 2. So after updating $\mathcal{C}, A_1, A_2$ and A_3 in the example section 2.1.1 , we can update α by fixing $\Theta = \mathcal{C} \times_1 A_1 \times_2 A_2 \times_3 A_3$. This updating process for α is just finding intercepts in ordinal logistic regression with fixed slope as 1. The full algorithm is described in Algorithm 3.

Algorithm 4 Ordinal tensor optimization with unknown boundary α

Input: $\mathcal{C}^0 \in \mathbf{R}^{r_1 \times \cdots \times r_N}, A_1^0 \in \mathbf{R}^{d_1 \times r_1}, \cdots, A_N^0 \in \mathbf{R}^{d_N \times r_N}$

Output: Optimizor of $\mathcal{L}_Z(\alpha, \Theta)$ given α

for t = 1,2,$\cdots$, **do** until convergence,

Update A_n

for n = 1,2,$\cdots$,N **do**

$\Theta_{(n)} = A_n^t \mathcal{C}_{(n)}^t \big(A_{n+1}^t \otimes \cdots \otimes A_N^t \otimes A_1^{t+1} \otimes \cdots \otimes A_{n-1}^{t+1}\big)^T$

$Vec(\Theta_{(n)}) = \Bigg(\big(A_{n+1}^t \otimes \cdots \otimes A_N^t \otimes A_1^{t+1} \otimes \cdots \otimes A_{n-1}^{t+1}\big)(C_{(n)}^t)^T \otimes I_{d_n}\Bigg) Vec(A_n^t)$

$Vec(A_n^{t+1}) = \arg\max(\mathcal{L}_{\mathcal{Y}}(\alpha^t, Vec(\Theta_{(n)}))$

Get A_n^{t+1}

Update $\mathcal{C}$

$\Theta_{(1)} = A_1^{t+1} \mathcal{C}_{(1)}^t \big(A_N^{t+1} \otimes \cdots \otimes A_2^{t+1}\big)^T$

$Vec(\Theta_{(1)}) = \big(A_N^{t+1} \otimes \cdots \otimes A_1^{t+1}\big) Vec(C_{(1)}^t)$

$Vec(C_{(1)}^t) = \arg\max(\mathcal{L}_{\mathcal{Y}}(\alpha^t, Vec(\Theta_{(1)}))$

Get $\mathcal{C}^{t+1}$

Update α

$\alpha^{t+1} = \arg\max(\mathcal{L}_{\mathcal{Y}}(\alpha, \Theta^{t+1}))$

return α, Θ

3.2 Simulations.

First simulation is to check how our algorithm converges. Simulation procedure is as follows.

1. Make a tensor $\Theta \in R^{20 \times 20 \times 20}$

2. Get realization ordinal tensor data $\mathcal{Y} \in R^{20\times 20\times 20}$ such that $P(y_{ijk} = l) = \pi_l(\theta_{ijk}, \omega)$ where $\omega = (-0.2, 0, 2)$

3. Implement algorithm 3 with random Gaussian initial points, save $\theta^{(t)}$ for each step and get an estimator $\hat{\theta}$ for θ.

4. Calculate $L_{\mathcal{Y}}(\Theta^{(t)})$ and $\|(\Theta^{(t)} - \Theta^{(t-1)}\|_\infty$

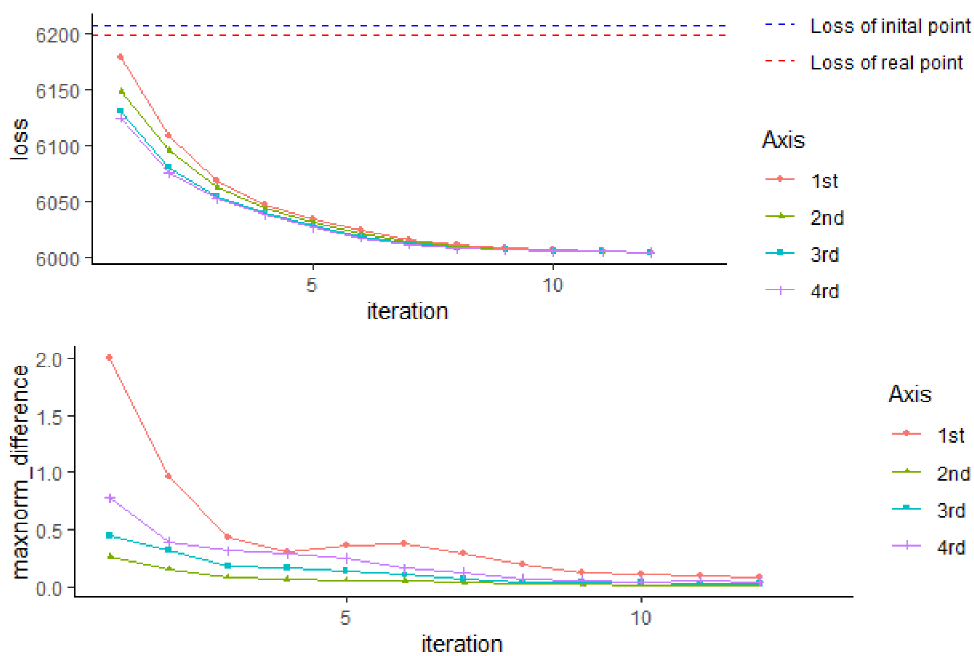


Figure 3: Blue horizontal line is a value of $L_{\mathcal{Y}}(\Theta^{(0)})$. Red horizontal line is a value of $L_{\mathcal{Y}}(\Theta)$. Above figure shows our $\hat{\theta}$ has about 200 smaller value than the loss value of real value. Also the second figure shows our algorithm converges after 13 iterations.

The second simulation is to check how close our estimator $\hat{\Theta}$ is from Θ. I divided cases into two. First one is when we know threshold ω and the second is we have no knowledge on ω. Simulation procedure is as follows.

When we know threshold $\omega = (-0.2, 0, 2)$.

1. Make a tensor $\Theta = \mathcal{C} \times_1 A_1 \times_2 A_2 \times_3 A_3 \in R^{d\times d\times d}$. All of their entries were drawn from unif(-1,1)

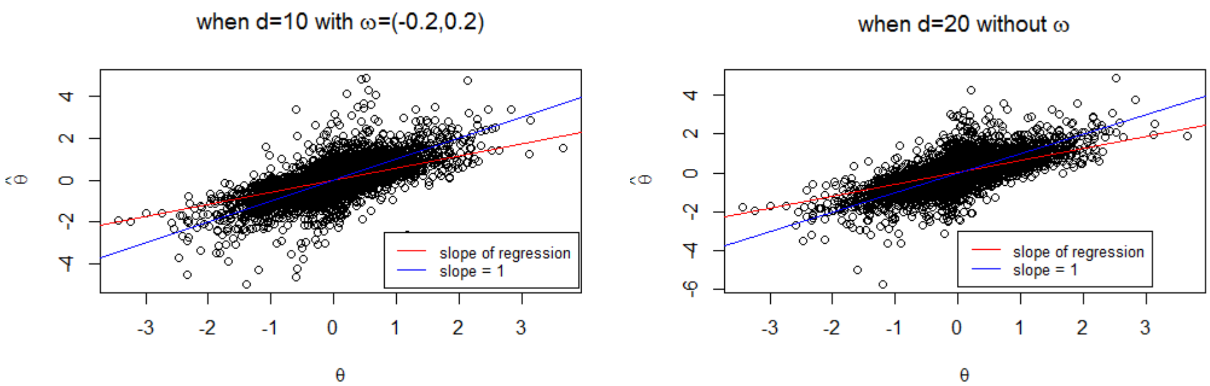
   We got $\|\Theta\|_\infty = 2.506964$ when d = 20 and $\|\Theta\|_\infty = 3.049013$ when d = 30

2. Get realization ordinal tensor data $\mathcal{Y} \in R^{d\times d\times d}$ such that $P(y_{ijk} = l) = \pi_l(\theta_{ijk}, \omega) \quad l \in [3]$.

3. Estimate Θ using Algorithm 3.

4. Repeat for $d \in \{20, 30\}$

Figure 4 shows our result.



(a) When we know threshold ω (b) When we don't know threshold ω

Figure 4: Scatter plots between real θ and our estimator $\hat{\theta}$. Red lines are slopes of ordinary least square estimators. Blue lines are line of $y = x$. Each slope from left to right figure is 0.68 and 0.691. Both cases have almost 0 intercept.

When we have no knowledge on threshold $\omega = (-0.2, 0, 2)$.

1. Make a tensor $\Theta = \mathcal{C} \times_1 A_1 \times_2 A_2 \times_3 A_3 \in R^{d\times d\times d}$. All of their entries were drawn from unif(-1,1)

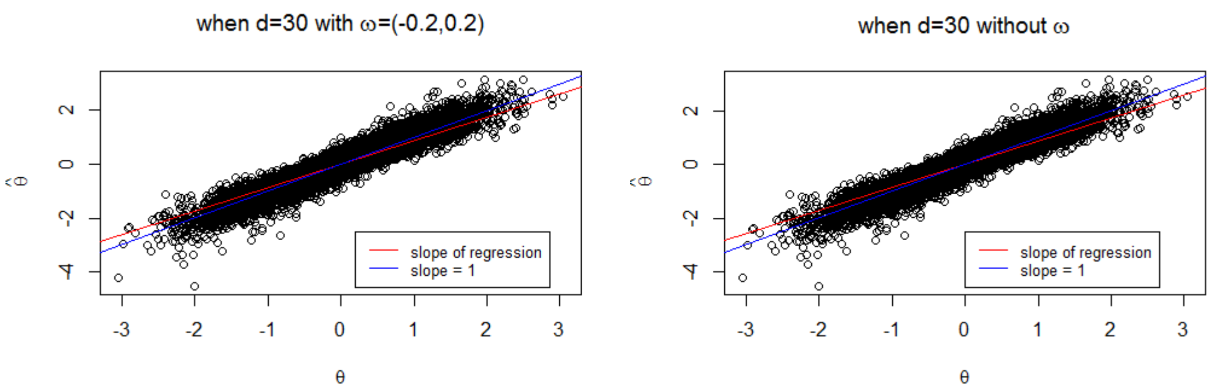
   We got $\|\Theta\|_\infty = 2.506964$ when d $=$ 20 and $\|\Theta\|_\infty = 3.049013$ when d $=$ 30

2. Get realization ordinal tensor data $\mathcal{Y} \in R^{d\times d\times d}$ such that $P(y_{ijk} = l) = \pi_l(\theta_{ijk}, \omega) \quad l \in [3]$.

3. Estimate Θ, ω using Algorithm 4.

4. Repeat for $d \in \{20, 30\}$

Our estimated $\hat{\omega} = (-0.198, 0.201)$ when d $= 20$ and $\hat{\omega} = (-0.202, 0.199)$ when d $= 30$. Since real $\omega = (-0.2, 0.2)$, Algorithm 4 successfully estimated ω. Figure 5 shows overall results.



(a) When we know threshold ω (b) When we don't know threshold ω

Figure 5: Scatter plots between real θ and our estimator $\hat{\theta}$. Red lines are slopes of ordinary least square estimators. Blue lines are line of $y = x$. Regression coefficients on the first scatter plot is intercept $= 0.001631$ and slope $= 0.867366$. Regression coefficients on the second scatter plot is intercept $= 0.00156$ and slope $= 0.86410$. Comparison between Figure 4 and Figure 5 shows the more data we have, the more accurate we can estimate θ.

4 Some questions and comments

1. I haven't started to study algorithm and convergence properties. I will start doing this on Wednesday.

2. Good initialization: I just set random initialization from normal distribution. Are there existing better way to set initial values based on ordinal tensor data?

5 Algorithm codes

5.1 Algorithm 1 and 2.

```
library(rTensor)

```

```r
StepAm = function(A,r,p){
  m = nrow(A); n = ncol(A)
  l = r+p
  omega = matrix(rnorm(n*l),nrow = n, ncol = l)
  Y = A%*%omega
  Q = qr.Q(qr(Y))
  return(Q)
}

StepA = function(A,r,p){
  m = nrow(A); n = ncol(A)
  l = r + p
  P1 = matrix(rnorm(m*r),nrow = m, ncol = r)
  P2 = matrix(rnorm(m*r),nrow = m, ncol = r)
  omega = khatri_rao(P1,P2)
  Y = A%*%omega
  Q = qr.Q(qr(Y))
  return(Q)
}

# Algorithm 1.
tensor_svd = function(tnsr,k1,k2,k3,p){
  App = list(Z=NULL,U=NULL)
  mat1 <- k_unfold(tnsr,m=1)
  mat2 <- k_unfold(tnsr,m=2)
  mat3 <- k_unfold(tnsr,m=3)
  Q1 <- StepAm(mat1@data,k1,p)
  Q2 <- StepAm(mat2@data,k2,p)
  Q3 <- StepAm(mat3@data,k3,p)
  Coreten <- ttm(ttm(ttm(tnsr,t(Q1),1),t(Q2),2),t(Q3),3)
  App$Z = Coreten
  App$U = list(Q1,Q2,Q3)
  return(App)
}

# Algorithm 2.
```

```
tensor_svd3 = function(tnsr,k1,k2,k3,p){
  App = list(Z=NULL,U=NULL)
  mat1 <- k_unfold(tnsr,m=1)
  mat2 <- k_unfold(tnsr,m=2)
  mat3 <- k_unfold(tnsr,m=3)
  Q1 <- StepA(mat1@data,k1,p)
  Q2 <- StepA(mat2@data,k2,p)
  Q3 <- StepA(mat3@data,k3,p)
  Coreten <- ttm(ttm(ttm(tnsr,t(Q1),1),t(Q2),2),t(Q3),3)
  App$Z = Coreten
  App$U = list(Q1,Q2,Q3)
  return(App)
}
```

5.2 Simulation for comparison of tensor algorithms.

```


## Recovery from a noise simulation.

sd = 0.01*1:50
result = data.frame(matrix(nrow = 150, ncol =3))
names(result)  <- c("sd","tensor_resid","method")
for (i in 1:50) {
  s=sd[i]
  result[i,1] = s
  result[i+50,1] = s
  result[i+100,1] = s
  e = as.tensor(array(rnorm(1000000,mean =0,sd = s),dim = c(100,100,100)))
  D = B+e
  est1 = tensor_svd(D,20,20,20,5)
  est2 = tensor_svd2(D,20,20,20,5)
  est3 = tensor_svd3(D,20,20,20,5)
  result[i,2] = tensor_resid(B,est1)
  result[i+50,2] = tensor_resid(B,est2)
  result[i+100,2]= tensor_resid(B,est3)
```

```
  result[i,3] = "1st"
  result[i+50,3] = "2nd"
  result[i+100,3]= '3rd'

}


ggplot(data = result, aes(x = sd, y = tensor_resid,col = method))+geom_
    point()+
  geom_line()

## Angle simulation.
a <- as.matrix(rnorm(100))
b <- as.matrix(rnorm(100))
c <- as.matrix(rnorm(100))
tnsr <- as.tensor(array(1,dim = c(1,1,1)))
X <- ttm(ttm(ttm(tnsr,a,1),b,2),c,3)
sd = 0.01*1:50
result = data.frame(matrix(0,nrow = 150, ncol =5))
names(result) <- c("sd","angle1","angle2","angle3","method")


for (i in 1:50) {
  s=sd[i]
  result[i,1] = s
  result[i+50,1] = s
  result[i+100,1] = s
  e = as.tensor(array(rnorm(1000000,mean =0,sd = s),dim = c(100,100,100)))
  D = X+e
  for (j in 1:100) {
    set.seed(j)
    est1 = tensor_svd(D,1,1,1,0)
    est2 = tensor_svd2(D,1,1,1,0)
    est3 = tensor_svd3(D,1,1,1,0)
    result[i,2] <- result[i,2]+angle(est1$U[[1]],a)
    result[i,3] <- result[i,3]+angle(est1$U[[2]],b)
```

```
    result[i,4] <- result[i,4]+angle(est1$U[[3]],c)
    result[i+50,2] <- result[i+50,2]+angle(est2$U[[1]],a)
    result[i+50,3] <- result[i+50,3]+angle(est2$U[[2]],b)
    result[i+50,4] <- result[i+50,4]+angle(est2$U[[3]],c)
    result[i+100,2] <- result[i+100,2]+angle(est3$U[[1]],a)
    result[i+100,3] <- result[i+100,3]+angle(est3$U[[2]],b)
    result[i+100,4] <- result[i+100,4]+angle(est3$U[[3]],c)

  }
  result[i,5] = "1st"
  result[i+50,5] = "2nd"
  result[i+100,5]= '3rd'

}
result[,2:4] <- result[,2:4]/100

g2 <- ggplot(data = result, aes(x=sd,y = abs(angle1) ,color = method))+
  geom_point(aes(x=sd, y = abs(angle1)))+geom_line(aes(x=sd, y = abs(angle1)))+ylab("1st axis")
g3 <- ggplot(data = result, aes(x=sd,y = abs(angle2) ,color = method))+
  geom_point(aes(x=sd, y = abs(angle2)))+geom_line(aes(x=sd, y = abs(angle2)))+ylab("2nd axis")
g4 <- ggplot(data = result, aes(x=sd,y = abs(angle3) ,color = method))+
  geom_point(aes(x=sd, y =  abs(angle3)))+geom_line(aes(x=sd, y =  abs(angle3)))+ylab("3rd axis")
library(gridExtra)
grid.arrange(g2,g3,g4)
```

6 Algorithm 3 and 4(complete version)

```
library(MASS)
library(rTensor)
library(pracma)
library(ggplot2)
library(ggthemes)
```

```
library(gridExtra)

# Some functions needed for Algorithm 3 and 4.
realization = function(tnsr,alpha){
  thet <- k_unfold(tnsr,1)@data
  theta1 <- thet + alpha[1]
  theta2 <- thet + alpha[2]
  result <- k_unfold(tnsr,1)@data
  p1 <- logistic(theta1)
  p2 <- logistic(theta2)-logistic(theta1)
  p3 <- matrix(1,nrow = nrow(thet),ncol = ncol(thet))-logistic(theta2)
  for (i in 1:nrow(thet)) {
    for(j in 1:ncol(thet)){
      result[i,j] <- sample(c(1,2,3),1,prob= c(p1[i,j],p2[i,j],p3[i,j]))
    }
  }
  return(k_fold(result,1,modes = tnsr@modes))
}


h1 = function(A_1,W1,ttnsr,omega){
  thet =W1%*%c(A_1)
  p1 = logistic(thet + omega[1])
  p2 = logistic(thet + omega[2])
  p = cbind(p1,p2-p1,1-p2)
  return(-sum(log(c(p[which(c(ttnsr)==1),1],p[which(c(ttnsr)==2),2],p[
   which(c(ttnsr)==3),3]))))
}
g1 = function(A_1,W1,ttnsr,omega){
  thet =W1%*%c(A_1)
  p1 = logistic(thet + omega[1])
  p2 = logistic(thet + omega[2])
  q1 <- p1-1
  q2 <- (p2*(1-p2)-p1*(1-p1))/(p1-p2)
  q3 <- p2
```

```
  gd = apply(diag(q1[which(c(ttnsr)==1)])%*%W1[which(c(ttnsr)==1),],2,sum) +
    apply(diag(q2[which(c(ttnsr)==2)])%*%W1[which(c(ttnsr)==2),],2,sum)+
    apply(diag(q3[which(c(ttnsr)==3)])%*%W1[which(c(ttnsr)==3),],2,sum)
  return(gd)
}


comb = function(A,W,ttnsr,k,omega,alph=TRUE){
  nA = A
  tnsr1 <- k_unfold(as.tensor(ttnsr),k)@data
  if (alph==TRUE) {
    l <- lapply(1:nrow(A),function(i){optim(A[i,],
                             function(x) h1(x,W,tnsr1[i,],omega),
                             function(x) g1(x,W,tnsr1[i,],omega),
                             method = "BFGS")$par})
    nA <- matrix(unlist(l),nrow = nrow(A),byrow = T)
  }else{
    l <- lapply(1:nrow(A),function(i){constrOptim(A[i,],
                             function(x) h1(x,W,tnsr1[i,],omega),function(x) g1(x,W,tnsr1[i,],omega),
                             ui = rbind(W,-W),ci = rep(-alph,2*nrow(W)), method = "BFGS")$par})
    nA <- matrix(unlist(l),nrow = nrow(A),byrow = T)
  }
  return(nA)
}


corecomb = function(C,W,ttnsr,omega,alph=TRUE){
  Cvec <- c(C@data)
  h <- function(x) h1(x,W,ttnsr,omega)
  g <- function(x) g1(x,W,ttnsr,omega)
  if (alph==TRUE) {
    d <- optim(Cvec,h,g,method = "BFGS")
    C <- new("Tensor",C@num_modes,C@modes,data =d$par)
```

```
  }else{
    d <- constrOptim(Cvec,h,g,ui = rbind(W,-W),ci = rep(-alph,2*nrow(W)),
   method = "BFGS")
    C <- new("Tensor",C@num_modes,C@modes,data =d$par)
  }
  return(C)
}


## Algorithm 3.
fit_ordinal = function(ttnsr,C,A_1,A_2,A_3,omega,alph = TRUE){

  alphbound <- alph+10^-4
  result = list()
  error<- 3
  iter = 0
  d1 <- nrow(A_1); d2 <- nrow(A_2); d3 <- nrow(A_3)
  r1 <- ncol(A_1); r2 <- ncol(A_2); r3 <- ncol(A_3)
  if (alph == TRUE) {
    while ((error > 10^-4)&(iter<50) ) {
      iter = iter +1


      #update A_1
      prevtheta <- ttm(ttm(ttm(C,A_1,1),A_2,2),A_3,3)
      prev <- likelihood(ttnsr,prevtheta,omega)
      W1 =kronecker(A_3,A_2)%*%t(k_unfold(C,1)@data)
      A_1 <- comb(A_1,W1,ttnsr,1,omega)


      # update A_2
      W2 <- kronecker(A_3,A_1)%*%t(k_unfold(C,2)@data)
      A_2 <- comb(A_2,W2,ttnsr,2,omega)

      # update A_3
      W3 <- kronecker(A_2,A_1)%*%t(k_unfold(C,3)@data)
```

```
      A_3 <- comb(A_3,W3,ttnsr,3,omega)

      # update C
      W4 <- kronecker(kronecker(A_3,A_2),A_1)
      C <- corecomb(C,W4,ttnsr,omega)
      theta <- ttm(ttm(ttm(C,A_1,1),A_2,2),A_3,3)
      new <- likelihood(ttnsr,theta,omega)
      error <- abs((new-prev)/prev)
    }
  }else{
    while ((error > 10^-4)&(iter<50) ) {
      iter = iter +1


      #update A_1
      prevtheta <- ttm(ttm(ttm(C,A_1,1),A_2,2),A_3,3)
      prev <- likelihood(ttnsr,prevtheta,omega)
      W1 =kronecker(A_3,A_2)%*%t(k_unfold(C,1)@data)
      A_1 <- comb(A_1,W1,ttnsr,1,omega,alphbound)
      if(max(abs(ttm(ttm(ttm(C,A_1,1),A_2,2),A_3,3)@data))>=alph) break


      # update A_2
      W2 <- kronecker(A_3,A_1)%*%t(k_unfold(C,2)@data)
      A_2 <- comb(A_2,W2,ttnsr,2,omega,alphbound)
      if(max(abs(ttm(ttm(ttm(C,A_1,1),A_2,2),A_3,3)@data))>=alph) break

      # update A_3
      W3 <- kronecker(A_2,A_1)%*%t(k_unfold(C,3)@data)
      A_3 <- comb(A_3,W3,ttnsr,3,omega,alphbound)
      if(max(abs(ttm(ttm(ttm(C,A_1,1),A_2,2),A_3,3)@data))>=alph) break

      # update C
      W4 <- kronecker(kronecker(A_3,A_2),A_1)
      C <- corecomb(C,W4,ttnsr,omega,alph)
      theta <- ttm(ttm(ttm(C,A_1,1),A_2,2),A_3,3)
```

```r
      new <- likelihood(ttnsr,theta,omega)
      error <- abs((new-prev)/prev)
      if(max(abs(ttm(ttm(ttm(C,A_1,1),A_2,2),A_3,3)@data))>=alph) break
    }
  }

  result$C <- C; result$A_1 <- A_1; result$A_2 <- A_2; result$A_3 <- A_3
  result$iteration <- iter
  return(result)
}

## Algorithm 4.
fit_ordinal2 = function(ttnsr,C,A_1,A_2,A_3,omega=TRUE,alph = TRUE){
  omega <- sort(rnorm(2))
  alphbound <- alph+10^-4
  result = list()
  error<- 3
  iter = 0
  d1 <- nrow(A_1); d2 <- nrow(A_2); d3 <- nrow(A_3)
  r1 <- ncol(A_1); r2 <- ncol(A_2); r3 <- ncol(A_3)
  if (alph == TRUE) {
    while ((error > 10^-4)&(iter<50) ) {
      iter = iter +1


      #update A_1
      prevtheta <- ttm(ttm(ttm(C,A_1,1),A_2,2),A_3,3)
      prev <- likelihood(ttnsr,prevtheta,omega)
      W1 =kronecker(A_3,A_2)%*%t(k_unfold(C,1)@data)
      A_1 <- comb(A_1,W1,ttnsr,1,omega)


      # update A_2
      W2 <- kronecker(A_3,A_1)%*%t(k_unfold(C,2)@data)
      A_2 <- comb(A_2,W2,ttnsr,2,omega)

```

```
      # update A_3
      W3 <- kronecker(A_2,A_1)%*%t(k_unfold(C,3)@data)
      A_3 <- comb(A_3,W3,ttnsr,3,omega)

      # update C
      W4 <- kronecker(kronecker(A_3,A_2),A_1)
      C <- corecomb(C,W4,ttnsr,omega)

      #update omega
      theta <- ttm(ttm(ttm(C,A_1,1),A_2,2),A_3,3)
      m <- polr(as.factor(c(ttnsr))~offset(-c(theta@data)))
      omega <- m$zeta



      theta <- ttm(ttm(ttm(C,A_1,1),A_2,2),A_3,3)
      new <- likelihood(ttnsr,theta,omega)
      error <- abs((new-prev)/prev)
    }
  }else{
    while ((error > 10^-4)&(iter<50) ) {
      iter = iter +1


      #update A_1
      prevtheta <- ttm(ttm(ttm(C,A_1,1),A_2,2),A_3,3)
      prev <- likelihood(ttnsr,prevtheta,omega)
      W1 =kronecker(A_3,A_2)%*%t(k_unfold(C,1)@data)
      A_1 <- comb(A_1,W1,ttnsr,1,omega,alphbound)
      if(max(abs(ttm(ttm(ttm(C,A_1,1),A_2,2),A_3,3)@data))>=alph) break


      # update A_2
      W2 <- kronecker(A_3,A_1)%*%t(k_unfold(C,2)@data)
      A_2 <- comb(A_2,W2,ttnsr,2,omega,alphbound)
      if(max(abs(ttm(ttm(ttm(C,A_1,1),A_2,2),A_3,3)@data))>=alph) break
```

```r

      # update A_3
      W3 <- kronecker(A_2,A_1)%*%t(k_unfold(C,3)@data)
      A_3 <- comb(A_3,W3,ttnsr,3,omega,alphbound)
      if(max(abs(ttm(ttm(ttm(C,A_1,1),A_2,2),A_3,3)@data))>=alph) break

      # update C
      W4 <- kronecker(kronecker(A_3,A_2),A_1)
      C <- corecomb(C,W4,ttnsr,omega,alph)
      if(max(abs(ttm(ttm(ttm(C,A_1,1),A_2,2),A_3,3)@data))>=alph) break

      #update omega
      theta <- ttm(ttm(ttm(C,A_1,1),A_2,2),A_3,3)
      m <- polr(as.factor(c(ttnsr))~offset(-c(theta@data)))
      omega <- m$zeta


      theta <- ttm(ttm(ttm(C,A_1,1),A_2,2),A_3,3)
      new <- likelihood(ttnsr,theta,omega)
      error <- abs((new-prev)/prev)
    }
  }

  result$C <- C; result$A_1 <- A_1; result$A_2 <- A_2; result$A_3 <- A_3
  result$iteration <- iter; result$omega <- omega
  return(result)
}
```