Modified proof, new algorithm simulation and ordinal tensor simulation

Chanwoo Lee

Nov 12, 2019

1 Estimation accuracy for randomized SVD

Our goal is to estimate a low-rank tensor signal tensor from a noisy tensor observation. Let $\mathcal{A} \in \mathbb{R}^{d_1\times d_2\times d_3}$ be the signal tensor to be of interest. We assume the signal tensor admits a low-rank Tucker decomposition,

$$\mathcal{A} = \mathcal{C} \times_1 M^{(1)} \times_2 M^{(2)} \times_3 M^{(3)}$$

where $\mathcal{C} \in \mathbb{R}^{r_1\times r_2\times r_3}$ and $M^{(1)}, M^{(2)}, M^{(3)}$ are orthonormal matrices in $\mathbb{R}^{d_1\times r_1}, \mathbb{R}^{d_2\times r_2}, \mathbb{R}^{d_3\times r_3}$ respectively. We will suggest various methods to recover $\mathcal{A}$ from a noisy tensor

$$\mathcal{D} = \mathcal{A} + \mathcal{E}$$

where $\mathcal{E}$ is a noise tensor with i.i.d. entries from $N(0, \sigma^2)$.

We provide some theorems to guarantee the convergence of estimators to true signal tensor in each method. Theorem 1 and Theorem 2 show that the convergence of a tensor is ensured by the convergence of orthonormal matrices. The only difference between the two theorems is distance measure used for orthonormal matrices: Theorem 1 uses Frobenius norm while Theorem 2 uses angle to measure the error. Theorem 3 suggests sufficient conditions to recover a signal tensor from a noisy observation. From these theorems, the proposed algorithms enjoy asymptotic consistency. In section 1.1, we provide the theorems to derive a generic error bound. We guarantee consistency of estimation under some conditions. The section 1.2 presents some linear algebraic tools we need in section 1.1.

1.1 Theoretical results

We develop general error bounds for suggested algorithmic methods. The first two results show that the accuracy of tensor estimation is guaranteed by the accuracy of estimations for orthonormal matrices. The last result guarantees the consistency of estimation under some conditions.

Theorem 1. *Let $\mathcal{A} = C \times_1 M^{(1)} \times_2 M^{(2)} \times_3 M^{(3)} \in \mathbb{R}^{d_1 \times d_2 \times d_3}$, where $\mathcal{C} \in \mathbb{R}^{r_1 \times r_2 \times r_3}$ is a core tensor and $M^{(1)}, M^{(2)}, M^{(3)}$ are orthonormal matrices in $\mathbb{R}^{d_1 \times r_1}, \mathbb{R}^{d_2 \times r_2}, \mathbb{R}^{d_3 \times r_3}$ respectively. Suppose that an estimator $\hat{M}^{(i)}$ is close to $M^{(i)}$ in Frobenious norm, in that $\|M^{(i)} - \hat{M}^{(i)}\|_F \leq \epsilon$. for all $i \in \{1, 2, 3\}$*
Then, $\hat{\mathcal{A}} = \hat{\mathcal{C}} \times_1 \hat{M}^{(1)} \times_2 \hat{M}^{(2)} \times_3 \hat{M}^{(3)}$ where $\hat{C} = \mathcal{A} \times_1 (\hat{M}^{(1)})^T \times_2 (\hat{M}^{(2)})^T \times_3 (\hat{M}^{(3)})^T$ has an upper error bound,

$$\|\hat{\mathcal{A}} - \mathcal{A}\|_F \leq \|\mathcal{A}\|_F (2(\sqrt{r_1} + \sqrt{r_2} + \sqrt{r_3}) + 3\epsilon)\epsilon.$$

Furthermore, if we assume that an estimator $\hat{M}^{(i)}$ is close to $M^{(i)}$ in spectral norm, in that $\|M^{(i)} - \hat{M}^{(i)}\|_\sigma \leq \epsilon$ for all $i \in \{1, 2, 3\}$
Then, $\hat{\mathcal{A}}$ has an upper error bound,

$$\|\hat{\mathcal{A}} - \mathcal{A}\|_F \leq \|\mathcal{A}\|_F (2\|M^{(1)}\|_\sigma + 2\|M^{(2)}\|_\sigma + 2\|M^{(3)}\|_\sigma + 3\epsilon)\epsilon.$$

Proof. First, assume the Frobenius norm error bound. Then, we have,

$$\begin{aligned} \|M^{(i)}(M^{(i)})^T - \hat{M}^{(i)}(\hat{M}^{(i)})^T\|_F &= \|M^{(i)}(M^{(i)})^T - M^{(i)}(\hat{M}^{(i)})^T + M^{(i)}(\hat{M}^{(i)})^T - \hat{M}^{(i)}(\hat{M}^{(i)})^T\|_F \\ &= \|M^{(i)}((M^{(i)})^T - (\hat{M}^{(i)})^T) + (M^{(i)} - \hat{M}^{(i)})(\hat{M}^{(i)})^T)\|_F \\ &\leq (2\|M^{(i)}\|_F + \epsilon)\epsilon \quad \text{for all } i \in \{1, 2, 3\}. \end{aligned}$$

Let us define $P_A := AA^T$. Then, the above inequality can be denoted as,

$$\|P_{M^{(i)}} - P_{\hat{M}^{(i)}}\|_F \leq (2\|M^{(i)}\|_F + \epsilon)\epsilon. \tag{1}$$

The theorem's main inequality can be obtained as follows

$$\begin{aligned}
\|\mathcal{A}-\hat{\mathcal{A}}\|_F &= \|\mathcal{A}-\hat{\mathcal{C}}\times_1 \hat{M}^{(1)} \times_2 \hat{M}^{(2)} \times_3 \hat{M}^{(3)}\|_F \\
&= \|\mathcal{A}-\mathcal{A}\times_1 (\hat{M}^{(1)})^T \times_2 (\hat{M}^{(2)})^T \times_3 (\hat{M}^{(3)})^T \times_1 \hat{M}^{(1)} \times_2 \hat{M}^{(2)} \times_3 \hat{M}^{(3)}\|_F \\
&= \|\mathcal{A}-\mathcal{A}\times_1 P_{\hat{M}^{(1)}} \times_2 P_{\hat{M}^{(1)}} \times_3 P_{\hat{M}^{(1)}}\|_F \\
&= \|A_{(1)} - P_{\hat{M}^{(1)}}A_{(1)}(P_{\hat{M}^{(2)}}\otimes P_{\hat{M}^{(3)}})\|_F \\
&= \|P_{M^{(1)}}A_{(1)}(P_{M^{(2)}}\otimes P_{M^{(3)}}) - P_{\hat{M}^{(1)}}A_{(1)}(P_{\hat{M}^{(2)}}\otimes P_{\hat{M}^{(3)}})\|_F \\
&= \|(P_{M^{(1)}}-P_{\hat{M}^{(1)}})A_{(1)}(P_{M^{(2)}}\otimes P_{M^{(3)}}) + P_{\hat{M}^{(1)}}A_{(1)}(P_{M^{(2)}}\otimes P_{M^{(3)}} - P_{\hat{M}^{(2)}}\otimes P_{\hat{M}^{(3)}})\|_F \\
&\le \|(P_{M^{(1)}}-P_{\hat{M}^{(1)}})A_{(1)}(P_{M^{(2)}}\otimes P_{M^{(3)}})\|_F + \|P_{\hat{M}^{(1)}}A_{(1)}(P_{M^{(2)}}\otimes P_{M^{(3)}} - P_{\hat{M}^{(2)}}\otimes P_{\hat{M}^{(3)}})\|_F.
\end{aligned} \tag{2}$$

Based on Frobenius versions of the Proposition 1, Proposition 2 and Proposition 3 on (2), the following inequality holds true. We proceed the inequality,

$$\begin{aligned}
\|\mathcal{A}-\hat{\mathcal{A}}\|_F &\le \|P_{M^{(1)}}-P_{\hat{M}^{(1)}}\|_F\|A_{(1)}\|_F + \|A_{(1)}\|_F\|P_{M^{(2)}}\otimes P_{M^{(3)}} - P_{\hat{M}^{(2)}}\otimes P_{\hat{M}^{(3)}}\|_F \\
&= \big(\|P_{M^{(1)}}-P_{\hat{M}^{(1)}}\|_F + \|P_{M^{(2)}}\otimes P_{M^{(3)}} - P_{\hat{M}^{(2)}}\otimes P_{\hat{M}^{(3)}}\|_F\big)\|\mathcal{A}\|_F \\
&\le (\|P_{M^{(1)}}-P_{\hat{M}^{(1)}}\|_F + \|(P_{M^{(2)}}-P_{\hat{M}^{(2)}})\otimes P_{M^{(3)}}\|_F + \|P_{\hat{M}^{(2)}}\otimes(P_{M^{(3)}}-P_{\hat{M}^{(3)}})\|_F)\|\mathcal{A}\|_F \\
&\le (\|P_{M^{(1)}}-P_{\hat{M}^{(1)}}\|_F + \|P_{M^{(2)}}-P_{\hat{M}^{(2)}}\|_F + \|P_{M^{(3)}}-P_{\hat{M}^{(3)}}\|_F)\|\mathcal{A}\|_F.
\end{aligned}$$

Finally, we get the inequality of the theorem from (1).
Likewise, we can get the same result under the assumption of spectral norm bound. By the same approach for Frobenius norm case, we get,

$$\|M^{(i)}(M^{(i)})^T - \hat{M}^{(i)}(\hat{M}^{(i)})^T\|_\sigma \le (2\|M^{(i)}\|_\sigma + \epsilon)\epsilon. \tag{3}$$

Also, From the inequality (2)

$$\begin{aligned}
\|\mathcal{A}-\hat{\mathcal{A}}\|_F &\le \|(P_{M^{(1)}}-P_{\hat{M}^{(1)}})A_{(1)}(P_{M^{(2)}}\otimes P_{M^{(3)}})\|_F + \|P_{\hat{M}^{(1)}}A_{(1)}(P_{M^{(2)}}\otimes P_{M^{(3)}} - P_{\hat{M}^{(2)}}\otimes P_{\hat{M}^{(3)}})\|_F \\
&\le \|P_{M^{(1)}}-P_{\hat{M}^{(1)}}\|_\sigma\|A_{(1)}(P_{M^{(2)}}\otimes P_{M^{(3)}})\|_F + \|P_{\hat{M}^{(1)}}A_{(1)}\|_F\|P_{M^{(2)}}\otimes P_{M^{(3)}} - P_{\hat{M}^{(2)}}\otimes P_{\hat{M}^{(3)}}\|_\sigma \qquad (4) \\
&\le \big(\|P_{M^{(1)}}-P_{\hat{M}^{(1)}}\|_\sigma + \|P_{M^{(2)}}\otimes P_{M^{(3)}} - P_{\hat{M}^{(2)}}\otimes P_{\hat{M}^{(3)}}\|_\sigma\big)\|A_{(1)}\|_F \qquad (5) \\
&\le (\|P_{M^{(1)}}-P_{\hat{M}^{(1)}}\|_\sigma + (\|P_{M^{(2)}}-P_{\hat{M}^{(2)}}\|_\sigma + (\|P_{M^{(3)}}-P_{\hat{M}^{(3)}}\|_\sigma\big)\|A_{(1)}\|_F. \qquad (6)
\end{aligned}$$

Proposition 3 is used in (4). Proposition 2 and 3 are used in (5) and (6). Finally, the main inequality of the theorem is derived from (3) and (6). $\square$

We define the principal angle to measure the distance between two subspaces.

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

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

A principal angle between two matrices is defined as

$$\cos\theta(A, B) = \cos\theta(\text{span}(A), \text{span}(B)),$$

where $A \in \mathbb{R}^{n\times m}, B \in \mathbb{R}^{n\times k}$ are two matrices.

If we use the principal angle to measure the error, the following Theorem 2 holds.

Theorem 2. *Under the same condition as in Theorem 1 but instead of the Frobenius norm or spectral norm error bound, we assume*

$$\sin(\theta(M^{(i)}, \hat{M}^{(i)})) = \sqrt{1 - \cos^2(\theta(M^{(i)}, \hat{M}^{(i)}))} < \epsilon.$$

Then, the following error bound holds

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

Proof. If the inequality $\|P_{M^{(i)}} - P_{\hat{M}^{(i)}}\|_\sigma \leq 2\epsilon$ is true, the main inequality holds from the last inequality of the proof in Theorem 1.

$$\|\mathcal{A} - \hat{\mathcal{A}}\|_F \leq (\|P_{M^{(1)}} - P_{\hat{M}^{(1)}}\|_\sigma + \|P_{M^{(2)}} - P_{\hat{M}^{(2)}}\|_\sigma + \|P_{M^{(3)}} - P_{\hat{M}^{(3)}}\|_\sigma)\|\mathcal{A}\|_F \leq 6\epsilon\|\mathcal{A}\|_F.$$

So it suffices to show $\|P_{M^{(i)}} - P_{\hat{M}^{(i)}}\|_\sigma \leq 2\epsilon$ under the given condition.

$$
\begin{aligned}
&\|P_{M^{(i)}} - P_{\hat{M}^{(i)}}\|_\sigma \\
&= \|M^{(i)}(M^{(i)})^T - \hat{M}^{(i)}(\hat{M}^{(i)})^T\|_\sigma \\
&= \|M^{(i)}(M^{(i)})^T - M^{(i)}(M^{(i)})^T\hat{M}^{(i)}(\hat{M}^{(i)})^T + M^{(i)}(M^{(i)})^T\hat{M}^{(i)}(\hat{M}^{(i)})^T - \hat{M}^{(i)}(\hat{M}^{(i)})^T\|_\sigma \\
&\leq \|M^{(i)}(M^{(i)})^T - M^{(i)}(M^{(i)})^T\hat{M}^{(i)}(\hat{M}^{(i)})^T\|_\sigma + \|M^{(i)}(M^{(i)})^T\hat{M}^{(i)}(\hat{M}^{(i)})^T - \hat{M}^{(i)}(\hat{M}^{(i)})^T\|_\sigma \\
&= \|M^{(i)}(M^{(i)})^T(I - \hat{M}^{(i)}(\hat{M}^{(i)})^T)\|_\sigma + \|(M^{(i)}(M^{(i)})^T - I)\hat{M}^{(i)}(\hat{M}^{(i)})^T\|_\sigma \\
&\leq \sin(\theta) + \sin(\theta) \leq 2\epsilon.
\end{aligned}
$$

The last inequality follows from Property 5. □

Next theorem guarantees the angle-wise convergence of $\hat{M}^{(i)}$ to $M^{(i)}$ under certain conditions. Thereby, we have a consistent estimator $\hat{\mathcal{A}}$ to the signal tensor $\mathcal{A}$.

Theorem 3. *Let $\mathcal{A} = \mathcal{C} \times_1 M^{(1)} \times_2 M^{(2)} \times_3 M^{(3)}$ be a signal tensor, where $\mathcal{C} \in \mathbb{R}^{r_1 \times r_2 \times r_3}$ is a core tensor and $M^{(1)}, M^{(2)}, M^{(3)}$ are orthonormal matrices in $\mathbb{R}^{d_1 \times r_1}, \mathbb{R}^{d_2 \times r_2}, \mathbb{R}^{d_3 \times r_3}$ respectively.*
Let $\mathcal{D} = \mathcal{A} + \mathcal{E}$ be a noisy tensor with a noise tensor $\mathcal{E}$ with i.i.d. entries from $N(0, \sigma^2)$ Suppose, $s_{min}(C_{(i)}) >> \sigma\sqrt{\max(d_i, \frac{d_1d_2d_3}{d_i})\frac{d_1d_2d_3}{d_ir_i}}$ as $d_1, d_2, d_3 \to \infty$, where $s_{min}(C_{(i)})$ is the smallest singular value of $C_{(i)}$.
Then, the following holds true.

$$\cos\theta(M^{(i)}, \hat{M}^{(i)}) \to 1 \textit{ in probability for } i \in [3].$$

$$\|\mathcal{A} - \hat{\mathcal{A}}\|_F \to 0 \textit{ in probability .}$$

where $(\hat{\mathcal{C}}, \hat{M}^{(1)}, \hat{M}^{(2)}, \hat{M}^{(3)})$ is an output of Algorithm 1 at the input $\mathcal{D}$ and $\mathcal{A}$ is an estimator such that $\hat{\mathcal{A}} = \hat{\mathcal{C}} \times_1 \hat{M}^{(1)} \times_2 \hat{M}^{(2)} \times_3 \hat{M}^{(3)}$.

Proof. It suffices to show when $i = 1$. Notice,

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

Define $B = (M^{(3)} \otimes M^{(2)})^T = \left(M_1^{(3)} \otimes M_1^{(2)}, \quad M_1^{(3)} \otimes M_2^{(2)}, \quad \cdots, \quad M_{r_3}^{(3)} \otimes M_{r_2}^{(2)}\right)$ where $M_j^{(i)}$

is the j-th column of $M^{(i)}$. the step A of Algorithm 1 generates a test random Gaussian matrix Ω and captures the image space of unfolded matrix $A_{(1)}$. Having this procedure in mind, we obtain,

$$\begin{aligned} A_{(1)}\Omega &= M^{(1)}C_{(1)}(M^{(3)} \otimes M^{(2)})^T\Omega \\ &= M^{(1)}C_{(1)}B\Omega \quad \text{ where } \Omega \in R^{d_2 d_3 \times r_1} \end{aligned}$$

Lemma 1. *In the proof of Theorem 1, all entries of* $\mathbf{Z} = \begin{pmatrix} Z_1, & Z_2, & \cdots & , Z_{r_1} \end{pmatrix}$ *is from i.i.d* $.N(0,1)$

Proof. To remind the definitions,

$$Z_i^T = \left(\sum_{k=1}^{d_2 d_3} B_{1,k}\Omega_{k,i}, \quad \cdots, \quad \sum_{k=1}^{d_2 d_3} B_{r_2 r_3,k}\Omega_{k,i}\right)^T = (z_{1,i}, \cdots, z_{r_1 r_2, i})^T.$$

$$B = (M^{(3)} \otimes M^{(2)})^T = \left(M_1^{(3)} \otimes M_1^{(2)}, \quad M_1^{(3)} \otimes M_2^{(2)}, \quad \cdots, \quad M_{r_3}^{(3)} \otimes M_{r_2}^{(2)}\right).$$

Notice that Z_i and Z_j are independent when $i \neq j$ because entries of Z_i only consist of i-th column of Ω. Therefore, it suffices to show entries of Z_1 are independent and from $N(0,1)$.

1. $z_{1,1} \sim N(0,1)$.
   Notice $z_{1,1} = \sum_{k=1}^{d_2 d_3} B_{1,k}\Omega_{k,1} = [B^1]^T\Omega_1$ where $B^1 \overset{def}{=}$ 1st row of B.
   Therefore, $z_{1,1}$ is from $N(0,1)$ because $\|B^1\| = 1$ and $\Omega_1 \overset{i.i.d}{\sim} N(0,1)$.

2. $z_{1,1}, \cdots, z_{r_2 r_3, 1}$ are independent.
   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 entries of $\mathbf{Z}$ is from i.i.d. $N(0,1)$ by 1,2 □

We can apply Theorem 3 to the lower dimension case.

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

$$\hat{a} = D\Omega, \text{ where } \Omega = (z_1, \cdots, z_{d_2})^T$$

Proof. Taking $\mathcal{C} = \lambda, M^{(1)} = a, M^{(2)} = b$ and $M^{(3)} = 1$ in Theorem 3. yields the result of Corollary 1. □

1.2 Some background from linear algebra

We collect linear algebraic background in this section.

Proposition 1 (Kronecker Product Norms). *Let $\|\cdot\|_F$ be Frobenious norm and $\|\cdot\|_\sigma$ be spectral norm. For any $A \in \mathbb{R}^{m\times n}, B \in \mathbb{R}^{n\times k}$*

$$\|A \otimes B\|_F = \|A\|_F \|B\|_F, \quad \|A \otimes B\|_\sigma = \|A\|_\sigma \|B\|_\sigma.$$

Proof. From the definition of Frobenius norm and Kronecker product, we can get

$$\|A \otimes B\|_F = \| \begin{bmatrix} a_{11}B & \cdots & a_{mn}B \\ \vdots & \cdots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix} \|_F = \|A\|_F \|B\|_F.$$

For the case of spectral norm, Let $A = \sum_i \sigma_i u_i v_i^T$ and $B = \sum_j \lambda_j x_j y_j^T$ be the singular value decomposition of two matrices. Then,

$$\begin{aligned} A \otimes B &= \sum_{i,j} \sigma_i \lambda_j (u_i v_i^T) \otimes (x_j y_j^T) \\ &= \sum_{i,j} \sigma_i \lambda_j (u_i \otimes x_j)(v_i^T \otimes y_j^T) \\ &= \sum_{i,j} \sigma_i \lambda_j (u_i \otimes x_j)(v_i \otimes y_j)^T. \end{aligned}$$

From this form, we can see $\{\sigma_i \lambda_j : i, j\}$ are singular values of $A \otimes B$. Therefore,

$$\|A \otimes B\|_\sigma = \max_{i,j} \sigma_i \lambda_j = (\max_i \sigma_i)(\max_j \lambda_j) = \|A\|_\sigma \|B\|_\sigma.$$

□

Proposition 2 (Norm of Projection Matrices)**.** *Let $P \in \mathbb{R}^{m\times n}$ be a projection matrix and $A \in \mathbb{R}^{n\times k}$ be an arbitrary matrix. Then,*

$$\|PA\|_F \leq \|A\|_F, \quad \|PA\|_\sigma \leq \|A\|_\sigma.$$

Proof. Notice for $x \in \mathbb{R}^n$, $\quad \|x\|_2^2 = \|Px\|_2^2 + \|(I-P)x\|_2^2$. Therefore, we have the following inequality

$$\|Px\|_2 \leq \|x\|_2.$$

This proves the main inequalities of proposition 2. □

Proposition 3 (Frobenius Norm of Matrix Product)**.** *Let $A \in \mathbb{R}^{m\times n}, B \in \mathbb{R}^{n\times k}$ be arbitrary matrices. Then*

$$\|AB\|_F \leq \|A\|_F\|B\|_F, \quad \|AB\|_F \leq \|A\|_F\|B\|_\sigma.$$

Proof. The submultiplicity of Frobenius norm is an application of Cauchy-Schwarz inequality. So we focus on proving $\|AB\|_F \leq \|A\|_F\|B\|_\sigma$ here. Let A_i denote i-th row of A. Then,

$$\|AB\|_F^2 = \sum_i \|(AB)_i\|_\sigma^2 = \sum_i \|A_iB\|_\sigma^2 \leq \sum_i \|A_i\|_F^2\|B\|_\sigma^2 = \|A\|_F^2\|B\|_\sigma^2.$$

□

The following 2 propositions offer a way to relate the principal angle and spectral norm.

Proposition 4. *Let $P_\mathcal{F}$ and $P_\mathcal{G}$ be the orthogonal projectors from $\mathbb{R}^d$ onto subspaces $\mathcal{F}$ and $\mathcal{G}$, respectively, then*

$$\cos\theta = \|P_\mathcal{F}P_\mathcal{G}\|_\sigma = \|P_\mathcal{G}P_\mathcal{F}\|_\sigma.$$

Proof. For vectors $x, y \in \mathbb{R}^d$ such that $\|x\|_2 = \|y\|_2 = 1$, we have $P_\mathcal{F}x \in \mathcal{F}$ and $P_\mathcal{G}y \in \mathcal{G}$. Then,

$$\cos\theta = \max_{u\in\mathcal{F}, v\in\mathcal{G}, \|u\|=\|v\|=1} v^t u = \max_{u\in\mathcal{F}, v\in\mathcal{G}, \|u\|\leq 1\|v\|\leq 1} v^t u = \max_{\|x\|\leq 1\|y\|\leq 1} y^t P_\mathcal{F}P_\mathcal{G}x = \|P_\mathcal{F}P_\mathcal{G}\|_\sigma.$$

□

Proposition 5. *Under the same condition as in Proposition 4,*

$$\|P_{\mathcal{F}}(I - P_{\mathcal{G}})\|_{\sigma} \leq \sin(\theta).$$

Proof.

$$\begin{aligned}\|P_{\mathcal{F}}(I - P_{\mathcal{G}})\|_{\sigma}^2 &= \max_{u \in \mathbb{R}^d, \|u\|=1} u^t P_{\mathcal{F}}(I - P_{\mathcal{G}})u = \max_{u \in \mathbb{R}^d, \|u\|=1} u^t P_{\mathcal{F}} u - u^t P_{\mathcal{F}} P_{\mathcal{G}} u \\ &\leq 1 - \|P_{\mathcal{F}} P_{\mathcal{G}}\|_{\sigma}^2 = \sin^2(\theta).\end{aligned}$$

□

Proposition 6 (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,*

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

2 New randomized Tucker decomposition algorithm and simulations

2.1 Algorithm 2.

The Algorithm 1 has expensive computation cost to generate random test matrices. The algorithm 2 suggests another way to overcome this drawback. The only difference between two algorithms is the generation of random test matrices in Step A. For each unfolded matrix $A_{(n)}$, the algorithm 1 generates a $\prod_{i \neq n} d_i \times r_n$ random Gaussian matrix. However, the algorithm 2 draws $N - 1$ Gaussian matrices such that $P_i \in \mathbb{R}^{d_i \times r_n}$ for $i \in [N]/\{n\}$. Taking Khatri Rao product on all drawn matrices, Algorithm 2 gets a same sized random matrix with Algorithm 1. The detailed algorithm 3 is as follows.

Algorithm 1 Randomized tensor SVD 3

```
procedure SVD($\mathcal{A}$)
    Step A: Randomized SVD of $\mathcal{A}$.
    for $n \leftarrow 1 : N$ do
        Unfold $\mathcal{A}$ into $A_{(n)}$.
        Generate Gaussian test matrices $P_i \in \mathbb{R}^{d_i \times r_n}$ for all $i = 1, \cdots, n-1, n+1, \cdots, N$ with i.i.d. entries from $N(0,1)$.
        Get $\Omega^{(n)} = P_1 \odot \cdots \odot P_{n-1} \odot P_{n+1} \odot \cdot \odot P_N$ .
        Define $Y^{(n)} = A_{(n)}\Omega^{(n)}$.
        Construct a matrix $\hat{M}^{(n)}$ whose columns form an orthonormal basis for the range of $Y^{(n)}$.
    Obtain $\hat{\mathcal{C}} = \mathcal{A} \times_1 (\hat{M}^{(1)})^T \times_2 (\hat{M}^{(2)})^T \cdots \times_N (\hat{M}^{(N)})^T$.
    return $(\hat{\mathcal{C}}, \hat{M}^{(1)}, \cdots, \hat{M}^{(N)})$
```

Algorithm 3 modifies the algorithm 2 to reduce variability. In this algorithm, we take averaged projected space in terms of angles. Thereby, we can get reduced variability of the image space from repeated projections. The following describes the detailed procedure of the algorithm.

Algorithm 2 Randomized tensor SVD 4

```
procedure SVD($\mathcal{A}$)
    Step A: Randomized SVD of $\mathcal{A}$.
    for $n \leftarrow 1 : N$ do
        Unfold $\mathcal{A}$ into $A_{(n)}$.
        for $r \leftarrow 1 : R$ do
            Generate Gaussian test matrices $P_i^r \in \mathbb{R}^{d_i \times r_n}$ for all $i = 1, \cdots, n-1, n+1, \cdots, N$ with i.i.d. entries from $N(0,1)$.
            Get $\Omega^{(n)} = P_1^r \odot \cdots \odot P_{n-1}^r \odot P_{n+1}^r \odot \cdot \odot P_N^r$.
            Define $Y_r^{(n)} = A_{(n)}\Omega^{(n)}$.
        Construct a matrix $\hat{M}^{(n)}$ whose columns form $r_n$ largest left singular vectors for the range of $(Y_1^{(n)}, \cdots, Y_R^{(n)})$.
    Obtain $\hat{\mathcal{C}} = \mathcal{A} \times_1 (\hat{M}^{(1)})^T \times_2 (\hat{M}^{(2)})^T \cdots \times_N (\hat{M}^{(N)})^T$.
    return $(\hat{\mathcal{C}}, \hat{M}^{(1)}, \cdots, \hat{M}^{(N)})$.
```

2.2 Simulation.

I compared 4 various randomized Tucker decomposition algorithms:

1. First method: Unstructured Gaussian test matrix + Single Random Projection.

2. Second method: Unstructured Gaussian test matrix + Random Projection from tensor onto tensor.

3. Third method: Khatri-Rao Gaussian test matrix + Single Random Projection.

4. Fourth method: Khatri-Rao Gaussian test matrix + Multiple Random Projections.

The first simulation investigate the accuracy of each method. We consider an order-3 dimension $(100, 100, 100)$ signal tensor B. All entries of B are i.i.d. drawn from $\mathrm{N}(0, 1)$. The tensor D is generated by adding a noise tensor $\mathcal{E}$ to the signal tensor B, where $\mathcal{E}$ consists of i.i.d $\mathrm{N}(0, \sigma^2)$. To be specific, $\mathcal{D} = \mathcal{B} + \mathcal{E}$. We vary the noise level $\sigma \in \{0.01, 0.02, \cdots 0.49, 0.5\}$ and estimated B using various methods with target rank 20. Figure 1 shows the deviation from the true signal tensor. While methods 1-3 have the similar performances, the method 4 outperforms other methods across all noise sizes.

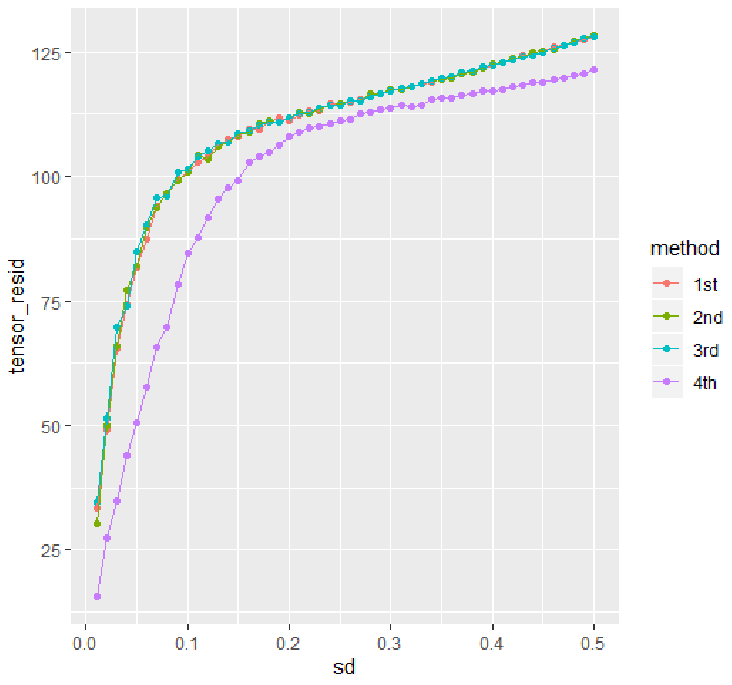


Figure 1: The y axis represents the Frobenius norm between the true signal and estimator. The x axis means the noise size.

The second simulation investigates the accuracy of estimators in terms of angles and MSE. We consider an order-3 dimension $(100, 100, 100)$ signal tensor B. We assume B has Tucker decomposition as $B = a \otimes b \otimes c$, where $a, b, c \in \mathbb{R}^{100}$ are the signal vectors. All entries of a, b, c are i.i.d. drawn from $\mathrm{N}(0, 1)$. We vary the noise level $\sigma \in \{0.01, 0.02, \cdots 0.49, 0.5\}$. We use target rank 1 and estimate the signal vectors according to each algorithms. We compare the angles between the true signal vectors and estimators. Figure 2 shows that 4-th method outperforms the other methods.

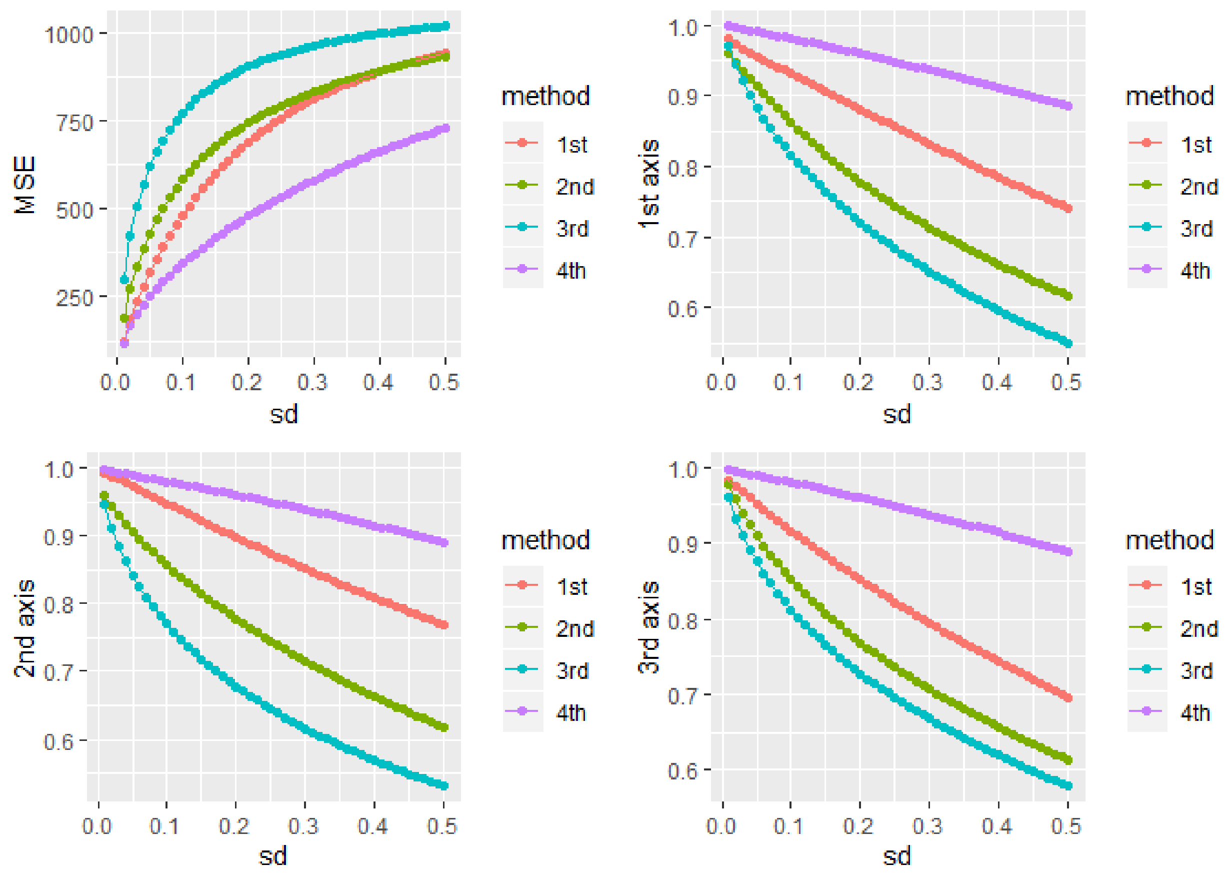


Figure 2: Each y axis means $\cos\theta(a,\hat{a})$, $\cos\theta(b,\hat{b})$ and $\cos\theta(c,\hat{c})$. We have consistent simulation results: Method 4 has the best performance in all respects. Method 1, Method 2 and Method 3 are followed in order.

3 Ordinal tensor model simulation

We propose to estimate the unknown parameter tensor Θ using a constrained likelihood. The loss function from log-likelihood is

$$\mathcal{L}_{\mathcal{Y}}(\Theta,\boldsymbol{\omega}) = -\sum_{i_1,\cdots,i_N}\Big[\sum_{l=1}^{K}\mathbb{1}(y_{i_1,\cdots,i_N}=l)\log(\pi_l(\theta_{i_1,\cdots,i_N},\boldsymbol{\omega}))\Big]. \tag{8}$$

where, $\pi_l(\theta_{i_1,\cdots,i_N},\boldsymbol{\omega}) = \text{logit}(\omega_l+\theta_{i_1,\cdots,i_N}) - \text{logit}(\omega_{l-1}+\theta_{i_1,\cdots,i_N})$ for $l\in[K]$. This loss minimization problem becomes the constrained optimization from Tucker decomposition assumption:

$$\underset{\Theta\subset\mathcal{D}}{\arg\min}\, \mathcal{L}_{\mathcal{Y}}(\Theta,\omega), \tag{9}$$
$$\text{where } \mathcal{D}=\{\Theta=\mathcal{C}\times_1 A_1\cdots\times_N A_N : \mathcal{C}\in\mathbb{R}^{\otimes_{i=1}^{N} r_i}, A_i\in\mathbb{R}^{d_i\times r_i}, \|\Theta\|_\infty\leq\alpha\}.$$

We assume that the search space $\mathcal{D}$ is a compact set containing the true parameter Θ_{true}. In this section, we describe how to solve the optmization problem (9). Afterward, we simulate our method to estimate the performance.

3.1 Algorithm construction

We introduce an algorithm to solve (9). The objective function $L_{\mathcal{Y}}(\Theta,\boldsymbol{\omega})$ is convex in $(\Theta,\boldsymbol{\omega})$. However, the feasible set $\mathcal{D}$ is non-convex. We utilized a formulation of tucker decomposition, and turn the optimization into a block wise convex problem. We divide cases into two: First, we know bin boundary $\omega_1,\cdots,\omega_K$. Second, we have no information about bin boundary.

3.1.1 Known Bin Boundary

From previous data or experience, we may have knowledge about bin boundary parameter ω. In this section we only consider this case. To get a minimizer for (9), we represent Θ into several blocks using Tucker decomposition. Specifically, write Θ as a function of N-mode matrices and a core tensor

$$\Theta=g(\mathcal{C},A_1,\cdots,A_N)=\mathcal{C}\times_1 A_1\cdots\times_N A_N. \tag{10}$$

Then the problem (9) can be rephrased as

$$\underset{(\mathcal{C},A_1,\cdots,A_N)}{\arg\min}\ \mathcal{L}_{\mathcal{Y}}(g(\mathcal{C},A_1,\cdots,A_N),\boldsymbol{\omega}) \text{ subject to } g(\mathcal{C},A_1,\cdots,A_N)\leq\alpha. \tag{11}$$

The objective function in (11) is convex in each block individually with all other blocks fixed. This feature enables us to update each block at a time while others being fixed. To see what happens in each iteration, let us denote $A_i^{(t)}$ as the i-th mode matrix in Tucker decomposition at t-th iteration and $\text{vec}(\cdot)$ as the operator that turns a tensor or a matrix into a vector. Then, we have an equivalent problem with (11).

$$\operatorname{vec}(A_i^{(t+1)}) = \underset{A_i \in \mathbb{R}^{d_i \times r_i}}{\arg\min}\, \mathcal{L}_{\mathcal{Y}}\Big(\big((A_N^{(t)} \otimes \cdots \otimes A_{i+1}^{(t)} \otimes A_{i-1}^{(t+1)} \otimes \cdots \otimes A_1^{(t+1)})(C_{(i)}^{(t)})^T \otimes I_{d_i}\big) \operatorname{vec}(A_i^{(t)}), \boldsymbol{\omega} \Big), \tag{12}$$

$$\text{subject to } g(\mathcal{C}^{(t)}, A_1^{(t+1)}, \cdots, A_{i-1}^{(t+1)}, A_i, A_{i+1}^{(t)}, \cdots, A_N^{(t)}) \le \alpha.$$

After updating all matrices for each iteration from (12), we update a core tensor as follows.

$$\operatorname{vec}(\mathcal{C}^{(t+1)}) = \underset{\mathcal{C} \in \mathbb{R}^{\otimes_{i=1}^N r_i}}{\arg\min}\, \mathcal{L}_{\mathcal{Y}}\Big(\big(A_N^{(t+1)} \otimes \cdots \otimes A_1^{(t+1)}\big) \operatorname{vec}(\mathcal{C}^{(t)}) \Big), \tag{13}$$

$$\text{subject to } g(\mathcal{C}, A_1^{(t+1)}, \cdots, A_N^{(t+1)}) \le \alpha.$$

We repeat this procedure until it converges. Quasi-Newton method, "BFGS",is used to update in each iteration. The full algorithm is described in Algorithm 3.

Algorithm 3 Ordinal tensor optimization with known boundary $\boldsymbol{\omega}$

```
    **Input:** Initializers , $\mathcal{C}^0 \in \mathbb{R}^{r_1 \times \cdots \times r_N}, A_1^0 \in \mathbb{R}^{d_1 \times r_1}, \cdots, A_N^0 \in \mathbb{R}^{d_N \times r_N}$.
    **Output:** Optimizor of $\mathcal{L}_Z(\boldsymbol{\omega}, \Theta)$ given $\boldsymbol{\omega}$.
for t = 1,2,$\cdots$, do until convergence,
    Update $A_n$
    for n = 1,2,$\cdots$,N do
        $\Theta_{(n)} = A_n^t \mathcal{C}_{(n)}^t \big(A_{n+1}^t \otimes \cdots \otimes A_N^t \otimes A_1^{t+1} \otimes \cdots \otimes A_{n-1}^{t+1}\big)^T$.
        $\operatorname{vec}(\Theta_{(n)}) = \Big( \big(A_{n+1}^t \otimes \cdots \otimes A_N^t \otimes A_1^{t+1} \otimes \cdots \otimes A_{n-1}^{t+1}\big)(C_{(n)}^t)^T \otimes I_{d_n} \Big) \operatorname{vec}(A_n^t)$.
        $\operatorname{vec}(A_n^{t+1}) = \arg\max(\mathcal{L}_{\mathcal{Y}}(\alpha, \operatorname{vec}(\Theta_{(n)})))$.
        Get $A_n^{t+1}$.
    Update $\mathcal{C}$
    $\Theta_{(1)} = A_1^{t+1} \mathcal{C}_{(1)}^t \big(A_N^{t+1} \otimes \cdots \otimes A_2^{t+1}\big)^T$.
    $\operatorname{vec}(\Theta_{(1)}) = \big(A_N^{t+1} \otimes \cdots \otimes A_1^{t+1}\big) \operatorname{vec}(C_{(1)}^t)$.
    $\operatorname{vec}(C_{(1)}^t) = \arg\max(\mathcal{L}_{\mathcal{Y}}(\boldsymbol{\omega}, \operatorname{vec}(\Theta_{(1)})))$.
    Get $\mathcal{C}^{t+1}$.
return $\Theta$
```

3.1.2 Unknown Bin Boundary

Having information about bin boundary rarely happens. If we have no knowledge on bin boundary, $\boldsymbol{\omega}$ becomes unknown parameter to estimate. Therefore, we modify to add one more line on algorithm 3 to update $\boldsymbol{\omega}$. Specifically, we update $\boldsymbol{\omega}$ in each iteration as follows

$$\boldsymbol{\omega}^{t+1} = \underset{\boldsymbol{\omega} \in R^K}{\arg\min} \mathcal{L}_{\mathcal{Y}}(\mathcal{C}^{t+1} \times_1 A_1^{(t+1)} \cdots \times A_N^{t+1}, \boldsymbol{\omega}) \text{ subject to } \omega_1 < \cdots < \omega_K = \infty. \tag{14}$$

Detailed algorithm is in algorithm 4.

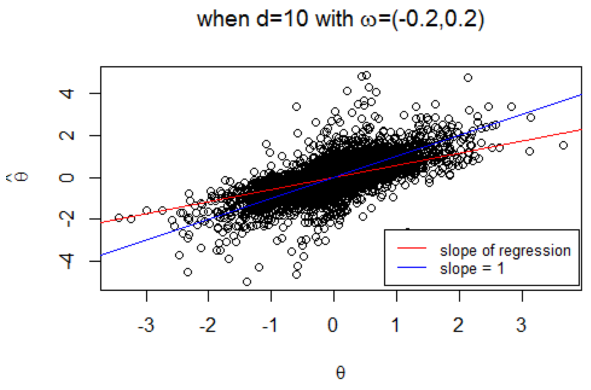
Algorithm 4 Ordinal tensor optimization with unknown boundary $\boldsymbol{\omega}$

```
    Input:  Initializers, $\mathcal{C}^0 \in \mathbb{R}^{r_1 \times \cdots \times r_N}, A_1^0 \in \mathbb{R}^{d_1 \times r_1}, \cdots, A_N^0 \in \mathbb{R}^{d_N \times r_N}$.
    Output: Optimizor of $\mathcal{L}_Z(\boldsymbol{\omega}, \Theta)$ given $\boldsymbol{\omega}$.
for t = 1,2,$\cdots$, do until convergence,
    Update $A_n$
    for n = 1,2,$\cdots$,N do
        $\Theta_{(n)} = A_n^t \mathcal{C}_{(n)}^t \left(A_{n+1}^t \otimes \cdots \otimes A_N^t \otimes A_1^{t+1} \otimes \cdots \otimes A_{n-1}^{t+1}\right)^T$.
        $\text{vec}(\Theta_{(n)}) = \left(\left(A_{n+1}^t \otimes \cdots \otimes A_N^t \otimes A_1^{t+1} \otimes \cdots \otimes A_{n-1}^{t+1}\right)(C_{(n)}^t)^T \otimes I_{d_n}\right) \text{vec}(A_n^t)$.
        $\text{vec}(A_n^{t+1}) = \arg\max(\mathcal{L}_{\mathcal{Y}}(\boldsymbol{\omega}^t, \text{vec}(\Theta_{(n)})))$.
        Get $A_n^{t+1}$
    Update $\mathcal{C}$
    $\Theta_{(1)} = A_1^{t+1} \mathcal{C}_{(1)}^t \left(A_N^{t+1} \otimes \cdots \otimes A_2^{t+1}\right)^T$.
    $\text{vec}(\Theta_{(1)}) = \left(A_N^{t+1} \otimes \cdots \otimes A_1^{t+1}\right) \text{vec}(C_{(1)}^t)$.
    $\text{vec}(C_{(1)}^t) = \arg\max(\mathcal{L}_{\mathcal{Y}}(\boldsymbol{\omega}^t, \text{vec}(\Theta_{(1)})))$.
    Get $\mathcal{C}^{t+1}$
    Update $\boldsymbol{\omega}$
    $\boldsymbol{\omega}^{t+1} = \arg\max(\mathcal{L}_{\mathcal{Y}}(\boldsymbol{\omega}, \Theta^{t+1}))$.
return $\boldsymbol{\omega}, \Theta$
```

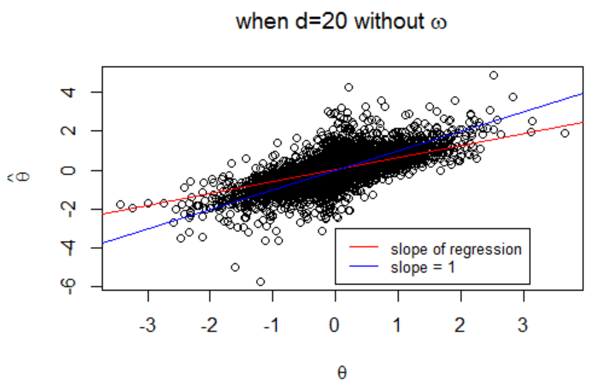
3.2 Simulations.

In this section, we investigate the performance of our method when the data indeed follows the Tucker decomposition tensor model. We set an order-3 dimension (d, d, d) ordinal tensor

$\mathcal{Y}$ with $K = 3$ from the model, where $\boldsymbol{\omega}_{true} = (-0.2, 0.2)$ and $\Theta_{true} = \mathcal{C} \times_1 A_1 \times_2 A_2 \times_3 A_3$ with rank of $\mathcal{C}$ being $(3, 3, 3)$. The entries of $\mathcal{C}$ and A_i are i.i.d. drawn from Uniform$[-1, 1]$. We get $\|\Theta\|_\infty = 2.507$ when $d = 20$ and $\|\Theta\|_\infty = 3.049$ when $d = 30$ for this simulation. We vary the tensor dimension d $\in \{20, 30\}$ and implemented algorithm 3 and algorithm 4. Figure 3 shows the scatter plot between Θ_{true} and $\hat{\Theta}_{\text{MLE}}$ when $d = 20$. Figure 3 is scatter plot between Θ_{true} and $\hat{\Theta}_{\text{MLE}}$ when $d = 30$. A larger tensor size has smaller deviation from true parameter. In addition, Algorithm 4 successfully estimated true bin boundary $\boldsymbol{\omega}_{\text{true}} = (-0.2, 0.2)$ regardless of tensor size. We obtain $\hat{\boldsymbol{\omega}}$ = (-0.198,0.201) when $d = 20$ and $\hat{\boldsymbol{\omega}}$ = (-0.211,0.203) when $d = 30$.

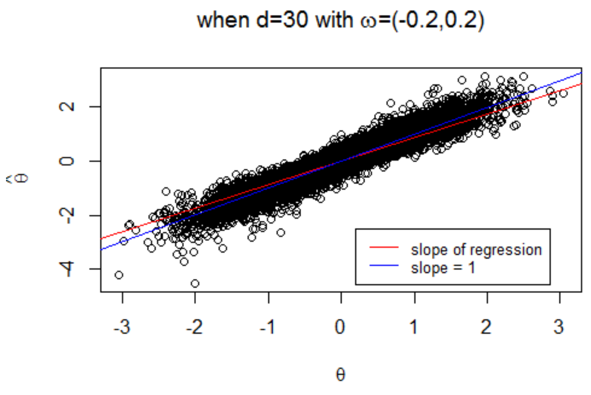


(a) When we know bin boundary ω

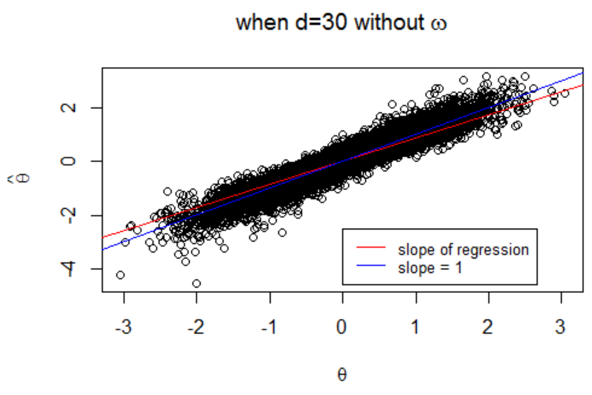


(b) When we don't know bin boundary ω

Figure 3: When d = 20. Red lines are slopes of ordinary least square estimators. Blue lines are line of $y = x$. The slope in left and right figures are 0.68 and 0.691 respectively.



(a) When we know bin boundary ω



(b) When we don't know bin boundary ω

Figure 4: When d = 30. Red lines are slopes of ordinary least square estimators. Blue lines are line of $y = x$. Left figure has the slope = 0.867366 and right figure has the slope = 0.86410

4 Moderate Sample Size for Simuation 3.2

Empirically, it is known that

$$\|\Theta_{\text{true}} - \hat{\Theta}\|_F \sim \sqrt{\frac{df_{\text{parameter}}}{n_{\text{sample}}}}, \tag{15}$$

where $df_{\text{parameter}}$ is the degree of freedom of the parameter space and n_{sample} is the sample size. The asymptotics (15) implies that it is helpful to error analysis to check the degree of freedom of the parameters and the number of samples. Suppose the parameter space is

$$\mathcal{D} = \{\Theta : \mathcal{C} \times_1 A_1 \times_2 A_2 \times_3 A_3, \text{ where } \mathcal{C} \in \mathbb{R}^{r\times r\times r}, \text{ and } A_i \in \mathbb{R}^{d\times r} \text{ for } i \in [3]\}.$$

The paper [1] shows that Tucker decomposition is unique. Low ranked version of the theorem is follows.

Theorem 4 (N-th order SVD)**.** *Every complex* $(I_1 \times \cdots \times I_N)$ *tensor* $\mathcal{A}$ *can be written as the product*

$$\mathcal{A} = S \times_1 U^{(1)} \times_2 U^{(2)} \cdots \times_N U^{(N)}.$$

in which

1. *$U^{(n)} = (U_1^{(n)} U_2^{(n)} \cdots U_{I_n}^{(n)})$ is a unitary $I_n \times r_n$ matrix.*

2. *S is an $(r_1 \times \cdots \times r_N)$ tensor of which the subtensors $S_{i_n=\alpha}$ obtained by fixing the n-th index to α, have the properties of*

   (a) *All-orthogonality: two subtensors $S_{i_n=\alpha}$ and $S_{i_n=\beta}$ are orthogonal for all possible values of n, α and β such that for $\alpha \neq \beta$, $\langle S_{i_n=\alpha}, S_{i_n=\beta} \rangle = 0$.*

   (b) *Ordering: $\|S_{i_n=1}\|_F \geq \|S_{i_n=2}\|_F \geq \cdots \|S_{i_n=I_N}\|_F \geq 0$.*

Based on this theorem, we can rewrite $\mathcal{D}$ as,

$$\mathcal{D} = \{\mathcal{S} \times_1 U_1 \times_2 U_2 \times_3 U_3 : \mathcal{S} \in \mathbb{R}^{r \times r \times r}, U_i \in \mathbb{R}^{d \times r} \text{ s.t. } \langle \mathcal{S}_{i=\alpha}, \mathcal{S}_{i=\beta} \rangle = 0 \text{ for } \alpha \neq \beta \text{ and } U_i^t U_i = I \text{ for } i \in [3]\}.$$

The condition on $\mathcal{S}$ reduces the number of independent parameters in $\mathcal{S}$ from r^3 to r^3-$\frac{3r(r-1)}{2}$. The orthonormal condition U_i reduces the number of independent parameters in U_i from dr to $dr - \frac{r(r-1)}{2} - r$ for all i. Therefore, we get,

$$df_{\text{parameter}} = r^3 - \frac{3r(r-1)}{2} + 3(dr - \frac{r(r+1)}{2}) = r^3 + 3r(d-r).$$

We can calculate the approximated convergency rate by combining this degree of freedom and (15). Following is the result.

1. When $d = 20$ and $r = 3$,

$$\|\Theta_{\text{true}} - \hat{\Theta}\|_F \sim \sqrt{\frac{df_{\text{parameter}}}{n_{\text{sample}}}} = \sqrt{\frac{r^3 + 3r(d-r)}{d^3}} \approx 0.15.$$

2. When $d = 30$ and $r = 3$,

$$\|\Theta_{true} - \hat{\Theta}\|_F \sim \sqrt{\frac{df_{\text{parameter}}}{n_{\text{sample}}}} = \sqrt{\frac{r^3 + 3r(d-r)}{d^3}} \approx 0.1.$$

holds asymptotically, for large d up to constants.

5 Algorithm codes

5.1 Algorithms for tensor svd.

```r
library(rTensor)

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,d1,d2){
  l = r + p
  P1 = matrix(rnorm(d1*l),nrow = d1, ncol = l)
  P2 = matrix(rnorm(d2*l),nrow = d2, ncol = l)
  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_svd2 = function(tnsr,k1,k2,k3,p){
  App = list(Z=NULL,U=NULL)
  rk = c(k1,k2,k3)
  a = c(1,2,3,1,2,3)
  Omega = list()
  Q = list()
  for (i in 1:3) {

    Omega[[i]] <- matrix(rnorm(tnsr@modes[i]*(rk[i]+p)),ncol = rk[i]+p)
  }
  for (i in 1:3) {
    ing <- matrix(rnorm(prod(rk+p)),ncol = rk[i]+p)
    tmp <- k_unfold(ttm(ttm(tnsr,t(Omega[[a[i+1]]]),a[i+1]),t(Omega[[a[i
   +2]]]),a[i+2]),m =i)@data%*%ing
    Q[[i]] <- qr.Q(qr(tmp))
  }
  Coreten <- ttm(ttm(ttm(tnsr,t(Q[[1]]),1),t(Q[[2]]),2),t(Q[[3]]),3)
  App$Z <- Coreten
  App$U <- Q
  return(App)
}



# Algorithm 3.
tensor_svd3 = function(tnsr,k1,k2,k3,p){
  App = list(Z=NULL,U=NULL)
  mode <- tnsr@modes
  mat1 <- k_unfold(tnsr,m=1)
  mat2 <- k_unfold(tnsr,m=2)
  mat3 <- k_unfold(tnsr,m=3)
  Q1 <- StepA(mat1@data,k1,p,mode[2],mode[3])
  Q2 <- StepA(mat2@data,k2,p,mode[3],mode[1])
  Q3 <- StepA(mat3@data,k3,p,mode[1],mode[2])
  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 4.
tensor_svd4 = function(tnsr,k1,k2,k3,p,nrep=5){
  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_rep=Q2_rep=Q3_rep=NULL
  mode <- tnsr@modes

  for(s in 1:nrep){
    Q1 <- StepA(mat1@data,k1,p,mode[2],mode[3])
    Q2 <- StepA(mat2@data,k2,p,mode[3],mode[1])
    Q3 <- StepA(mat3@data,k3,p,mode[1],mode[2])

    Q1_rep=cbind(Q1_rep,Q1)
    Q2_rep=cbind(Q2_rep,Q2)
    Q3_rep=cbind(Q3_rep,Q3)
  }

  Q1=as.matrix(svd(Q1_rep)$u[,1:k1])
  Q2=as.matrix(svd(Q2_rep)$u[,1:k2])
  Q3=as.matrix(svd(Q3_rep)$u[,1:k3])

  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 = 200, 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
  result[i+150,1] = 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)
  est4 = tensor_svd4(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+150,2]= tensor_resid(B,est4)
  result[i,3] = "1st"
  result[i+50,3] = "2nd"
  result[i+100,3]= '3rd'
  result[i+150,3]= "4th"

}


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))
```

```r
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 = 200, ncol =6))
names(result) <- c("sd","angle1","angle2","angle3","method","MSE")


for (i in 1:50) {
  s=sd[i]
  result[i,1] = s
  result[i+50,1] = s
  result[i+100,1] = s
  result[i+150,1]= s
  for (j in 1:100) {
    set.seed(j)
    e = as.tensor(array(rnorm(1000000,mean =0,sd = s),dim = c(100,100,100)
   ))
    D = X+e
    est1 = tensor_svd(D,1,1,1,0)
    est2 = tensor_svd2(D,1,1,1,0)
    est3 = tensor_svd3(D,1,1,1,0)
    est4 = tensor_svd4(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$U[[3]],c)
    result[i,6] <- result[i,6]+tensor_resid(X,est1)
    result[i+50,2] <- result[i+50,2]+angle(est2$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+50,6] <- result[i+50,6]+tensor_resid(X,est2)
    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$U[[3]],c)
    result[i+100,6] <- result[i+100,6]+tensor_resid(X,est3)
    result[i+150,2] <- result[i+150,2]+angle(est4$U[[1]],a)
    result[i+150,3] <- result[i+150,3]+angle(est4$U[[2]],b)
    result[i+150,4] <- result[i+150,4]+angle(est4$U[[3]],c)
    result[i+150,6] <- result[i+150,6]+tensor_resid(X,est4)

  }
  result[i,5] = "1st"
  result[i+50,5] = "2nd"
  result[i+100,5]= '3rd'
  result[i+150,5] = "4th"

}
result[,2:4] <- result[,2:4]/100
result[,6] <- result[,6]/100

write.table(result, file = "4comparison.csv")


g1 <- ggplot(data = result, aes(x=sd,y = MSE ,color = method))+
  geom_point(aes(x=sd, y = MSE))+geom_line(aes(x=sd, y = MSE))
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(g1,g2,g3,g4)
```

6 Algorithms for ordinal tensor analysis(complete version)

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

```

# Some functions needed for Algorithm 4 and 5.
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)
```

```r
}


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 4.
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)
      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 5.
fit_ordinal2 = function(ttnsr,C,A_1,A_2,A_3,omega=TRUE,alph = TRUE){
  omega <- sort(rnorm(2))
```

```r
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)
```

```r
      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)
      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)
}
```

References

[1] L De Lathauwer, B De Moor, J Vandewalle *A multilinear singular value decomposition* SIAM J. Matrix Anal. Appl., 21(4), 1253–1278.