Accuracy proof for algorithm and Ordinal tensor model simulation

Chanwoo Lee

1 Accuracy proof for algorithm

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

```
procedure SVD(A)
    Step A: Approximate SVD of A
    for n ← 1 : N do
        Unfold A as A_(n)
        Generate an d_1 ··· d_{n-1} d_{n+1} ··· d_N × (r_i) Gaussian test matrix Ω^(n) from N(0, 1)
        For Y^(n) = A_(n) Ω^(n)
        Construct a matrix M̂^(n) whose columns form an orthonormal basis for the range of Y^(n)
    get Ĉ = A ×_1 M̂^(1) ×_2 M̂^(2) ··· ×_N M̂^(N)
    Step B: Get approximated SVD
    return (Ĉ, M̂^(1), ··· , M̂^(N))
```

I will show that under some good conditions, our estimation $\hat{M}^{(i)}$ is converging to our parameter $M^{(i)}$ with regards to principle angle which is defined by as follows

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*

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

Our main theorem is as follows

Theorem 1. *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, $\max_{k \in [r_i]} \|C_{(i)}^k\|_2 >> \sigma \sqrt{\max(d_i, \frac{d_1 d_2 d_3}{d_i}) \frac{d_1 d_2 d_3}{d_i}}$ as $d_1, d_2, d_3 \to \infty$, where $C_{(i)}^k$ is k-th row 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

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 $M^{(i)}$.

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

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,

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

To show $\cos(A_{(1)}\Omega x, (A_{(1)} + E_{(1)})\Omega y) \to 1$, it suffices to show that $\cot(A_{(1)}\Omega x, (A_{(1)} + E_{(1)})\Omega y) \to \infty$. We can get inequality as follows

$$\cot(A_{(1)}\Omega x, (A_{(1)} + E_{(1)})\Omega y) \geq \frac{\|A_{(1)}\Omega x\|_2}{\|E_{(1)}\Omega y\|_2} \geq \frac{\max_{k \in [r_1]} \|C_{(1)}^k\|_2 |Z|}{\|E\|_2 \|\Omega y\|_2} \tag{4}$$

To get numerator part in equation (4),

$$\begin{aligned} \|A_{(1)}\Omega x\|_2 &\overset{(1)}{=} \|M^{(1)} C_{(1)} \mathbf{Z} x\|_2 = \|C_{(1)} \sum_{i=1}^{r_1} Z_i x_i\|_2 \quad \text{by orthonormality of } M^{(1)} \\ &= \|C_{(1)} \begin{pmatrix} \tilde{z}_1, & \tilde{z}_2, & \cdots & , \tilde{z}_{r_2 r_3} \end{pmatrix}^T \|_2 \quad \text{where } \tilde{z}_i = \text{ i-th row of } \sum_{i=1}^{r_1} Z_i x_i \end{aligned} \tag{5}$$

By Lemma 1, all elements of $\mathbf{Z}$ are $i.i.d.N(0,1)$. Therefore, elements of $\tilde{Z} = \begin{pmatrix}\tilde{z}_1, & \tilde{z}_2, & \cdots & , \tilde{z}_{r_2 r_3}\end{pmatrix}^T$ from $i.i.d.N(0,1)$ by the linearity of normal distribution and $\|x\|_2 = 1$. Finally we can have following inequality from (5) with a notation $C^i_{(1)}$ =i-th row of $C_{(1)}$

$$\begin{aligned}\|A_{(1)}\Omega x\|_2 &= \|C_{(1)}\tilde{Z}\|_2 = \|([C^1_{(1)}]^T\tilde{Z}, [C^2_{(1)}]^T\tilde{Z}, \cdots, [C^{r_1}_{(1)}]^T\tilde{Z})^T\|_2 \\ &\geq \max_{k\in[r_1]} \|C^k_{(1)}\|_2 |Z| \quad \text{ where } Z \sim N(0,1)\end{aligned} \tag{6}$$

Finally, we can make numerator part in (4). Also, you note that $\|\Omega y\|_2^2 \sim \chi^2(d_2 d_3)$ because $\|y\|_2 = 1$. Based on above, $\|\Omega y\|_2 \asymp (1 + o(1))\sqrt{d_2 d_3}$. Furthermore, we have that $\|E\| \asymp (2 + o(1))\sigma\sqrt{\max(d_1, d_2 d_3)}$ by Lemma 2. Finally, for any fixed $L > 0$

$$\begin{aligned}P(\cot(A_{(1)}\Omega x, (A_{(1)} + E_{(1)})\Omega y) \geq L) &\geq P(\frac{\max_{k\in[r_1]} \|C^k_{(1)}\|_2 |Z|}{\|E\|_2 \|\Omega y\|_2} \geq L) \\ &\geq P(|Z| \geq \frac{2L\sigma\sqrt{d_2 d_3 \max(d1, d_2 d_3)}}{\max_{k\in[r_1]} \|C^k_{(1)}\|_2}) \\ &\geq 1 - \frac{4\lambda}{\sqrt{2\pi}}\end{aligned} \tag{7}$$

where $\lambda \overset{def}{=} \frac{L\sigma\sqrt{d_2 d_3 \max(d1, d_2 d_3)}}{\max_{k\in[r_1]} \|C^k_{(1)}\|_2}$. The last line of equation (7) used the fact that $P(|(N(0,1)| \geq t) \geq 1 - 2t\phi(0)$ for any $t \geq 0$, where $\phi(\cdot)$ is the pdf for the standard normal. Based on assumption $\lambda \to 0$, we conclude that $\cot(A_{(1)}\Omega x, (A_{(1)} + E_{(1)}) \geq L$ with high probability. Therefore, we can get desired results sending $L \to \infty$ by the equation (3). □

Lemma 1. *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* $i.i.dN(0,1)$ *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* $B = (M^{(3)} \otimes M^{(2)})^T = \begin{pmatrix}M^{(3)}_1 \otimes M^{(2)}_1, & M^{(3)}_1 \otimes M^{(2)}_2, & \cdots, & M^{(3)}_{r_3} \otimes M^{(2)}_{r_2}\end{pmatrix}$

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 $B_i = M^{(3)}_{ind_1(i)} \otimes M^{(2)}_{ind_2(i)}$.
   To give a simple example, $(ind_1(1), ind_2(1)) = (1,1)$ because $B_1 = M^{(3)}_1 \otimes M^{(2)}_1$.
   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 $\square$

Lemma 2 (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*

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

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 Ordinal tensor model simulation

2.1 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.

2.1.1 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

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

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.

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

Therefore, $L_{\mathcal{Y}}(\Theta)$ becomes $L_{\mathcal{Y}}(Vec(A_1))$ which is a simple 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.

$$Vec(\Theta_{(2)}) = \big((A_3 \otimes A_1)\mathcal{C}_{(2)}^T \otimes I_{d_2}\big)Vec(A_2)$$

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

$$Vec(\Theta_{(3)}) = \big((A_2 \otimes A_1)\mathcal{C}_{(3)}^T \otimes I_{d_3}\big)Vec(A_3)$$

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 2 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

2.1.2 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 3 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

2.2 Simulation

For this section, I want to talk about simulation results of Algorithm 2 and Algorithm 3.

2.2.1 Known Bin boundary case

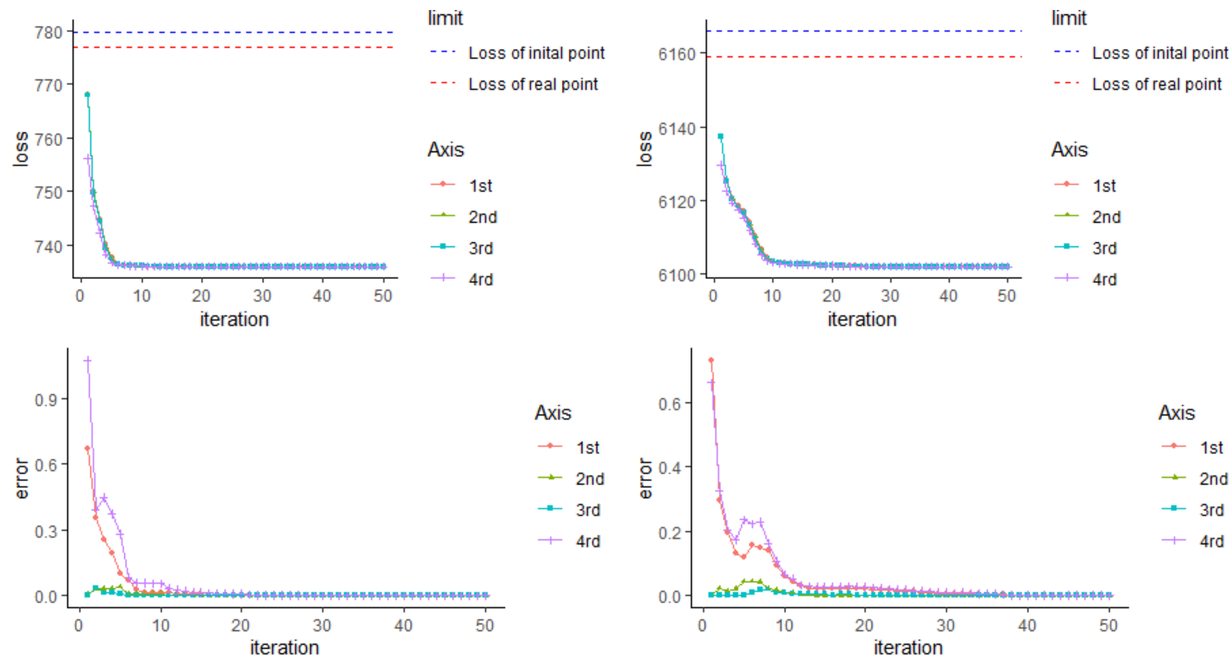
Our given ordinal tensor data $D \in R^{d \times d \times d}$ has three values such that $D_{i,j,k} \in \{1, 2, 3\}$. This data D was made by following procedure.

1. Choose arbitrary core tensor $\mathcal{C} \in R^{3 \times 3 \times 3}$, orthonormal matrix $A_1 \in R^{d \times 3}, A_2 \in R^{d \times 3}$ and $A_3 \in R^{d \times 3}$
2. Get tensor $\Theta = \mathcal{C} \times_1 A_1 \times_2 A_2 \times_3 A_3 \in R^{d \times d \times d}$
3. Fix $\alpha = (\alpha_1, \alpha_2) = (1, 2)$
4. Make ordinal tensor $D \in R^{d \times d \times d}$ such that

   $P(D_{i,j,k} = 1) = logistic(\alpha_1 + \Theta_{i.j.l}), \quad P(D_{i,j,k} = 2) = logistic(\alpha_2 + \Theta_{i.j.l}) - logistic(\alpha_1 + \Theta_{i.j.l}),$
   $P(D_{i,j,k} = 3) = 1 - logistic(\alpha_2 + \Theta_{i.j.l})$

   where $logistic(x) = \dfrac{1}{1 + e^{-x}}$

From this given data D, our goal is to estimate $\mathcal{C}, A_1, A_2$ and A_3 using Algorithm 2 with randomly selected initial values. I repeated above simulation for $d = 10$ and $d = 20$. Results are in Figure 1.



(a) When $d = 10$ (b) When $d = 20$

Figure 1: When we have knowledge that $\alpha = (1, 2)$. Y axis in upper part figures is values of loss function at each iteration. Blue dashed horizontal line is loss value evaluated at initial point and red dashed horizontal line is loss value evaluated at real parameter point. Y axis in lower part figures is value of norm difference between previous parameter and updated parameter. You can check our algorithm converges in both $d = 10$ and $d = 20$ cases and gives us parameters having smaller loss value than real parameters.

Also I could check difference norm between real Θ and estimated $\hat{\Theta}$ such that

$$diff(\Theta, \hat{\Theta}) = \|\Theta - \hat{\Theta}\|_F / \sqrt{d^3}$$

When $d = 10$, $diff(\Theta, \hat{\Theta}) = 0.7000825$. When $d = 20$, $diff(\Theta, \hat{\Theta}) = 0.2789708$. I think our estimaters are quite close to real Θ

2.2.2 Unknown Bin boundary case

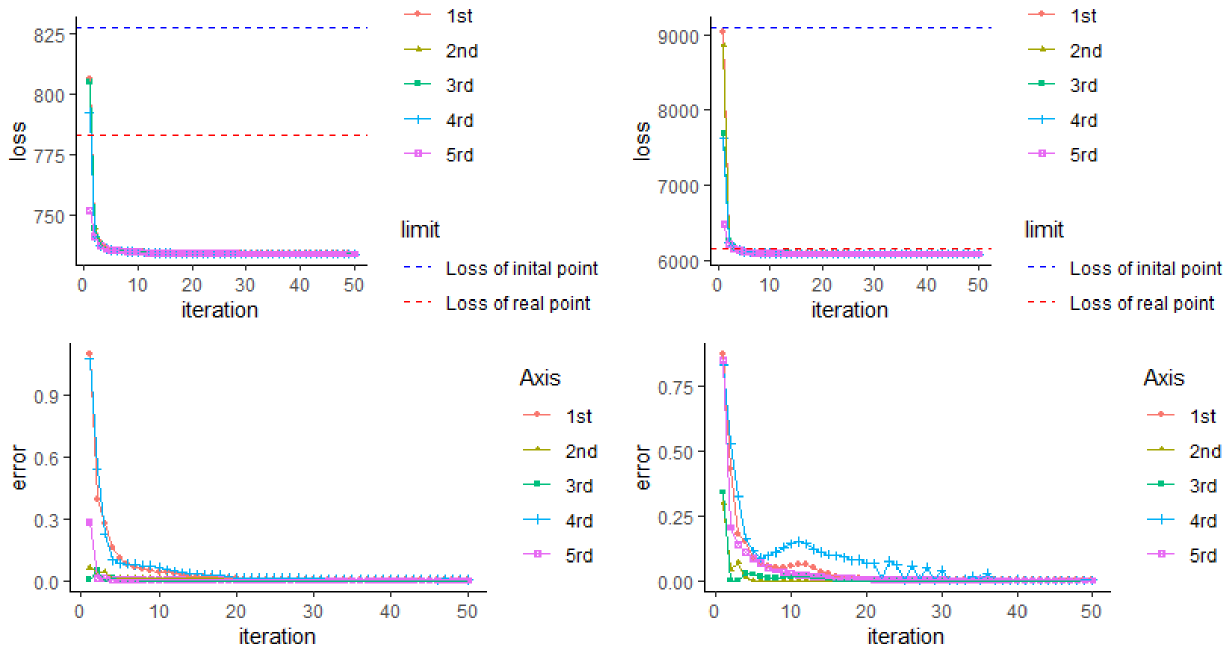
Our given ordinal tensor data $D \in R^{d\times d\times d}$ has three values such that $D_{i,j,k} \in \{1, 2, 3\}$. This data D was made by following procedure.

1. Choose arbitrary core tensor $\mathcal{C} \in R^{3\times3\times3}$, orthonormal matrix $A_1 \in R^{d\times3}$, $A_2 \in R^{d\times3}$ and $A_3 \in R^{d\times3}$
2. Get tensor $\Theta = \mathcal{C} \times_1 A_1 \times_2 A_2 \times_3 A_3 \in R^{d\times d\times d}$
3. Choose $\alpha = (1, 2) = (\alpha_1, \alpha_2)$

4. Make ordinal tensor $D \in R^{d \times d \times d}$ such that

$$P(D_{i,j,k} = 1) = logistic(\alpha_1 + \Theta_{i.j.l}), \quad P(D_{i,j,k} = 2) = logistic(\alpha_2 + \Theta_{i.j.l}) - logistic(\alpha_1 + \Theta_{i.j.l}),$$
$$P(D_{i,j,k} = 3) = 1 - logistic(\alpha_2 + \Theta_{i.j.l})$$
$$\text{where } logistic(x) = \frac{1}{1 + e^{-x}}$$

Only difference between the first and the second simulation is we don't know $\alpha = (1, 2)$ in the second and will estimate α including other parameters. From this given data D, we estimated $\alpha, \mathcal{C}, A_1, A_2$ and A_3 using Algorithm 3. I repeated above simulation for $d = 10$ and $d = 20$. Results are in Figure 2.



(a) When $d = 10$ (b) When $d = 20$

Figure 2: When we don't have knowledge that $\alpha = (1, 2)$Y axis in upper part figures is values of loss function at each iteration. Blue dashed horizontal line is loss value evaluated at initial point and red dashed horizontal line is loss value evaluated at real parameter point. Y axis in lower part figures is value of norm difference between previous parameter and updated parameter. You can check our algorithm converges in both $d = 10$ and $d = 20$ cases and gives us parameters having smaller loss value than real parameters.

Let's check how close it is between real $\alpha = (1, 2)$ and estimated $\hat{\alpha}$. When $d = 10$, our estimated $\hat{\alpha} = (0.9993238, 2.0961265)$ and when $d = 20$, our estimated $\hat{\alpha} = (0.9484518, 1.9872800)$ For difference norm defined above, we got $diff(\Theta, \hat{\Theta}) = 0.8158596$ and $diff(\Theta, \hat{\Theta}) = 0.3243989$ for each simulation. I think those simulation gave us moderate results.

3 Question and to do list

1. Actually, I can't fully understand why graphs of Loss vs iteration look different according to knowledge of α: Loss value evaluated at real point have a tendency to be small when we don't know α.

2. It took about 20 minutes for my algorithm to converge in the case of $d = 20$ and when we don't know the value of α. Is it natural phenomenon or my algorithm is not efficient?(there's no problem when $d = 10$)

3. I am now studying on estimation properties based on binary tensor research paper.

4 R codes

4.1 Algorithm 2

```

library(MASS)
library(rTensor)
library(pracma)

normf = function(tnsr){
  return(rTensor::fnorm(tnsr))
}

logistic = function(x){
  return(1/(1+exp(-x)))
}

alpha = c(1,2)

likelihood = function(ttnsr,thet,alpha){
  p1 = logistic(c(thet@data) + alpha[1])
  p2 = logistic(c(thet@data) + alpha[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]))))
}

mleordinal = function(ttnsr,C,A_1,A_2,A_3){
  result = list()
  alpha = c(1,2)
  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)
  while (error > 10^-3 ) {
    iter = iter +1
    #update A_1
    tmp1 <- A_1
    W1 =  kronecker(kronecker(A_3,A_2)%*%t(k_unfold(C,1)@data),diag(1,d1))
```

```
  h1 = function(A_1){
    thet =W1%*%c(A_1)
    p1 = logistic(thet + alpha[1])
    p2 = logistic(thet + alpha[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){
    thet =W1%*%c(A_1)
    p1 = logistic(thet + alpha[1])
    p2 = logistic(thet + alpha[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)
  }

  a <- optim(c(A_1),h1,g1,method = "BFGS")
  A_1 <- matrix(a$par,nrow = d1,ncol = r1)
  error1 <- norm(A_1-tmp1,"F")/sqrt(length(c(A_1)))


  # update A_2
  tmp2 <- A_2
  W2 <- kronecker(kronecker(A_3,A_1)%*%t(k_unfold(C,2)@data),diag(1,d2))
  h2 = function(A_2){
    tmpthet = W2%*%c(A_2)
    thet = c(k_unfold(k_fold(matrix(tmpthet,nrow = d2),2,modes = c(d1,d2
,d3)),1)@data)
    p1 = logistic(thet + alpha[1])
    p2 = logistic(thet + alpha[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]))))
  }
  g2 = function(A_2){
    tmpthet = W2%*%c(A_2)
    thet = c(k_unfold(k_fold(matrix(tmpthet,nrow = d2),2,modes = c(d1,d2
,d3)),1)@data)
    p1 = logistic(thet + alpha[1])
    p2 = logistic(thet + alpha[2])
    q1 <- p1-1
    q2 <- (p2*(1-p2)-p1*(1-p1))/(p1-p2)
    q3 <- p2
    gd = apply(diag(q1[which(c(ttnsr)==1)])%*%W2[which(c(ttnsr)==1),],2,
sum)+
```

```
      apply(diag(q2[which(c(ttnsr)==2)])%*%W2[which(c(ttnsr)==2),],2,sum)+
      apply(diag(q3[which(c(ttnsr)==3)])%*%W2[which(c(ttnsr)==3),],2,sum)
    return(gd)
  }
  b <- optim(c(A_2),h2,g2,method = "BFGS")
  A_2 <- matrix(b$par,nrow = d2,ncol = r2)
  error2 <- norm(A_2-tmp2,"F")/sqrt(length(c(A_2)))

  # update A_3
  tmp3 <- A_3
  W3 <- kronecker(kronecker(A_2,A_1)%*%t(k_unfold(C,3)@data),diag(1,d3))
  h3 = function(A_3){
    tmpthet = W3%*%c(A_3)
    thet = c(k_unfold(k_fold(matrix(tmpthet,nrow = d3),3,modes = c(d1,d2,d3)),1)@data)
    p1 = logistic(thet + alpha[1])
    p2 = logistic(thet + alpha[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]))))
  }
  g3 = function(A_2){
    tmpthet = W3%*%c(A_3)
    thet = c(k_unfold(k_fold(matrix(tmpthet,nrow = d3),3,modes = c(d1,d2,d3)),1)@data)
    p1 = logistic(thet + alpha[1])
    p2 = logistic(thet + alpha[2])
    q1 <- p1-1
    q2 <- (p2*(1-p2)-p1*(1-p1))/(p1-p2)
    q3 <- p2
    gd = apply(diag(q1[which(c(ttnsr)==1)])%*%W3[which(c(ttnsr)==1),],2,sum)+
      apply(diag(q2[which(c(ttnsr)==2)])%*%W3[which(c(ttnsr)==2),],2,sum)+
      apply(diag(q3[which(c(ttnsr)==3)])%*%W3[which(c(ttnsr)==3),],2,sum)
    return(gd)
  }
  c <- optim(c(A_3),h3,g3,method = "BFGS")
  A_3 <- matrix(c$par,nrow = d3,ncol = r3)
  error3 <- norm(A_3-tmp3,"F")/sqrt(length(c(A_3)))
  # update C
  tmp4 <- C
  W4 <- kronecker(kronecker(A_3,A_2),A_1)
  h4 = function(Cvec){
    thet = W4%*%Cvec
    p1 = logistic(thet + alpha[1])
    p2 = logistic(thet + alpha[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]))))
  }
```

```
    g4 = function(Cvec){
      thet = W4%*%Cvec
      p1 = logistic(thet + alpha[1])
      p2 = logistic(thet + alpha[2])
      q1 <- p1-1
      q2 <- (p2*(1-p2)-p1*(1-p1))/(p1-p2)
      q3 <- p2
      gd = apply(diag(q1[which(c(ttnsr)==1)])%*%W4[which(c(ttnsr)==1),],2,sum)+
        apply(diag(q2[which(c(ttnsr)==2)])%*%W4[which(c(ttnsr)==2),],2,sum)+
        apply(diag(q3[which(c(ttnsr)==3)])%*%W4[which(c(ttnsr)==3),],2,sum)
      return(gd)
    }
    d <- optim(c(C@data),h4,g4,method = "BFGS")
    C <- new("Tensor",C@num_modes,C@modes,data =d$par)
    error4 <- normf(C-tmp4)/sqrt(length(c(C@data)))
    error <- max(error1,error2,error3,error4)
  }
  result$C <- C; result$A_1 <- A_1; result$A_2 <- A_2; result$A_3 <- A_3; result$iteration <- iter
  return(result)
}
```

4.2 Algorithm 3

```
mleordinal2 = function(ttnsr,C,A_1,A_2,A_3,alpha){
  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)
  while (error > 10^-3 ) {
    iter = iter +1
    #update A_1
    tmp1 <- A_1
    W1 =  kronecker(kronecker(A_3,A_2)%*%t(k_unfold(C,1)@data),diag(1,d1))
    h1 = function(A_1){
      thet =W1%*%c(A_1)
      p1 = logistic(thet + alpha[1])
      p2 = logistic(thet + alpha[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){
      thet =W1%*%c(A_1)
      p1 = logistic(thet + alpha[1])
      p2 = logistic(thet + alpha[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)
  }

  a <- optim(c(A_1),h1,g1,method = "BFGS")
  A_1 <- matrix(a$par,nrow = d1,ncol = r1)
  error1 <- norm(A_1-tmp1,"F")/sqrt(length(c(A_1)))


  # update A_2
  tmp2 <- A_2
  W2 <- kronecker(kronecker(A_3,A_1)%*%t(k_unfold(C,2)@data),diag(1,d2))
  h2 = function(A_2){
    tmpthet = W2%*%c(A_2)
    thet = c(k_unfold(k_fold(matrix(tmpthet,nrow = d2),2,modes = c(d1,d2,d3)),1)@data)
    p1 = logistic(thet + alpha[1])
    p2 = logistic(thet + alpha[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]))))
  }
  g2 = function(A_2){
    tmpthet = W2%*%c(A_2)
    thet = c(k_unfold(k_fold(matrix(tmpthet,nrow = d2),2,modes = c(d1,d2,d3)),1)@data)
    p1 = logistic(thet + alpha[1])
    p2 = logistic(thet + alpha[2])
    q1 <- p1-1
    q2 <- (p2*(1-p2)-p1*(1-p1))/(p1-p2)
    q3 <- p2
    gd = apply(diag(q1[which(c(ttnsr)==1)])%*%W2[which(c(ttnsr)==1),],2,sum)+
      apply(diag(q2[which(c(ttnsr)==2)])%*%W2[which(c(ttnsr)==2),],2,sum)+
      apply(diag(q3[which(c(ttnsr)==3)])%*%W2[which(c(ttnsr)==3),],2,sum)
    return(gd)
  }
  b <- optim(c(A_2),h2,g2,method = "BFGS")
  A_2 <- matrix(b$par,nrow = d2,ncol = r2)
  error2 <- norm(A_2-tmp2,"F")/sqrt(length(c(A_2)))

  # update A_3
  tmp3 <- A_3
  W3 <- kronecker(kronecker(A_2,A_1)%*%t(k_unfold(C,3)@data),diag(1,d3))
  h3 = function(A_3){
    tmpthet = W3%*%c(A_3)
```

```
    thet = c(k_unfold(k_fold(matrix(tmpthet,nrow = d3),3,modes = c(d1,d2,d3)),1)@data)
    p1 = logistic(thet + alpha[1])
    p2 = logistic(thet + alpha[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]))))
  }
  g3 = function(A_2){
    tmpthet = W3%*%c(A_3)
    thet = c(k_unfold(k_fold(matrix(tmpthet,nrow = d3),3,modes = c(d1,d2,d3)),1)@data)
    p1 = logistic(thet + alpha[1])
    p2 = logistic(thet + alpha[2])
    q1 <- p1-1
    q2 <- (p2*(1-p2)-p1*(1-p1))/(p1-p2)
    q3 <- p2
    gd = apply(diag(q1[which(c(ttnsr)==1)])%*%W3[which(c(ttnsr)==1),],2,sum)+
      apply(diag(q2[which(c(ttnsr)==2)])%*%W3[which(c(ttnsr)==2),],2,sum)+
      apply(diag(q3[which(c(ttnsr)==3)])%*%W3[which(c(ttnsr)==3),],2,sum)
    return(gd)
  }
  c <- optim(c(A_3),h3,g3,method = "BFGS")
  A_3 <- matrix(c$par,nrow = d3,ncol = r3)
  error3 <- norm(A_3-tmp3,"F")/sqrt(length(c(A_3)))
  # update C
  tmp4 <- C
  W4 <- kronecker(kronecker(A_3,A_2),A_1)
  h4 = function(Cvec){
    thet = W4%*%Cvec
    p1 = logistic(thet + alpha[1])
    p2 = logistic(thet + alpha[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]))))
  }
  g4 = function(Cvec){
    thet = W4%*%Cvec
    p1 = logistic(thet + alpha[1])
    p2 = logistic(thet + alpha[2])
    q1 <- p1-1
    q2 <- (p2*(1-p2)-p1*(1-p1))/(p1-p2)
    q3 <- p2
    gd = apply(diag(q1[which(c(ttnsr)==1)])%*%W4[which(c(ttnsr)==1),],2,sum)+
      apply(diag(q2[which(c(ttnsr)==2)])%*%W4[which(c(ttnsr)==2),],2,sum)+
      apply(diag(q3[which(c(ttnsr)==3)])%*%W4[which(c(ttnsr)==3),],2,sum)
    return(gd)
  }
```

```
    d <- optim(c(C@data),h4,g4,method = "BFGS")
    C <- new("Tensor",C@num_modes,C@modes,data =d$par)
    error4 <- normf(C-tmp4)/sqrt(length(c(C@data)))

    #update alpha
    tmp5 <- alpha
    theta <- ttm(ttm(ttm(C,A_1,1),A_2,2),A_3,3)
    m <- polr(as.factor(c(ttnsr))~offset(-c(theta@data)))
    alpha <- m$zeta
    error5 <- sqrt(sum((alpha-tmp5)^2)/2)
    error <- max(error1,error2,error3,error4,error5)
  }
  result$C <- C; result$A_1 <- A_1; result$A_2 <- A_2; result$A_3 <- A_3
  result$alpha <- alpha ; result$iteration <- iter
  return(result)
}
```