New version of the theorem for the convergence and The new algorithm for ordinal tensors

Chanwoo Lee

Nov 19, 2019

1 Estimation accuracy for more general randomized SVD

We generalize from random normal test matrices to arbitrary test matrices. We can guarantee the convergence of estimators under the certain conditions in the next theorem.

Theorem 1. *Let $\mathcal{A} = \mathcal{C} \times_1 M^{(1)} \times_2 \cdots \times_N M^{(N)}$ be a signal tensor, where $\mathcal{C} \in \mathbb{R}^{r_1\times\cdots\times r_N}$ is a core tensor and $M^{(i)}$ is an orthonormal matrix in $\mathbb{R}^{d_i\times r_i}$ for all $i \in [N]$. 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, $(\hat{\mathcal{C}}, \hat{M}^{(1)}, \cdots, \hat{M}^{(N)})$ is obtained from the randomized algorithms with test matrices $\Omega^{(i)}$ for $i \in [N]$. If $s_{min}(C_{(i)})\sqrt{\mathbb{1}_{r_i}^T(\Omega^{(i)})^T P^{(i)}(P^{(i)})^T\Omega^{(i)}\mathbb{1}_{r_i}} >> \sigma\sqrt{r_i \max(d_i, \frac{\prod_{j\neq i} d_j}{d_i})}\|\Omega^{(i)}\|_\sigma$ as $d_1, \cdots, d_N \to \infty$, where $s_{min}(C_{(i)})$ is the smallest singular value of $C_{(i)}$, $P^{(i)} = \left[(M^{(N)} \otimes \cdots \otimes M^{(i+1)} \otimes M^{(i-1)} \otimes \cdots \otimes M^{(1)})V_{r_i}^{(i)}\right]$ and $V_{r_i}^{(i)}$ is the matrix of the r largest left singular vectors of $C_{(i)}$. Then, the following holds true.*

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

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

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

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

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

Define $B = (M^{(N)} \otimes \cdots \otimes M^{(2)})$. the randomized algorithms generates a test matrix $\Omega^{(1)}$ and captures the image space of unfolded matrix $A_{(1)}$. Having this procedure in mind, we obtain,

$$\begin{aligned} A_{(1)}\Omega^{(1)} &= M^{(1)}C_{(1)}(M^{(N)} \otimes \cdots \otimes M^{(2)})^T\Omega^{(1)} \\ &= M^{(1)}C_{(1)}B^T\Omega^{(1)}. \end{aligned}$$

However, since the input is $\mathcal{D} = \mathcal{A} + \mathcal{E}$, we have the image space of $A_{(1)} + E_{(1)}$ instead of $A_{(1)}$. Therefore, the estimator $\hat{M}^{(1)}$ is obtained from the following equality.

$$\begin{aligned} (A_{(1)} + E_{(1)})\Omega^{(1)} &= M^{(1)}C_{(1)}B^T\Omega^{(1)} + E_{(1)}\Omega^{(1)} \\ &= \hat{M}^{(1)}R \quad \text{(QR decomposition)}. \end{aligned}$$

From the relationship that $\text{span}(A_{(1)}\Omega^{(1)}) \subset \text{span}(M^{(1)})$ and $\text{span}(A_{(1)}\Omega^{(1)} + E_{(1)}\Omega^{(1)}) = \text{span}(\hat{M^{(1)}})$, we have the following.

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

The first argument in the theorem holds true by (1) if

$$\max_{x \in R^{r_1}, y \in R^{r_1}, \|x\|_2 = \|y\|_2 = 1} \cos(A_{(1)}\Omega^{(1)}x, (A_{(1)} + E_{(1)})\Omega^{(1)}y) \to 1. \tag{2}$$

Also (2) holds true, if

$$\cot(A_{(1)}\Omega^{(1)}x, (A_{(1)} + E_{(1)})\Omega^{(1)}y) \to \infty \text{ for some } x, y \text{ such that } \|x\| = \|y\| = 1. \tag{3}$$

So the main proof of this theorem is to show (3). We prove (3) by the following inequality.

$$\cot(A_{(1)}\Omega^{(1)}x, (A_{(1)} + E_{(1)})\Omega^{(1)}y) \geq \frac{\|A_{(1)}\Omega^{(1)}x\|_2}{\|E_{(1)}\Omega^{(1)}y\|_2} \geq \frac{s_{min}(C_{(1)})\sqrt{\mathbb{1}_{r_1}^T(\Omega^{(1)})^T P^{(1)}(P^{(1)})^T\Omega^{(1)}\mathbb{1}_{r_1}}}{\sqrt{r_1}\|E\|_F\|\Omega^{(1)}\|_\sigma}. \tag{4}$$

for some x and y.

To get the numerator part in (4),

$$
\begin{aligned}
\|A_{(1)}\Omega^{(1)}x\|_2 &= \|M^{(1)}C_{(1)}B^T\Omega^{(1)}x\|_2 \\
&\overset{(i)}{=} \|C_{(1)}B^T\Omega^{(1)}x\|_2 \\
&\overset{(ii)}{=} \|U^{(1)}\Sigma^{(1)}(V^{(1)})^T B^T\Omega^{(1)}x\|_2 \\
&= \|\Sigma^{(1)}(V^{(1)}_{r_1})^T B^T\Omega^{(1)}x\|_2 \\
&\geq s_{\min}(C_{(1)})\|(V^{(1)}_{r_1})^T B^T\Omega^{(1)}x\|_2 \\
&= s_{\min}(C_{(1)})\|(P^{(1)})^T\Omega^{(1)}x\|_2 \\
&\overset{(iii)}{=} s_{\min}(C_{(1)})\frac{1}{\sqrt{r_1}}\|(P^{(1)})^T\Omega^{(1)}\mathbb{1}_{r_1}\|_2 \\
&= \frac{s_{\min}(C_{(1)})\sqrt{\mathbb{1}_{r_1}^T(\Omega^{(1)})^T P^{(1)}(P^{(1)})^T\Omega^{(1)}\mathbb{1}_{r_1}}}{\sqrt{r_1}}.
\end{aligned}
$$

(i) is from the orthonormality of $M^{(1)}$. Singular value decomposition of $C_{(1)}$ is used in (ii). Notice, In (iii), we put $x = \mathbb{1}_{r_1}/\sqrt{r_1}$. Therefore, we get the numerator part. For the denominator of (4), we knows

$$\|E\|_F \asymp (2 + o(1))\sigma\sqrt{\max(d_1, d_2 \cdots d_N)}.$$

Also, notice that

$$\|\Omega y\|_2 \leq \|\Omega\|_\sigma.$$

Therefore, we get (4). The last argument that $\|\mathcal{A} - \hat{\mathcal{A}}\| \to 0$ is derived directly from Theorem 2 and Theorem 3 in the 7th meeting note. □

When all entries of the test matrices are i.i.d. from $N(0,1)$, we have the following corollary.

Corollary 1. *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. Suppose we use standard normal random matrices as test matrices in Theorem 1. If $s_{min}(C_{(i)}) >> \sigma\sqrt{\max(d_i, \frac{d_1 d_2 d_3}{d_i})\frac{d_1 d_2 d_3}{d_i r_i}}$ as $d_1, d_2, d_3 \to \infty$, where $s_{min}(C_{(i)})$ is the smallest singular value of $C_{(i)}$. Then, the following holds true.*

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

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

Proof. It is enough to show that the condition in Theorem 1 implies the condition in the Corollary 1. We will show this argument when $i=1$ with fixed $\Omega^{(1)}$ replaced by a standard normal random matrix. First, let us check

$$\sqrt{\mathbb{1}_{r_1}^T(\Omega^{(1)})^T P^{(1)}(P^{(1)})^T\Omega^{(1)}\mathbb{1}_{r_1}}. \tag{5}$$

where $\Omega^{(1)}$ is a standard normal random matrix. Notice,

$$(P^{(1)})^T\Omega^{(1)}\mathbb{1}_{r_1} \sim (P^{(1)})^T N_{r_1}(0, r_1 I_{r_1}) \overset{\mathrm{d}}{=} N_{r_1}(0, r_1(P^{(1)})^T P^{(1)}) = N_{r_1}(0, r_1 I_{r_1}).$$

Because $(P^{(1)})^T P^{(1)} = I_{r_1}$ by the definition of $P^{(1)}$. We get the following distribution for (5).

$$\sqrt{\mathbb{1}_{r_1}^T(\Omega^{(1)})^T P^{(1)}(P^{(1)})^T\Omega^{(1)}\mathbb{1}_{r_1}} \overset{\mathrm{d}}{=} \sqrt{N_{r_1}(0, r_1 I_{r_1})^T N_{r_1}(0, r_1 I_{r_1})} \overset{\mathrm{d}}{=} \sqrt{r_1\chi^2_{r_1}}. \tag{6}$$

Secondly, we have

$$\begin{aligned}
\|\Omega^{(1)}\|_\sigma &\geq \|\Omega^{(1)}y\|_2 \quad \text{ where } y\in\mathbb{R}^{r_1} \text{ such that } \|y\|_2 = 1\\
&\overset{\mathrm{d}}{=} \sqrt{N_{d_2d_3}(0, I_{d_2d_3})^T N_{d_2d_3}(0, I_{d_2d_3})}\\
&\overset{\mathrm{d}}{=} \sqrt{\chi^2_{d_2d_3}}\\
&\asymp (1+o(1))\sqrt{d_2d_3}.
\end{aligned}$$

Therefore, the condition in Theorem 1 can be rewritten as,

$$s_{\min}(C_{(1)})\sqrt{\chi^2_{r_1}} >> \sigma\sqrt{\max(d_1, d_2d_3)d_2d_3}. \tag{7}$$

By the following inequality, we have the condition of this corollary from (7). (7) implies the left side of the following equation converges to 1.

$$\begin{aligned}
P\big(s_{\min}(C_{(1)})\sqrt{\chi^2_{r_1}} > \sigma\sqrt{\max(d_1, d_2d_3)d_2d_3}\big) &= P\big(\chi^2_{r_1} \geq \frac{\sigma^2 d_2d_3\max(d_1, d_2d_3)}{s_{min}(C_{(1)})^2}\big)\\
&\overset{(i)}{\leq} 1-\left(\lambda e^{1-\lambda}\right)^{\frac{r_1}{2}}.
\end{aligned}$$

$\square$

In (i), we defined $\lambda \overset{def}{=} \frac{\sigma^2 d_2 d_3 \max(d1, d_2 d_3)}{r_1 s_{min}(C_{(1)})^2}$ and used Chernoff bounds,

$$P(\chi_r^2 \geq t) \leq 1 - \big(\frac{t}{r} e^{1-\frac{t}{r}}\big)^{\frac{r}{2}} \text{ for any } t \geq 0.$$

Therefore, λ should converge to 0 when (7) holds. Now we have the condition of the corollary true.

2 Extended angle simulation for an arbitrary rank

This simulation investigates the accuracy of estimators in terms of angles and MSE for an arbitrary rank. We consider an order-3 dimension $(20, 20, 20)$ signal tensor X. We assume X has Tucker decomposition as $X = \mathcal{C} \otimes_1 B_1 \otimes_2 B_2 \otimes_3 B_3$, where $B_i \in \mathbb{R}^{20\times 3}$ for all i. and $\mathcal{C} \in \mathbb{R}^{3\times 3\times 3}$ a core tensor. All entries of $\mathcal{C}$ are i.i.d. drawn from $\mathrm{N}(0, 1)$. B_1, B_2, B_3 are randomly drawn from orthonormal matrices. We vary the noise level $\sigma \in \{0.01, 0.02, \cdots 0.49, 0.5\}$. We use target rank 3 and estimate the signal matrices according to each algorithms. We compare the principal angles between the true signal matrices and estimators. Figure 1 shows that Method 3 outperforms the other methods in all respects.

3 Improved ordinal tensors algorithm

First, I constructed stochastic gradient descent(SGD) algorithm for updating the core tensor. However, there were some problems to implement SGD method. First, I used various batch sizes from $B = 100$ to $B = 1000$. But this algorithm had more than 100 the number of iterations in all batch sizes. Secondly, I picked the tolerance size as 10^-4. this algorithm reached to this tolerance size fast but had the smaller likelihood value that the value with the true parameter. This means it did not get to the optimal point but had small improvement on each update. In addition, I found that it also has variation issues, which is inevitable. There were some cases that this algorithm worked quite well with moderate iteration numbers and time. However, it sometimes performed poorly with large iteration numbers and bad outputs. Instead of using stochastic gradient method, I constructed the algorithm which calculate hessian in each update for the core tensor. This hessian function reduced iteration time dramatically for updating the core tensor. Also, it converged with the less the number

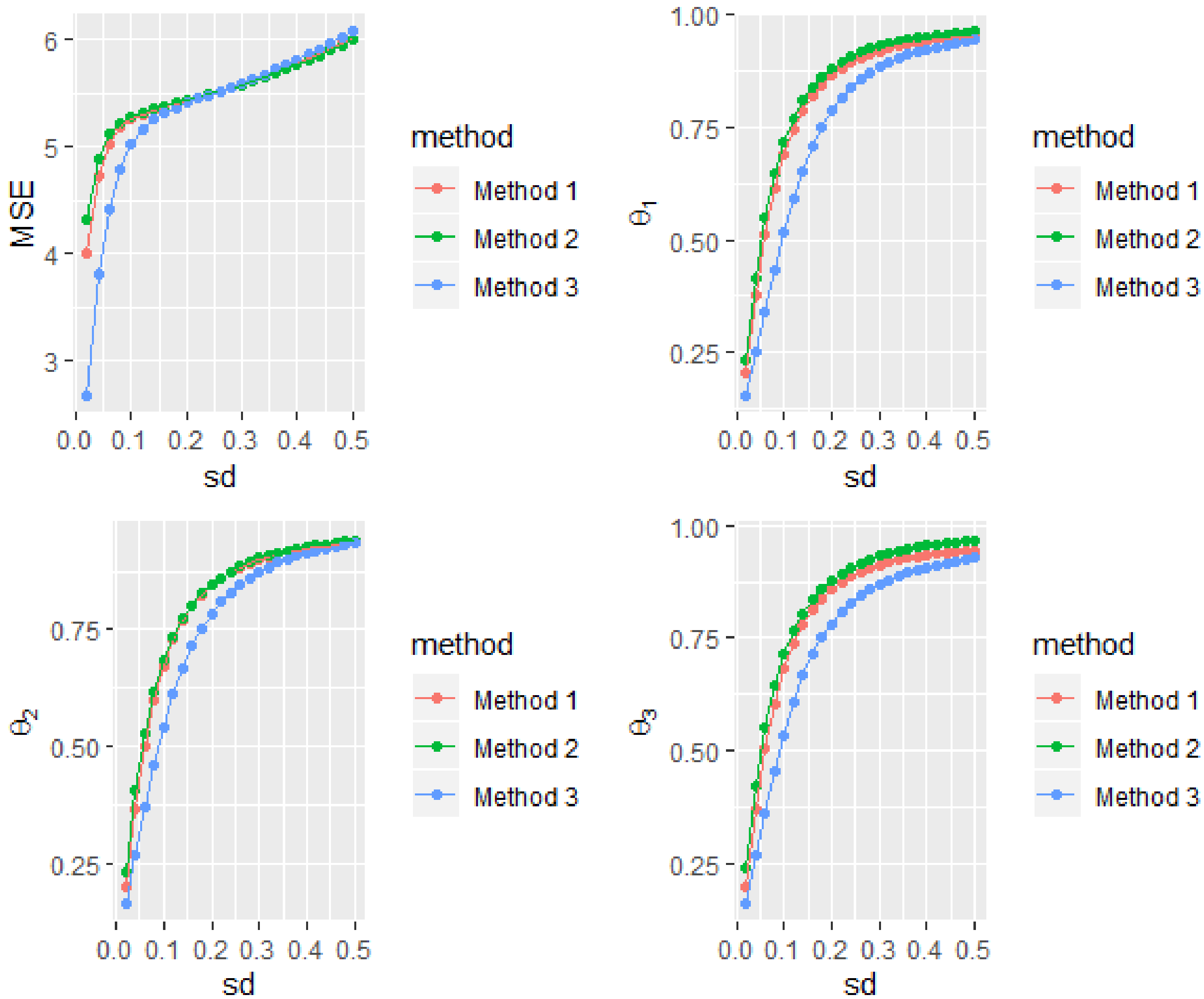

Figure 1: Each y axis means the principal angles for each mode. We do not have consistent simulation results: Method 3 has the best performance in MSE and the principal angles.

of iterations than the previous algorithm. All output had the greater likelihood value than the value with the true parameters.

3.1 Simulation for ordinal tensors

I performed the simulations having the same setting as in section 3.2 in the 7-th note. I summarized the output as follows.

1. When $d = 20$ and $r = 3$ with $\max(\Theta_{\text{True}}) = 5.78633$ and $\omega = (-0.2, 0.2)$.
   We have $L(\Theta_{\text{True}}) = 6459.568$ and $L(\Theta_0) = 7414.672$.

	(With ω information)	(Without ω inforamtion)
$L(\hat{\Theta})$	6373.461	6373.191
Computation time	51 sec	52 sec

   When we implement algorithm without ω information, we have an estimate $\hat{\omega} = (-1.8011, 0.2513)$

2. When $d = 30$ and $r = 3$ with $\max(\Theta_{\text{True}}) = 6.8348$ and $\omega = (-0.2, 0.2)$.
   We have $L(\Theta_{\text{True}}) = 21895.92$ and $L(\Theta_0) = 24917.37$.

	(With ω information)	(Without ω inforamtion)
$L(\hat{\Theta})$	21761.86	21761.92
Computation time	432 sec	385.56 sec

   When we implement algorithm without ω information, we have an estimate $\hat{\omega} = (-0.221403, 0.1868)$

The following is the scatter plot between true parameters and estimators.

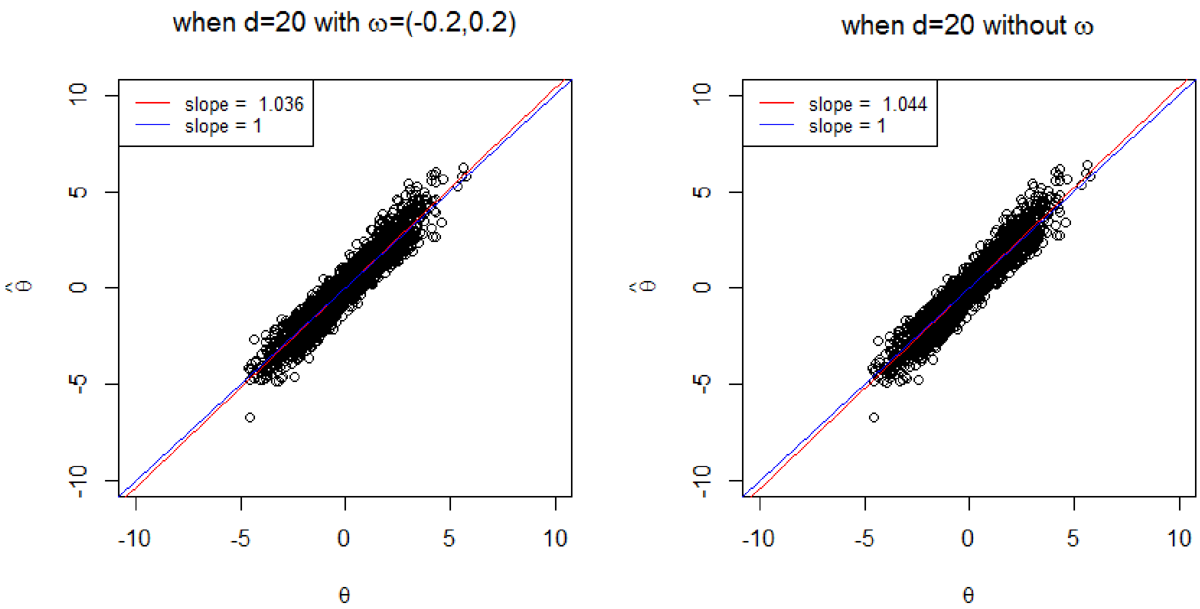

Figure 2: When d = 20. Red lines are slopes of ordinary least square estimators. Blue lines are line of $y = x$.

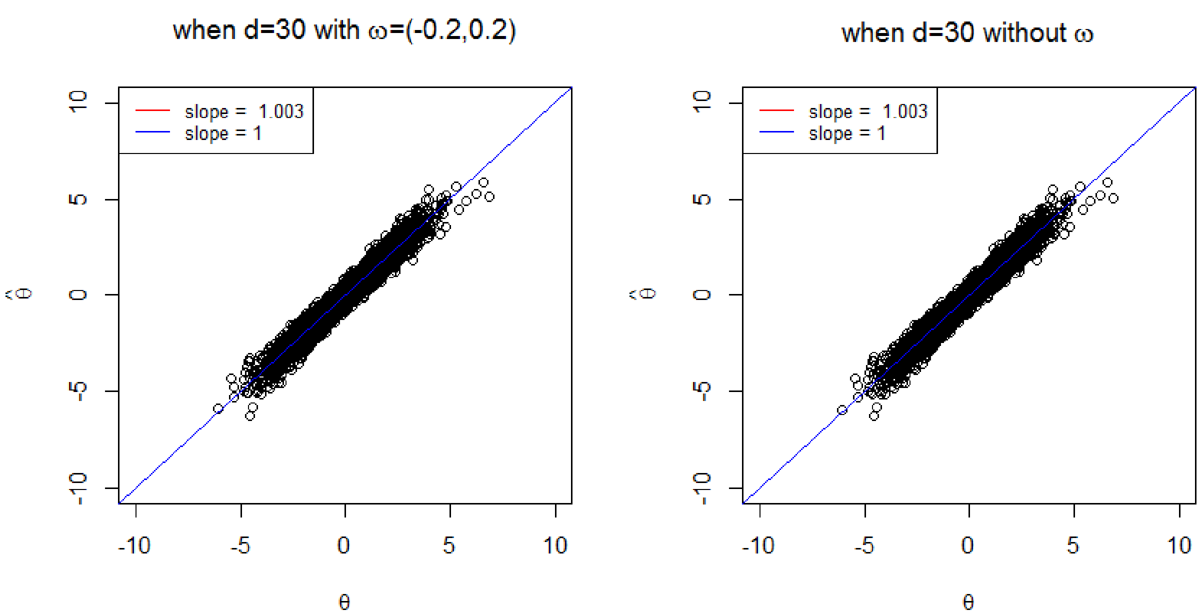

Figure 3: When d = 30. Red lines are slopes of ordinary least square estimators. Blue lines are line of $y = x$.

We can check that our estimators has a tendency to overestimate the true parameters.

The reason we had the opposite tendency in the 7th note is that I made mistake to calculate the slope of each scatter plot. I put Θ_{True} into response variables and $\hat{\Theta}$ into explanatory variables, which is the opposite direction from what I wanted to calculate. I checked we have the same overestimation tendency from the previous algorithm too if I calculate the slope in the right way. The overall comparisons between the previous algorithm and the current algorithm are shown in the following table. The table shows that the number of the iterations and the slope between the true parameters and the estimators are almost the same. However they have huge difference in the computation time. To find out the reason, I checked the core tensor updating inner function, which is the only different part between the two. For the previous algorithm, we use "optim" function to update the core tensors while we use "nlminb" function in the current algorithm. I checked that "optim" function needs 16.39 sec and 21 iteration numbers to update the core tensor. On the other hand, "nlminb" function takes 7.34 sec and 4 iteration numbers under the same condition. We can conclude that this computation time difference causes difference between the two main algorithms. However, I checked that "optim" and "nlminb" have the really close outputs. This means that the two main algorithm have the similar output for each update for the core tensor. Therefore, we can explain why the two main algorithms do not have big distinction for the output values and the iteration number in the main loop.

	$d = 20$			$d = 30$		
	Iteration #	Computation time	Slope	Iteration #	Computation time	Slope
Algorithm 1	7	121.79 sec	1.053	6	1470.78 sec	0.99698
Algorithm 2	7	45.5 sec	1.053	6	431 sec	0.99712

Table 1: Algorithm 1 is the previous algorithm and Algorithm 2 is the current algorithm using Hessian. We set the tolerance as 10^{-4}.

4 Rank selection with BIC

In practice, we have no information about true rank of the model. Estimating an appropriate rank is an important part to deal with. We use the Bayesian information criterion(BIC), and choose the rank that has the minimum BIC. Our estimated rank can be written as

$$\hat{R} = \underset{R\in\mathbb{R}_+}{\arg\min}\, BIC(R) = \underset{R\in\mathbb{R}_+}{\arg\min}(-2\mathcal{L}_{\mathcal{Y}}(\hat{\Theta}(R)) + \big(\prod_k r_k + \sum_k (d_k r_k - r_k^2)\big)\log(\prod_k d_k).$$

where $\hat{\Theta}(R)$ is the estimated tensor Θ under the rank size R. The front term of log is the effective number of parameters and we proved this formula in the 7th note. We simulated the tensor rank selection by BIC. We considered the tensor dimension $d = 20$ case and varied the rank $R \in \{3, 4\}$. The following table shows the BIC values for each rank according to true rank R. BICs with the close ranks to true rank have the small values as we expected.

	$R = 2$	$R = 3$	$R = 4$	$R = 5$	$R = 6$
$R_{\text{True}} = 3$	14483.72	14559.58	15067.75	15688.72	16749.21
$R_{\text{True}} = 4$	14267.40	13985.33	13818.74	14448.10	15219.82

Table 2: BIC values in ordinal tensor decomposition according to true rank and various ranks.

5 Algorithms

5.1 Extended angle simulation

```
library(rTensor)
library(pracma)
B_1 = matrix(rnorm(20*3),nrow = 20)
B_2 = matrix(rnorm(20*3),nrow = 20)
B_3 = matrix(rnorm(20*3),nrow = 20)
C = as.tensor(array(rnorm(3^3),dim = c(3,3,3)))
X = ttm(ttm(ttm(C,B_1,1),B_2,2),B_3,3)
sd = 0.02*1:25
result = data.frame(matrix(0,nrow = 75, ncol =6))
names(result) <- c("sd","angle1","angle2","angle3","method","MSE")


for (i in 1:25) {
  s=sd[i]
```

```
  result[i,1] = s
  result[i+25,1] = s
  result[i+50,1] = s
  for (j in 1:200) {
    set.seed(j)
    e = as.tensor(array(rnorm(8000,mean =0,sd = s),dim = c(20,20,20)))
    D = X+e
    est1 = tensor_svd(D,3,3,3,0)
    est2 = tensor_svd3(D,3,3,3,0)
    est3 = tensor_svd4(D,3,3,3,0)
    result[i,2] <- result[i,2]+subspace(est1$U[[1]],B_1)
    result[i,3] <- result[i,3]+subspace(est1$U[[2]],B_2)
    result[i,4] <- result[i,4]+subspace(est1$U[[3]],B_3)
    result[i,6] <- result[i,6]+tensor_resid(X,est1)
    result[i+25,2] <- result[i+25,2]+subspace(est2$U[[1]],B_1)
    result[i+25,3] <- result[i+25,3]+subspace(est2$U[[2]],B_2)
    result[i+25,4] <- result[i+25,4]+subspace(est2$U[[3]],B_3)
    result[i+25,6] <- result[i+25,6]+tensor_resid(X,est2)
    result[i+50,2] <- result[i+50,2]+subspace(est3$U[[1]],B_1)
    result[i+50,3] <- result[i+50,3]+subspace(est3$U[[2]],B_2)
    result[i+50,4] <- result[i+50,4]+subspace(est3$U[[3]],B_3)
    result[i+50,6] <- result[i+50,6]+tensor_resid(X,est3)
  }
  result[i,5] = "Method 1"
  result[i+25,5] = "Method 2"
  result[i+50,5]= 'Method 3'
}
result[,2:4] <- result[,2:4]/200
result[,6] <- result[,6]/200

library(gridExtra)
library(ggplot2)
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(expression(theta[1]))
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(expression(theta[2]))
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(expression(theta[3]))
grid.arrange(g1,g2,g3,g4)
```

5.2 New ordinal tensor algorithms

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

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))
}

#Hessian function
Hessi = function(A_1,W4,ttnsr,omega){
  thet =W4%*%c(A_1)
  p1 = logistic(thet + omega[1])
  p2 = logistic(thet + omega[2])
  q1 = p1*(1-p1)
```

```
  q2 = p2*(1-p2)+p1*(1-p1)
  q3 = p2*(1-p2)
  H = t(W4[which(c(ttnsr)==1),])%*%diag(q1[which(c(ttnsr)==1)])%*%W4[which
    (c(ttnsr)==1),]+
     t(W4[which(c(ttnsr)==2),])%*%diag(q2[which(c(ttnsr)==2)])%*%W4[which(c
    (ttnsr)==2),]+
     t(W4[which(c(ttnsr)==3),])%*%diag(q3[which(c(ttnsr)==3)])%*%W4[which(c
    (ttnsr)==3),]
  return(H)
}

#Function
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]))))
}

#Gradient
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)
}

optim(Cvec,h,g,method = "BFGS")
nlminb(Cvec,h,g,H)

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)
  H <- function(x) Hessi(x,W,ttnsr,omega)
  d <- nlminb(Cvec,h,g,H)
  C <- new("Tensor",C@num_modes,C@modes,data =d$par)

  return(C)
}

#previous core tensor updating algorithm.
prevcorecomb = 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)
```

```
  H <- function(x) Hessi(x,W,ttnsr,omega)


  if (alph==TRUE) {
    d <- nlminb(Cvec,h,g,H)
    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)
}




#####################################################################
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,c(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,c(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)
      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)
}


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,c(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

      # 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,c(ttnsr),omega)
```

```
      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)
}
```

6 Stochastic gradient method.

```
fit_ordinal = function(ttnsr,C,A_1,A_2,A_3,omega,alph = TRUE){
  alphbound <- alph+10^-4
  Batchsize <- 1000
  ransample <- sample(1:length(c(ttnsr)),Batchsize)
  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[ransample,],c(ttnsr)[ransample],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[ransample,],c(ttnsr)[ransample],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)
      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)
}
```