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 $s_{min}(C_{(i)}) \sqrt{\mathbb{1}_{r_i}^T (\Omega^{(i)})^T P^{(i)} (P^{(i)})^T \Omega^{(i)} \mathbb{1}_{r_i}} >> \sigma \sqrt{r_i \max(d_i, \frac{\Pi_{j \neq i} d_j}{d_i})} \|\Omega^{(i)}\|_\sigma$ as $d_1, \cdots, d_N \to \infty$, where $s_{min}(C_{(i)})$ is the smallest singular value of $C_{(i)}$, $P^{(i)} = [(M^{(N)} \otimes \cdots \otimes M^{(i+1)} \otimes M^{(i-1)} \otimes \cdots \otimes M^{(1)}) V_{r_i}^{(i)}]$ 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,

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

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.

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

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.

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

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.

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

for some x and y.

To get the numerator part in (4),

$$\|A_{(1)}\Omega^{(1)}x\|_2 = \|M^{(1)}C_{(1)}B^T\Omega^{(1)}x\|_2$$

$$\overset{(i)}{=} \|C_{(1)}B^T\Omega^{(1)}x\|_2$$

$$\overset{(ii)}{=} \|U^{(1)}\Sigma^{(1)}(V^{(1)})^T B^T\Omega^{(1)}x\|_2$$

$$= \|\Sigma^{(1)}(V_{r_1}^{(1)})^T B^T\Omega^{(1)}x\|_2$$

$$\geq s_{\min}(C_{(1)})\|(V_{r_1}^{(1)})^T B^T\Omega^{(1)}x\|_2$$

$$= s_{\min}(C_{(1)})\|(P^{(1)})^T\Omega^{(1)}x\|_2$$

$$\overset{(iii)}{=} s_{\min}(C_{(1)})\frac{1}{\sqrt{r_1}}\|(P^{(1)})^T\Omega^{(1)}\mathbb{1}_{r_1}\|_2$$

$$= \frac{s_{\min}(C_{(1)})\sqrt{\mathbb{1}_{r_1}^T(\Omega^{(1)})^T P^{(1)}(P^{(1)})^T\Omega^{(1)}\mathbb{1}_{r_1}}}{\sqrt{r_1}}.$$

(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.*

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

For future work, you could also think about Corollary for
1. KhtriRao-type Gaussian test matrices.
2. Count sketch test matrices.

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

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,

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

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

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

Secondly, we have

$$\begin{aligned}
\|\Omega^{(1)}\|_\sigma &\geq \|\Omega^{(1)} y\|_2 \quad \text{where } y \in \mathbb{R}^{r_1} \text{ such that } \|y\|_2 = 1 \\
&\overset{\text{d}}{=} \sqrt{N_{d_2 d_3}(0, I_{d_2 d_3})^T N_{d_2 d_3}(0, I_{d_2 d_3})} \\
&\overset{\text{d}}{=} \sqrt{\chi_{d_2 d_3}^2} \\
&\asymp (1 + o(1)) \sqrt{d_2 d_3}.
\end{aligned}$$

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

$$s_{\min}(C_{(1)}) \sqrt{\chi_{r_1}^2} >> \sigma \sqrt{\max(d_1, d_2 d_3) d_2 d_3}. \tag{7}$$

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.

$$\begin{aligned}
P\big(s_{\min}(C_{(1)}) \sqrt{\chi_{r_1}^2} > \sigma \sqrt{\max(d_1, d_2 d_3) d_2 d_3}\big) &= P\big(\chi_{r_1}^2 \geq \frac{\sigma^2 d_2 d_3 \max(d_1, d_2 d_3)}{s_{min}(C_{(1)})^2}\big) \\
&\overset{(i)}{\leq} 1 - \big(\lambda e^{1-\lambda}\big)^{\frac{r_1}{2}}.
\end{aligned}$$

$\square$

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,

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

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 $\mathcal{C}, B_1, B_2, B_3$ are i.i.d. drawn from N(0, 1). 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 MSE. However, our simulation does not show consistent result for the principal angles. I think we need to find a good reason for this phenomena.

See correction on code (page 9). Normalization may fix the issue. Please convince yourself of the reason.

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
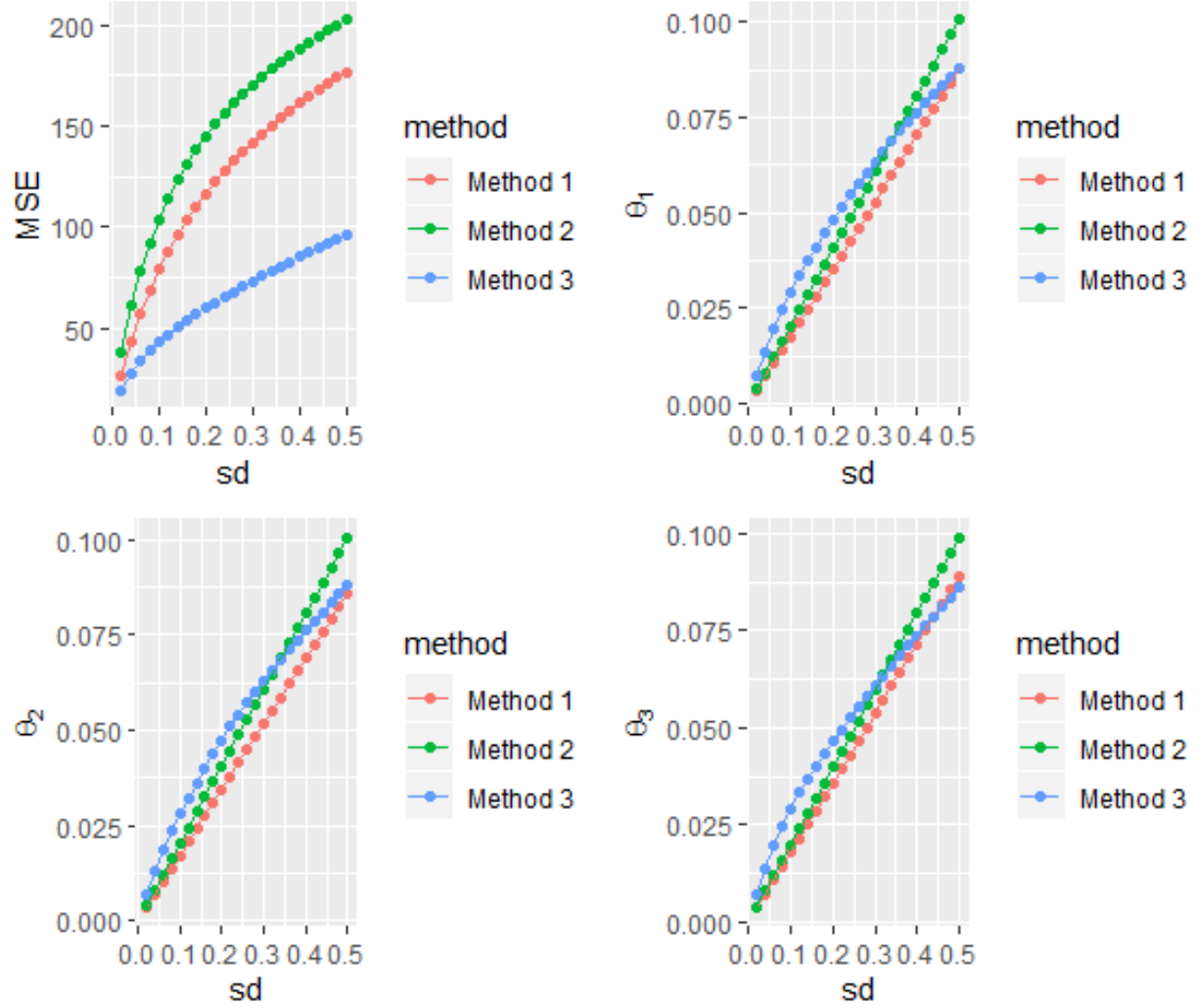
Can I take a look at the SGD script?

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 but Method 1 has better for the principal angles thought they have minor different values

time dramatically for updating the core tensor. Also, it converged with the less the number 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.

Please investigate:
1. Does the speed-up come from (1) fewer iterations, or (2) faster update per iteration?
2. Could you remind me the computational time (# iteration and per-iter time) for previous algorithm?
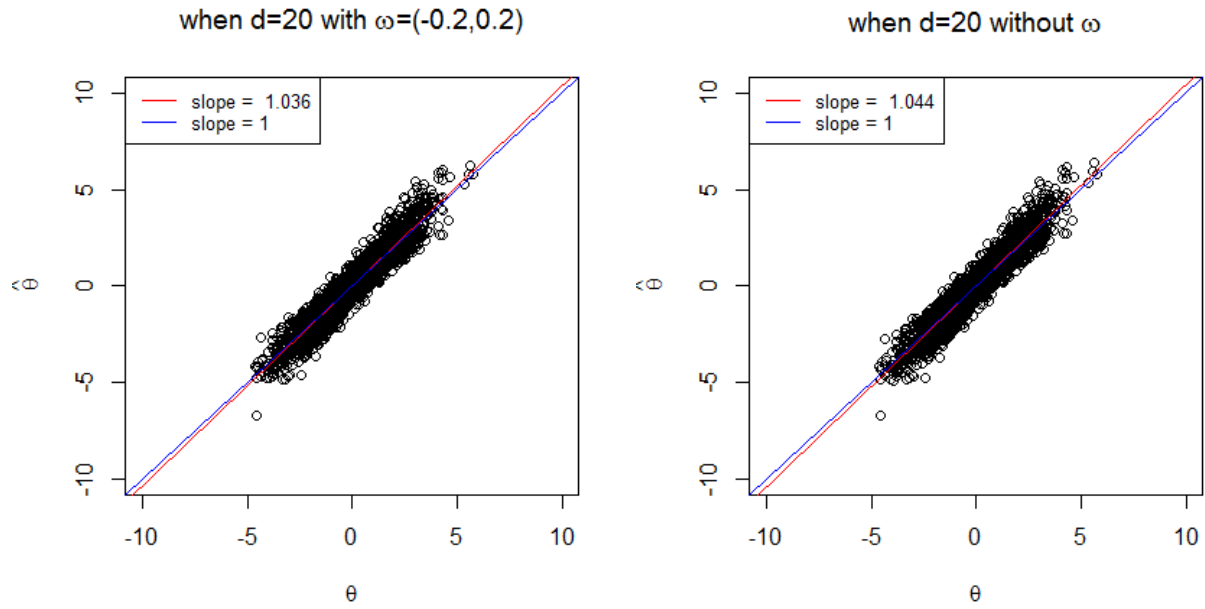**3. Investigate the effect of initialization.**

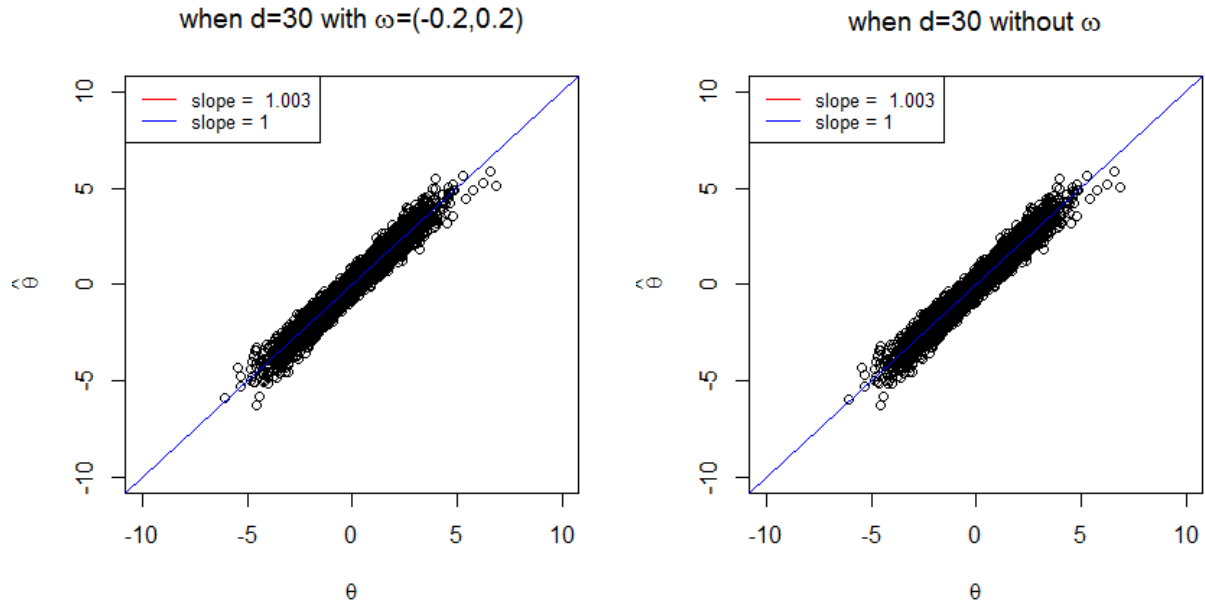Figure 2: When d = 20. Red lines are slopes of ordinary least square estimators. Blue lines are line of $y = x$.

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. Another thing to notice is the new algorithm performs better when $d = 20$ compared to the previous one. I think Hessian function played major rule to calculate the optimizer with more accuracy and less computation time.

4   Algorithms

4.1   Extended angle simulation

Please generate matrices with orthonormal columns.
E.g. B_1=randortho(20, type = c("orthonormal"))[,1:r]

```
1  library(rTensor)
2  library(pracma)
3  B_1 = matrix(rnorm(20*3),nrow = 20)
4  B_2 = matrix(rnorm(20*3),nrow = 20)
5  B_3 = matrix(rnorm(20*3),nrow = 20)
6  C = as.tensor(array(rnorm(3^3),dim = c(3,3,3)))
7  X = ttm(ttm(ttm(C,B_1,1),B_2,2),B_3,3)
8  sd = 0.02*1:25
9  result = data.frame(matrix(0,nrow = 75, ncol =6))
10 names(result) <- c("sd","angle1","angle2","angle3","method","MSE")
11
12
13 for (i in 1:25) {
14   s=sd[i]
15   result[i,1] = s
16   result[i+25,1] = s
17   result[i+50,1] = s
18   for (j in 1:200) {
19     set.seed(j)
20     e = as.tensor(array(rnorm(8000,mean =0,sd = s),dim = c(20,20,20)))
21     D = X+e
22     est1 = tensor_svd(D,3,3,3,0)
23     est2 = tensor_svd3(D,3,3,3,0)
24     est3 = tensor_svd4(D,3,3,3,0)
```

```
25    result[i,2] <- result[i,2]+subspace(est1$U[[1]],B_1)
26    result[i,3] <- result[i,3]+subspace(est1$U[[2]],B_2)
27    result[i,4] <- result[i,4]+subspace(est1$U[[3]],B_3)
28    result[i,6] <- result[i,6]+tensor_resid(X,est1)
29    result[i+25,2] <- result[i+25,2]+subspace(est2$U[[1]],B_1)
30    result[i+25,3] <- result[i+25,3]+subspace(est2$U[[2]],B_2)
31    result[i+25,4] <- result[i+25,4]+subspace(est2$U[[3]],B_3)
32    result[i+25,6] <- result[i+25,6]+tensor_resid(X,est2)
33    result[i+50,2] <- result[i+50,2]+subspace(est3$U[[1]],B_1)
34    result[i+50,3] <- result[i+50,3]+subspace(est3$U[[2]],B_2)
35    result[i+50,4] <- result[i+50,4]+subspace(est3$U[[3]],B_3)
36    result[i+50,6] <- result[i+50,6]+tensor_resid(X,est3)
37  }
38  result[i,5] = "Method 1"
39  result[i+25,5] = "Method 2"
40  result[i+50,5]= 'Method 3'
41 }
42 result[,2:4] <- result[,2:4]/200
43 result[,6] <- result[,6]/200
44
45 library(gridExtra)
46 library(ggplot2)
47 g1 <- ggplot(data = result, aes(x=sd,y = MSE ,color = method))+
48   geom_point(aes(x=sd, y = MSE))+geom_line(aes(x=sd, y = MSE))
49 g2 <- ggplot(data = result, aes(x=sd,y = abs(angle1) ,color = method))+
50   geom_point(aes(x=sd, y = abs(angle1)))+geom_line(aes(x=sd, y = abs(
     angle1)))+ylab(expression(theta[1]))
51 g3 <- ggplot(data = result, aes(x=sd,y = abs(angle2) ,color = method))+
52   geom_point(aes(x=sd, y = abs(angle2)))+geom_line(aes(x=sd, y = abs(
     angle2)))+ylab(expression(theta[2]))
53 g4 <- ggplot(data = result, aes(x=sd,y = abs(angle3) ,color = method))+
54   geom_point(aes(x=sd, y = abs(angle3)))+geom_line(aes(x=sd, y = abs(
     angle3)))+ylab(expression(theta[3]))
55 grid.arrange(g1,g2,g3,g4)
```

4.2   New ordinal tensor algorithms

```
1 library(MASS)
```

```r
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)
}
```

```r
#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,],
```

```r
73                                                    function(x) h1(x,W,tnsr1
    [i,],omega),function(x) g1(x,W,tnsr1[i,],omega),
74                                                    ui = rbind(W,-W),ci =
    rep(-alph,2*nrow(W)),method = "BFGS")$par})
75      nA <- matrix(unlist(l),nrow = nrow(A),byrow = T)
76    }
77    return(nA)
78  }
79
80  optim(Cvec,h,g,method = "BFGS")
81  nlminb(Cvec,h,g,H)
82
83  corecomb = function(C,W,ttnsr,omega,alph=TRUE){
84    Cvec <- c(C@data)
85    h <- function(x) h1(x,W,ttnsr,omega)
86    g <- function(x) g1(x,W,ttnsr,omega)
87    H <- function(x) Hessi(x,W,ttnsr,omega)
88    d <- nlminb(Cvec,h,g,H)
89    C <- new("Tensor",C@num_modes,C@modes,data =d$par)
90
91    return(C)
92  }
93
94  #previous core tensor updating algorithm.
95  prevcorecomb = function(C,W,ttnsr,omega,alph=TRUE){
96    Cvec <- c(C@data)
97    h <- function(x) h1(x,W,ttnsr,omega)
98    g <- function(x) g1(x,W,ttnsr,omega)
99    H <- function(x) Hessi(x,W,ttnsr,omega)
100
101
102    if (alph==TRUE) {
103      d <- nlminb(Cvec,h,g,H)
104      C <- new("Tensor",C@num_modes,C@modes,data =d$par)
105    }else{
106      d <- constrOptim(Cvec,h,g,ui = rbind(W,-W),ci = rep(-alph,2*nrow(W)),
    method = "BFGS")
107      C <- new("Tensor",C@num_modes,C@modes,data =d$par)
108    }
```

13

```r
109    return(C)
110 }
111
112
113
114
115 ###################################################################
116 fit_ordinal = function(ttnsr,C,A_1,A_2,A_3,omega,alph = TRUE){
117   alphbound <- alph+10^-4
118   result = list()
119   error<- 3
120   iter = 0
121   d1 <- nrow(A_1); d2 <- nrow(A_2); d3 <- nrow(A_3)
122   r1 <- ncol(A_1); r2 <- ncol(A_2); r3 <- ncol(A_3)
123   if (alph == TRUE) {
124     while ((error > 10^-4)&(iter<50) ) {
125       iter = iter +1
126
127       #update A_1
128       prevtheta <- ttm(ttm(ttm(C,A_1,1),A_2,2),A_3,3)
129       prev <- likelihood(ttnsr,prevtheta,omega)
130       W1 =kronecker(A_3,A_2)%*%t(k_unfold(C,1)@data)
131       A_1 <- comb(A_1,W1,ttnsr,1,omega)
132
133
134       # update A_2
135       W2 <- kronecker(A_3,A_1)%*%t(k_unfold(C,2)@data)
136       A_2 <- comb(A_2,W2,ttnsr,2,omega)
137
138       # update A_3
139       W3 <- kronecker(A_2,A_1)%*%t(k_unfold(C,3)@data)
140       A_3 <- comb(A_3,W3,ttnsr,3,omega)
141
142       # update C
143       W4 <- kronecker(kronecker(A_3,A_2),A_1)
144       C <- corecomb(C,W4,c(ttnsr),omega)
145       theta <- ttm(ttm(ttm(C,A_1,1),A_2,2),A_3,3)
146       new <- likelihood(ttnsr,theta,omega)
147       (error <- abs((new-prev)/prev))
```

```
148       }
149     }else{
150       while ((error > 10^-4)&(iter<50) ) {
151         iter = iter +1
152
153         #update A_1
154         prevtheta <- ttm(ttm(ttm(C,A_1,1),A_2,2),A_3,3)
155         prev <- likelihood(ttnsr,prevtheta,omega)
156         W1 =kronecker(A_3,A_2)%*%t(k_unfold(C,1)@data)
157         A_1 <- comb(A_1,W1,ttnsr,1,omega,alphbound)
158         if(max(abs(ttm(ttm(ttm(C,A_1,1),A_2,2),A_3,3)@data))>=alph) break
159
160
161         # update A_2
162         W2 <- kronecker(A_3,A_1)%*%t(k_unfold(C,2)@data)
163         A_2 <- comb(A_2,W2,ttnsr,2,omega,alphbound)
164         if(max(abs(ttm(ttm(ttm(C,A_1,1),A_2,2),A_3,3)@data))>=alph) break
165
166         # update A_3
167         W3 <- kronecker(A_2,A_1)%*%t(k_unfold(C,3)@data)
168         A_3 <- comb(A_3,W3,ttnsr,3,omega,alphbound)
169         if(max(abs(ttm(ttm(ttm(C,A_1,1),A_2,2),A_3,3)@data))>=alph) break
170
171         # update C
172         W4 <- kronecker(kronecker(A_3,A_2),A_1)
173         C <- corecomb(C,W4,c(ttnsr),omega)
174         theta <- ttm(ttm(ttm(C,A_1,1),A_2,2),A_3,3)
175         new <- likelihood(ttnsr,theta,omega)
176         error <- abs((new-prev)/prev)
177         if(max(abs(ttm(ttm(ttm(C,A_1,1),A_2,2),A_3,3)@data))>=alph) break
178       }
179     }
180
181     result$C <- C; result$A_1 <- A_1; result$A_2 <- A_2; result$A_3 <- A_3
182     result$iteration <- iter
183     return(result)
184 }
185
186
```

```r
187 fit_ordinal2 = function(ttnsr,C,A_1,A_2,A_3,omega=TRUE,alph = TRUE){
188   omega <- sort(rnorm(2))
189   alphbound <- alph+10^-4
190   result = list()
191   error<- 3
192   iter = 0
193   d1 <- nrow(A_1); d2 <- nrow(A_2); d3 <- nrow(A_3)
194   r1 <- ncol(A_1); r2 <- ncol(A_2); r3 <- ncol(A_3)
195   if (alph == TRUE) {
196     while ((error > 10^-4)&(iter<50) ) {
197       iter = iter +1
198
199       #update A_1
200       prevtheta <- ttm(ttm(ttm(C,A_1,1),A_2,2),A_3,3)
201       prev <- likelihood(ttnsr,prevtheta,omega)
202       W1 =kronecker(A_3,A_2)%*%t(k_unfold(C,1)@data)
203       A_1 <- comb(A_1,W1,ttnsr,1,omega)
204
205
206       # update A_2
207       W2 <- kronecker(A_3,A_1)%*%t(k_unfold(C,2)@data)
208       A_2 <- comb(A_2,W2,ttnsr,2,omega)
209
210       # update A_3
211       W3 <- kronecker(A_2,A_1)%*%t(k_unfold(C,3)@data)
212       A_3 <- comb(A_3,W3,ttnsr,3,omega)
213
214       # update C
215       W4 <- kronecker(kronecker(A_3,A_2),A_1)
216       C <- corecomb(C,W4,c(ttnsr),omega)
217
218       #update omega
219       theta <- ttm(ttm(ttm(C,A_1,1),A_2,2),A_3,3)
220       m <- polr(as.factor(c(ttnsr))~offset(-c(theta@data)))
221       omega <- m$zeta
222
223
224
225       theta <- ttm(ttm(ttm(C,A_1,1),A_2,2),A_3,3)
```

```r
      new <- likelihood(ttnsr,theta,omega)
      error <- abs((new-prev)/prev)
  }
}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)
```

```r
    }
  }

  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)
}
```