# Algorithms and simulations of Tropps' paper

Chanwoo Lee

## 1 2-dimensional matrix case

### 1.1 Algorithm part

Brief algorithm for 2-dimensional matrix svd I used as follows and real codes for this will be in Appendix.

**Algorithm 1** SVD with randomness

```
1: procedure
2:     Part A (Getting $m \times l$ orthonormal matrix $Q$ from given $m \times n$ matrix $A$)
3:     Draw an $m \times l$ Gaussian random matrix $\Omega$
4:     Form the $m \times l$ matrix $Y = A \times \Omega$
5:     Construct an $m \times l$ matrix $Q$ using QR decomposition on$Y$
6:     Part B (Getting approximated SVD using Q)
7:     Form the matrix $B = Q^*A$
8:     Compute an SVD of the small matrix: $B = \tilde{U}\Sigma V^*$
9:     Form the orthonormal matrix $U = Q\tilde{U}$
```

### 1.2 Simulation part

There are a few simulations I did to answer my own questions.

1. How much oversampling p affects approximation accuracy? what about computation cost?
2. Does dimension of column or row matrix affect accuracy of approximation?
3. Does theoretical error bound in the paper hold true for simulations?
4. How much faster it is to use svd with randomness than with exact svd?

**Answers to above questions**

1. Simulation procedure:
   (a) I made arbitrary matrix whose rank equals 150, the number of rows is 1000 and the number of columns is 500

(b) Our targeting rank of the matrix is 130 and I varied oversampling parameter p from 0 to 20.

(c) I checked Frobenius norm of difference between objective matrix and approximated SVD matrix according to each p. Also, I checked iterating time to get approximated matrix and compared relationship between oversampling p and each variables

Result:

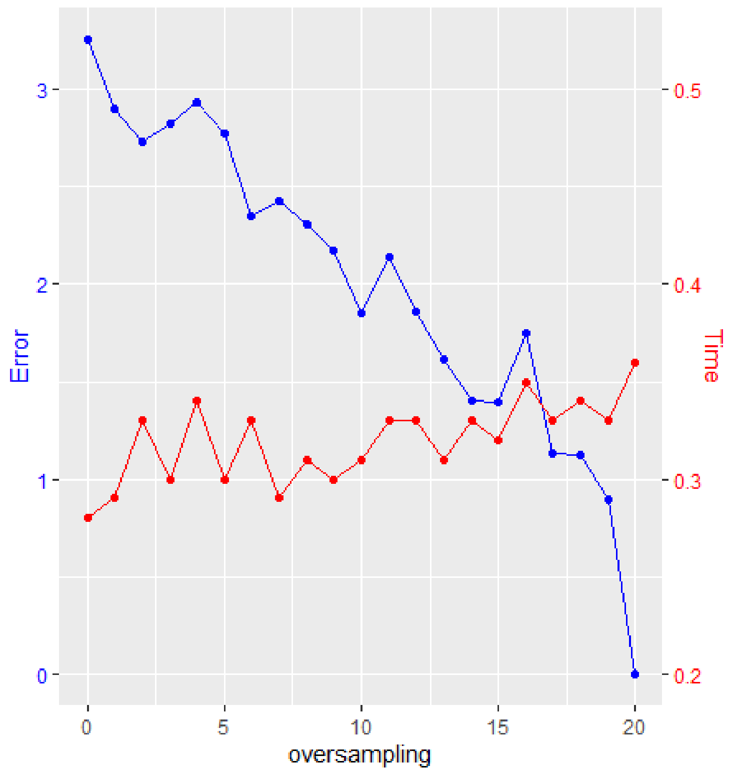

Figure 1: Oversampling vs Error or Iteration time

As you can see above Figure 1, there is a tendency that oversampling increases computation time but it works opposite way with regards to error. So finding out suitable oversampling size p would be another issue. One thing I want to point out is that when oversampling size becomes 20, error becomes almost 0 in Figure 1 because sum of target rank and oversampling size becomes exactly same with actual matrix rank.

2. Simulation Procedure:

(a) I made arbitrary matrices whose size are $(100, 100), (200, 100), \cdots, (1000, 100)$ and rank is constant as 50 with same singular values.

(b) I calculated errors for each matrix approximation with targer rank 42 and oversampling size 5

(c) I also calculated theoretical expected error bound.

(d) I got matrices whose size are $(100, 100), (100, 200), \cdots, (100, 1000)$ by transposing matrices on (a)

(e) Repeat (b) on matrices on (d)

Result:

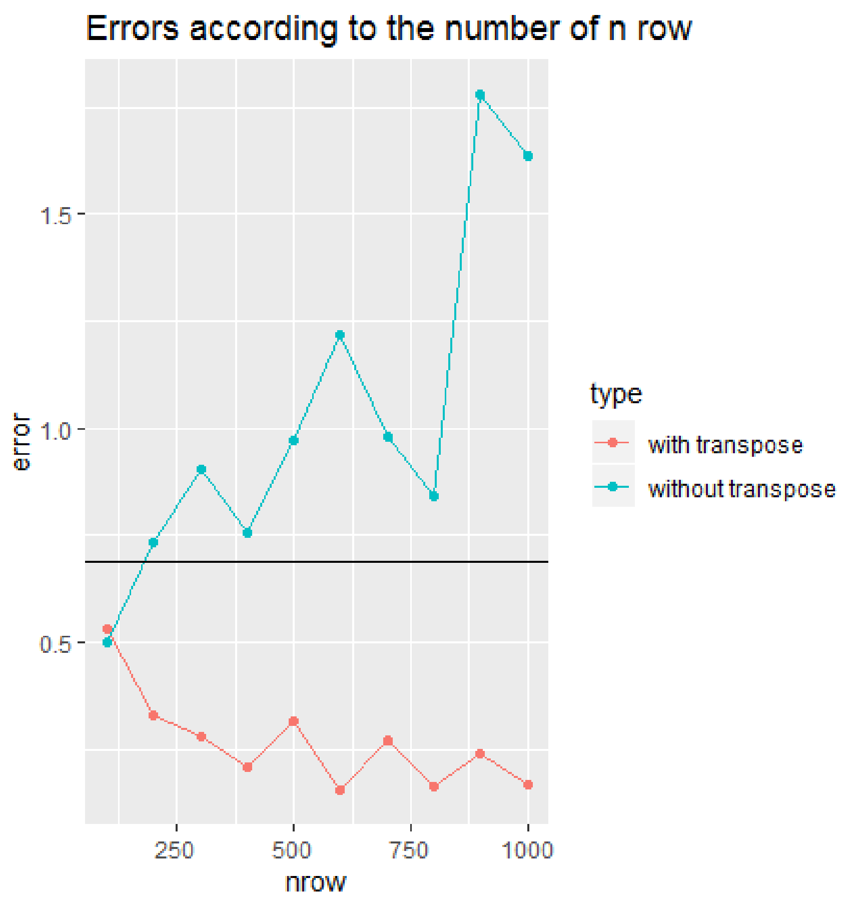

Figure 2: Error vs n-row or n-columns

It's really interesting to observe that matrices with large columns work better than matrices with large rows. To formulate above phenomena, let $A \in \mathcal{R}^{m\times n}$ be a given matrix such that $m > n$. Let $Q_1$ be orthonormal basis for the range of the sample matrix $Y_1 = A\Omega_1$. Also for $A^T \in \mathcal{R}^{n\times n}$, let $Q_2$ be orthornormal basis for the range of the sample matrix $Y_2 = A^t\Omega_2$. Above phenomena says that

$$\|A - Q_1Q_1^TA\| \geq \|A^T - Q_2Q_2^TA^T\|$$

To recheck above phenomena, I did simulation again. Fixing the number of columns, I increased the number of rows and calculated each $\|A - Q_1Q_1^TA\|, \|A^T - Q_2Q_2^TA^T\|$. Result is similar to Figure 2. Also if their size of rows and columns are similar, they have close errors.

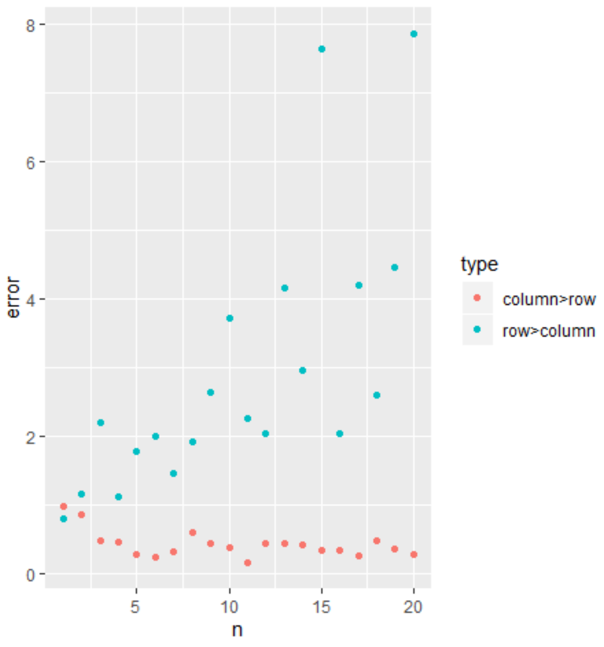

Figure 3: $50n \times 50$ matrix out put

My first conjecture of this phenomena is that with large rows, our random Gaussian matrix fails to capture image space of object matrix because of huge row's dimensions. However, a problem is theoretical error bound doesn't hold when size of rows are bigger than size of columns. I tried to check that there are some theorem's conditions that don't hold in larger row case. I also checked proof of theorems again. But I couldn't find any deviation. I will have to study more about this issue next week and find out exact reason for this.

3. Simulation procedure:

   (a) I made $1000 \times 500$ matrix with rank 150

   (b) I approximated matrix using the paper's method with $p = 5$ and target rank 140 and calculated norm difference between two.

   (c) Repeated (b) 1000 times.

   (d) Averaged errors and check for theoretical expected error bound and probabilistic error bound.

   Result: Theoretical expectation error bound for this setting is 1.244751 and I choose $u = 2, t = 3$ where probabilistic error bound and maximum failure rate look moderate based on following figures and got bound 15.4196646 with failure rate at most 0.2912467 Our 1000 simulations result statistics is as follows:

$$max(\text{error}) = 1.895834, \quad min(\text{error}) = 0.866689, \quad mean(\text{error}) = 1.2492$$

   So we can conclude that theoretical error bound hold true in simulation case.
   As you can see below figure 3, however, probabilistic error bound and failure rate has

inverse proportional relationship and usually doesn't have good bound. So I think we need to find better bound which is more useful.

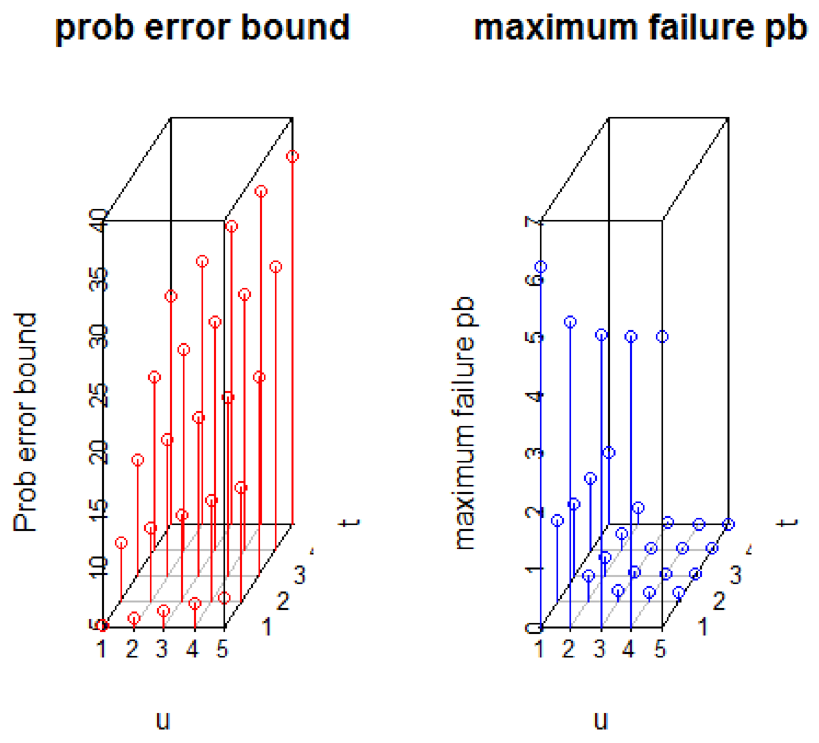

Figure 4: Prob error vs maximum failure rate according to u and t

4. For $1000 \times 500$ matrix, computing time of probabilistic method$= 0.24 + 0.18 = 0.42$sec but that of exact svd method$= 1.01$sec. I can guess that as dimension of matrix get larges, computation time difference get greater.

## 2 Tensor extension

### 2.1 Algorithm part

There are 2 ways to approximate tensor svd by generalizing Tropp's papaer. Brief algorithm for first one is as follows.

As far as I remember, the second method looks like this.

**Algorithm 2** Approx tensor SVD 1

```
1: procedure SVD($\mathcal{A}$)
2:     Step A: Approximate SVD of $\mathcal{A}$
3:     for $n \leftarrow 1 : N$ do
4:         Unfold $\mathcal{A}$ as $A_{(n)}$
5:         Generate an $I_n \times I_1 \cdots I_{n-1} I_{n+1} \cdots I_N$ Gaussian test matrix $\Omega^{(n)}$
6:         For $\mathbf{Y^{(n)}} = \mathbf{A_{(n)}} \Omega^{(n)}$
7:         Construct a matrix $\mathbf{Q}^{(n)}$ whose columns form an orthonormal basis for the range
   of $\mathbf{Y^{(n)}}$
8:         Form $\mathbf{P_{Y^{(n)}}} = \mathbf{Q^{(n)}} \mathbf{Q^{(n)*}}$
9:     get $\hat{\mathcal{A}} = \mathcal{A} \times_1 P_{Y^{(1)}} \times_2 P_{Y^{(2)}} \cdots \times_N P_{Y^{(N)}}$
10:    Step B: Get approximated SVD
11:    $\mathcal{S} = \mathcal{A} \times_1 Q^{(1)*} \times_2 Q^{(2)*} \cdots \times_N Q^{(N)*}$
12:    for $i \leftarrow 1 : N$ do
13:        $U^{(i)} = Q^{(i)}$
14:    return $(\mathcal{S}, U^{(1)} \cdots U^{(n)})$
```

**Algorithm 3** Approx tensor SVD 2

```
1: procedure SVD($\mathcal{A}$)
2:     Step A: Approximate SVD of $\mathcal{A}$
3:     for $i \leftarrow 1 : N$ do
4:         Get Gaussian test matrix $\Omega_n$ whose size is $I_i \times (k_i + p)$
5:         Form $A^{(i)} = \text{Unfold}_i(\mathcal{A} \times_1 \Omega_1^* \times \cdots \times_{i-1} \Omega_{i-1}^* \times_{i+1} \Omega_{i+1}^* \times \cdots \times_N \Omega_N^*)$
6:         Find a matrix $Q^{(i)}$ whose size is $I_i \times (k_i + p)$
7:         $U^{(i)} = Q^{(i)}$
8:     get $\mathcal{S} = \mathcal{A} \times_1 Q^{(1)*} \times_2 Q^{(2)*} \cdots \times_N Q^{(N)*}$
9:     return $(\mathcal{S}, U^{(1)} \cdots U^{(n)})$
```

The exact codes for those algorithms are on appendix.

## 2.2 Tensor modeling simulation

1. First, let's check our algorithm approximate an object tensor well comparing two different methods at the same time. Main procedure is as follows.

   (a) Make an object $100 \times 100 \times 100$ tensor whose each unfolding matrix has rank 20

   (b) Set target rank from 1 to 15 with oversampling size 5.

   (c) Do approximated SVD and calculate a norm difference from the object tensor for the first method.

   (d) Repeat for the second method.

   Result:

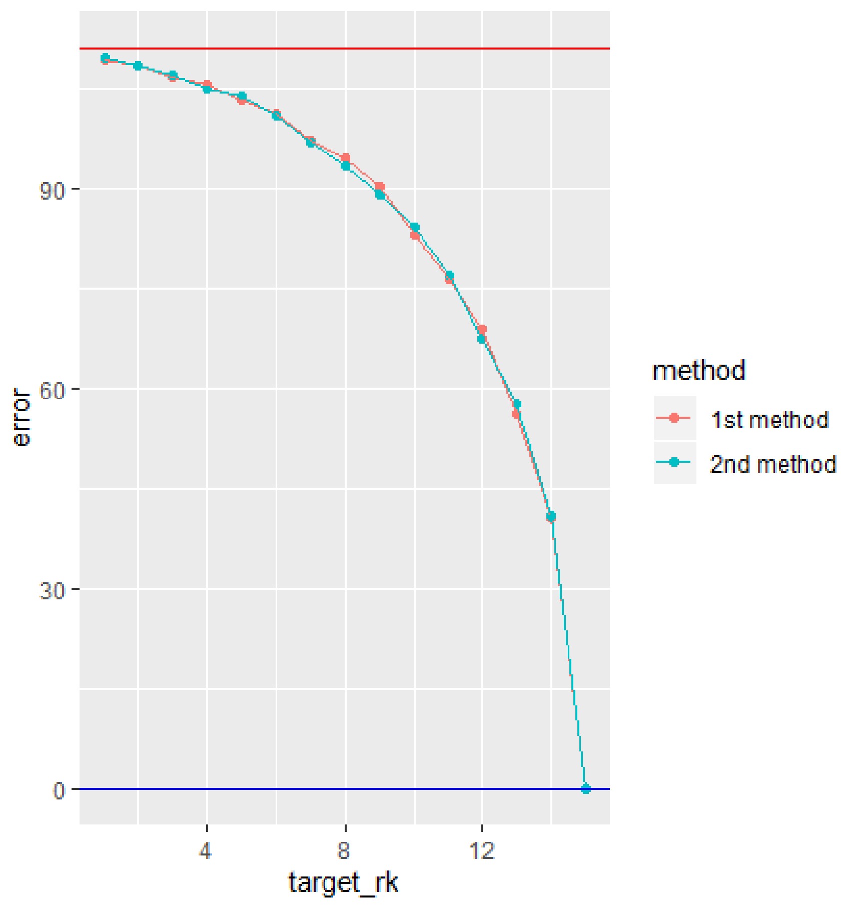

Figure 5: Approximation errors in tensor case

In Figure 4, Red line is Frobenius norm of the given tensor and blue line is horizontal line of 0. As you can see, errors converges to 0 as our target rank increases to real rank which means our algorithm works well. Also there is no huge difference between two methods and I think two methods are essentially same and difference comes from randomization.

2. Finding the best Tucker decomposition model
We want to estimate low rank signal $M$ from a given data $A$.
To make it clear, for $\epsilon \sim N(0,1)$ our model looks like this.

$$A = \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} g_{pqr} a_p \circ b_q \circ c_r + \epsilon$$

So our goal is to find good estimations for parameters having small

$$\|A - \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} \hat{g_{pqr}} \hat{a_p} \circ \hat{b_q} \circ \hat{c_r}\|$$

Let's see our tensor approximation give us good approximation for parameters. My simulation procedure is as follows.

(a) Make an object $100 \times 100 \times 100$ tensor whose each unfolding matrix has rank 20

(b) Make $100 \times 100 \times 100$ noise tensors having normal distribution with mean 0 standard deviation 1

(c) Change standard deviation from 1 to 0.005

(d) Calculate estimation for standard deviation and parameters for each standard deviation.

(e) Calculate$\|A - \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} \hat{g_{pqr}} \hat{a_p} \circ \hat{b_q} \circ \hat{c_r}\|$ for each case.

Result:

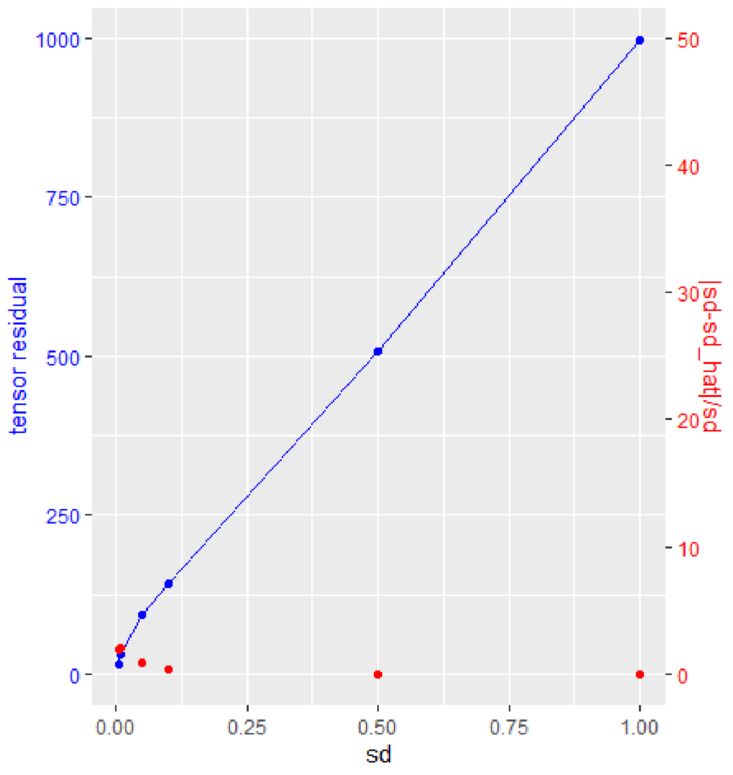

Figure 6: How good our approximation is with a noise

In Figure 5, Y-axis on the right side is $\frac{SD - \hat{SD}}{SD}$ and you can check that it is quite good at figuring out standard deviation of a noise. As you can see, tensor estimation becomes accurate when standard deviation of noise become small. One thing I want to mention is when standard deviation of noise equals 1, norm difference between target tensor and our estimation become extremely large. This is because our target tensor is also made by drawing standard Gaussian distribution, which means noise can hide our target tensor and we can't distinct between the target and the noise. I would say that with moderate variance of noise, we can estimate parameters quite well.

3. Binary case simulation: I am still working on this and I want to do more simulations in tensor setting.

## 3 Miscellaneous

### 3.1 Problem

Let $\mathcal{A} = C \times_1 M_1 \times_2 M_2 \times_3 M_3 \in \mathcal{R}^{d_1 \times d_2 \times d_3}$ where $C \in \mathcal{R}^{r_1 \times r_2 \times r_3}$ and $M_i \in \mathcal{R}^{d_i \times r_i}$ for each i

Suppose we have estimation $\hat{M}_1, \hat{M}_2, \hat{M}_3$ such that $\|M_i - \hat{M}_i\|_F \leq \epsilon$ for each i

Let $\hat{C} = \mathcal{A} \times_1 \hat{M}_1^{\ t} \times_2 \hat{M}_2^{\ t} \times_3 \hat{M}_3^{\ t}$, and $\hat{A} = \hat{C} \times_1 \hat{M}_1 \times_2 \hat{M}_2 \times_3 \hat{M}_3$

Then what can you get for error abound for $\|\hat{A} - \mathcal{A}\|_F$?

*Proof.* First, notice that for each i,

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

Main proof is as follows

$$\begin{aligned}
\|\mathcal{A} - \hat{A}\| &= \|\mathcal{A} - \hat{C} \times_1 \hat{M}_1 \times_2 \hat{M}_2 \times_3 \hat{M}_3\| = \|\mathcal{A} - \mathcal{A} \times_1 \hat{M}_1^{\ t} \times_2 \hat{M}_2^{\ t} \times_3 \hat{M}_3^{\ t} \times_1 \hat{M}_1 \times_2 \hat{M}_2 \times_3 \hat{M}_3\| \\
&= \|\mathcal{A} - \mathcal{A} \times_1 \hat{M}_1 \hat{M}_1^{\ t} \times_2 \hat{M}_2 \hat{M}_2^{\ t} \times_2 \hat{M}_2 \hat{M}_2^{\ t}\| \\
&= \|A_{(1)} - \hat{M}_1 \hat{M}_1^{\ t} A_{(1)} (\hat{M}_2 \hat{M}_2^{\ t} \otimes \hat{M}_3 \hat{M}_3^{\ t})\| \\
&= \|M_1 M_1^t A_{(1)} (M_2 M_2^t \otimes M_3 M_3^t) - \hat{M}_1 \hat{M}_1^{\ t} A_{(1)} (\hat{M}_2 \hat{M}_2^{\ t} \otimes \hat{M}_3 \hat{M}_3^{\ t})\| \\
&= \|(M_1 M_1^t - \hat{M}_1 \hat{M}_1^{\ t}) A_{(1)} (M_2 M_2^t \otimes M_3 M_3^t) + \hat{M}_1 \hat{M}_1^{\ t} A_{(1)} (M_2 M_2^t \otimes M_3 M_3^t - \hat{M}_2 \hat{M}_2^{\ t} \otimes \hat{M}_3 \hat{M}_3^{\ t})\| \\
&= \|(M_1 M_1^t - \hat{M}_1 \hat{M}_1^{\ t}) A_{(1)} (M_2 M_2^t \otimes M_3 M_3^t)\| \\
&\quad + \|\hat{M}_1 \hat{M}_1^{\ t} A_{(1)} (M_2 M_2^t \otimes M_3 M_3^t - \hat{M}_2 \hat{M}_2^{\ t} \otimes \hat{M}_3 \hat{M}_3^{\ t})\| \\
&\leq \|M_1 M_1^t - \hat{M}_1 \hat{M}_1^{\ t}\| \|A_{(1)}\| \|M_2 M_2^t \otimes M_3 M_3^t\| \\
&\quad + \|\hat{M}_1 \hat{M}_1^{\ t}\| \|A_{(1)}\| \|M_2 M_2^t \otimes M_3 M_3^t - \hat{M}_2 \hat{M}_2^{\ t} \otimes \hat{M}_3 \hat{M}_3^{\ t}\| \\
&\leq (\|M_1 M_1^t - \hat{M}_1 \hat{M}_1^{\ t}\| + \|M_2 M_2^t \otimes M_3 M_3^t - \hat{M}_2 \hat{M}_2^{\ t} \otimes \hat{M}_3 \hat{M}_3^{\ t}\|) \|\mathcal{A}\| \\
&\leq (\|M_1 M_1^t - \hat{M}_1 \hat{M}_1^{\ t}\| + \|(M_2 M_2^t - \hat{M}_2 \hat{M}_2^{\ t}) \otimes M_3 M_3^t\| + \|\hat{M}_2 \hat{M}_2^{\ t} \otimes (M_3 M_3^t - \hat{M}_3 \hat{M}_3^{\ t})\|) \|\mathcal{A}\| \\
&\leq (\|M_1 M_1^t - \hat{M}_1 \hat{M}_1^{\ t}\| + \|M_2 M_2^t - \hat{M}_2 \hat{M}_2^{\ t}\| + \|M_3 M_3^t - \hat{M}_3 \hat{M}_3^{\ t}\|) \|\mathcal{A}\| \\
&\leq \|\mathcal{A}\| (2\|M_1\| + 2\|M_2\| + 2\|M_3\| + 3\epsilon)\epsilon
\end{aligned}$$

□

### 3.2 Question.

1. I got a question about oversampling parameter: I can't understand what make it different when we use $l = k_1 + p_1$ and when we use $l = k_2 + p_2$ because I think they are the same in algorithm's respect but give different error bound. Also, when I checked proof of error bound, there's no logical difference as long as we are certain that $\Omega_1 = V_1^* \Omega$ has full rank

2. I spent a day to figure out a reason for having different performance according to column and row's size. But I couldn't get a good explanation for that.

### 3.3 To do list

1. I want to study more about above question 2.

2. I will do more simulations and modeling part in tensor generalization.

3. I will read *Learning from Binary Multiway Data: Probabilistic Tensor Decomposition and its Statistical Optimality*

## 4 Appendix: My R codes

### 4.1 Codes for SVD with randomness

```r

normf = function(y){
  return(as.numeric(sqrt(sum(y^2))))
}

inner = function(x,y){
  return(as.numeric(t(x)%*%y))
}

E_errorbound = function(k,p,sigma){
  return((1+k/(p-1))^{1/2}*(sqrt(sum(sigma[(k+1):length(sigma)]^2))))
}

P_errorbound = function(k,p,sigma,u,t){
  errbd = (1+t*sqrt(12*k/p))*(sqrt(sum(sigma[(k+1):length(sigma)]^2)))+u*t
   *exp(1)*sqrt(k+p)*sigma[k+1]/(p+1)
  fp = 5*t^-p +2*exp(-u^2/2)
  return(c(errbd,fp))
}


StepAm = function(A,r,p){
  m = nrow(A); n = ncol(A)
  q = 3
  l = r+p
  omega = matrix(rnorm(n*l),nrow = n, ncol = l)
  Y = A%*%omega
  Q = qr.Q(qr(Y))
  return(Q)
}

StepB = function(Q,A){
  B = svd(t(Q)%*%A)
  B$u = Q%*%B$u
  return(B)
}
```

## 4.2 Simulation: Oversampling vs Precision

```
result = matrix(nrow = 21, ncol =2)
result[,1] = 0:20
for (i in 0:20) {
  tic("Probabilistic method")
  B = StepAm(A,130,i)
  C = StepB(B,A)
  result[i+1,2] <- norm(A-C$u%*%diag(C$d)%*%t(C$v))
  print(norm(A-C$u%*%diag(C$d)%*%t(C$v)))
  toc()
}

result <- as.data.frame(result)
names(result) <- c("oversampling","error")
library(ggplot2)
result = cbind(result,time)

ggplot(result, aes(x=oversampling)) +
  geom_point(aes(y=error), col="blue") +
  geom_line(aes(y=error), col="blue") +
  geom_point(aes(y=10*(time-0.2)),col="red") +
  geom_line(aes(y=10*(time-0.2)),col="red")+
  scale_y_continuous("Error", sec.axis = sec_axis(~(.)/10+0.2, name = "
   Time")) +
  theme(
    axis.title.y.left=element_text(color="blue"),
    axis.text.y.left=element_text(color="blue"),
    axis.title.y.right=element_text(color="red"),
    axis.text.y.right=element_text(color="red")
  )
```

## 4.3 Simulation: Error vs nrow

```
sigma = c(sort(abs(rnorm(50)),decreasing = T),rep(0,50))
result = as.data.frame(matrix(nrow = 20, ncol = 3))
names(result) = c("nrow","error","type")



## different row
```

```r
for (i in 1:10) {
  nrow = i*100 ; ncol = 100
  result[i,1] = nrow
  A = randortho(nrow)[,1:100]%*%diag(sigma)%*%randortho(ncol)
  tic("Probabilistic method")
  B = StepAm(A,42,5)
  C = StepB(B,A)
  result[i,2] <- norm(A-C$u%*%diag(C$d)%*%t(C$v))
  result[i,3] <- "without transpose"
  print(norm(A-C$u%*%diag(C$d)%*%t(C$v)))
  toc()
}

## When I do transpose on object matrix

for (i in 1:10) {
  nrow = i*100 ; ncol = 100
  result[i+10,1] = nrow
  A = t(randortho(nrow)[,1:100]%*%diag(sigma)%*%randortho(ncol))
  tic("Probabilistic method")
  B = StepAm(A,42,5)
  C = StepB(B,A)
  result[i+10,2] <- norm(A-C$u%*%diag(C$d)%*%t(C$v))
  result[i+10,3] <- "with transpose"
  print(norm(A-C$u%*%diag(C$d)%*%t(C$v)))
  toc()
}


P_errorbound(42,5,sigma,3,3)
P_errorbound(42,5,sigma,2,3)
P_errorbound(42,5,sigma,3,2)
E_errorbound(42,5,sigma)
```

## 4.4 Simulation: Comparisons for theoretical error bound

```r
set.seed(18)
m = 1000
n = 500
sigma = c(sort(abs(rnorm(150)),decreasing = T),rep(0,350))
A = randortho(1000)[,1:500]%*%diag(sigma)%*%randortho(500)

set.seed(18)
result = 1:1000
for (i in 1:1000) {
  B = StepAm(A,140,5)
  C = StepB(B,A)
  result[i] = norm(A-C$u%*%diag(C$d)%*%t(C$v))
}
mean(result)
max(result) #largest: 1.895834
min(result) #smallest: 0.866689
```

```
# estimated expectation error bound:  1.24092 with 1000 samples

E_errorbound(140,5,sigma)
# Theoretical Expectation error bound : 1.244751
P_errorbound(140,5,sigma,2,3)
# Theoretical Probabilistic error bound and failure probability:
   15.4196646  0.2912467
# actually this bound is not helpful

par(mfrow = c(1,2))
library("scatterplot3d") # load
ut <- cbind(expand.grid(1:5,1:5),matrix(nrow = 25,ncol = 1))
for (i in 1:nrow(ut)) {
  ut[i,3] <-  P_errorbound(140,5,sigma,ut[i,1],ut[i,2])[1]
}
scatterplot3d(ut,type = "h",color = "red",xlab = "u",ylab = "t",
              zlab = "Prob error bound",main = "prob error bound")

ut <- cbind(expand.grid(1:5,1:5),matrix(nrow = 25,ncol = 1))
for (i in 1:nrow(ut)) {
  ut[i,3] <-  P_errorbound(140,5,sigma,ut[i,1],ut[i,2])[2]
}
scatterplot3d(ut,type = "h",color = "blue",xlab = "u",ylab = "t",
              zlab = "maximum failure pb",main = "maximum failure pb")
# -> c(2,3)
```

## 4.5 Computing time of exact svd vs svd with randomness

```

####### Probabilistic method #######
tic("stepAm")
B = StepAm(A,140,5)
toc()
# stepAm: 0.24 sec elapsed

tic("StepB")
C = StepB(B,A)
norm(A-C$u%*%diag(C$d)%*%t(C$v))
toc()
# StepB: 0.18 sec elapsed
# norm difference: 1.137337

##  Approximation error without probabilistic method
tic("without probabilisit method")
C = svd(A)
norm(A-C$u[,1:145]%*%diag(C$d[1:145])%*%t(C$v[,1:145]))
toc()
# without probabilisit method: 1.01 sec elapsed
# norm difference: 0.2207723
```

## 4.6 Higher order tensor algorithm 1.

```r
## tensor_svd approx first method.
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)
}


tensor_resid = function(tnsr,App){
  a = normf(tnsr-ttm(ttm(ttm(App$Z,App$U[[1]],1),App$U[[2]],2),App$U
    [[3]],3))
  return(a)
}
```

## 4.7 Higher order tensor algorithm 2.

```r
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[-i]+p)*(rk[i]+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)
}
```

## 4.8 Simulation for tensor part 1

```r
## Simulation for model.
tnsr = rand_tensor(modes = c(100,100,100))
C = as.tensor(array(rep(0,1000000),dim = c(100,100,100)))
```

```r
A = hosvd(tnsr)
C[1:20,1:20,1:20] <-A$Z@data[1:20,1:20,1:20]
B = ttm(ttm(ttm(C,A$U[[1]],1),A$U[[2]],2),A$U[[3]],3)
normf(B)

## B is an object tensor
result = as.data.frame(matrix(nrow = 30,ncol = 3))
names(result) <- c("target_rk","error","method")
for(i in 1:15){
  result[i,1] <- i
  result[i,2] <- tensor_resid(B,tensor_svd(B,i,i,i,5))
  result[i,3] <- "1st method"
}
for(i in 1:15){
  result[i+15,1] <- i
  result[i+15,2] <- tensor_resid(B,tensor_svd2(B,i,i,i,5))
  result[i+15,3] <- "2nd method"
}

library(ggplot2)
ggplot(data = result, aes(x = target_rk,y = error,col = method))
+geom_point()+geom_line()+geom_hline(yintercept = normf(B),col = "red")
+geom_hline(yintercept = 0,col = "blue")
```

## 4.9 Simulation model 1

```r
################################
## Simulation for model.
tnsr = rand_tensor(modes = c(100,100,100))
C = as.tensor(array(rep(0,1000000),dim = c(100,100,100)))
A = hosvd(tnsr)
C[1:20,1:20,1:20] <-A$Z@data[1:20,1:20,1:20]
B = ttm(ttm(ttm(C,A$U[[1]],1),A$U[[2]],2),A$U[[3]],3)
normf(B)



#it has the same variance with operating matrix and noise-> hard to catch
    real data without noise.
#when sd = 1



sd = c(0.005,0.01,0.05,0.1,0.5,1)
result = data.frame(matrix(nrow = 6, ncol =3))
names(result)  <- c("sd","sd_hat","tensor_resid")
for (i in 1:6) {
  s=sd[i]
  result[i,1] = s
  e = as.tensor(array(rnorm(1000000,mean =0,sd = s),dim = c(100,100,100)))
  D = B+e
  est = tensor_svd(D,20,20,20,6)
  result[i,3] = tensor_resid(B,est)
  result[i,2] = sqrt(tensor_resid(B,est)^2/1000000)
}
result

# normf(D-B)^2/1000000 always get right variance.




ggplot(result, aes(x=sd)) +
  geom_point(aes(y=tensor_resid), col="blue") +
  geom_line(aes(y=tensor_resid), col="blue") +
  geom_point(aes(y=20*(abs(sd_hat-sd)/sd)),col="red") +
  scale_y_continuous("tensor residual", sec.axis = sec_axis(~(.)/20, name
   = "|sd-sd_hat|/sd")) +
  theme(
    axis.title.y.left=element_text(color="blue"),
    axis.text.y.left=element_text(color="blue"),
    axis.title.y.right=element_text(color="red"),
    axis.text.y.right=element_text(color="red")
  )
```

## References

[1] TG Kolda, BW Bade. *Tensor decompositions and applications*. SIAM Rev., 51(3), 455–500

[2] N Halko, PG Martinsson, JA Tropp *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions*. SIAM Rev., 53(2), 217–288.

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