Simulations of Tropps' paper and Tensor extension.

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

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

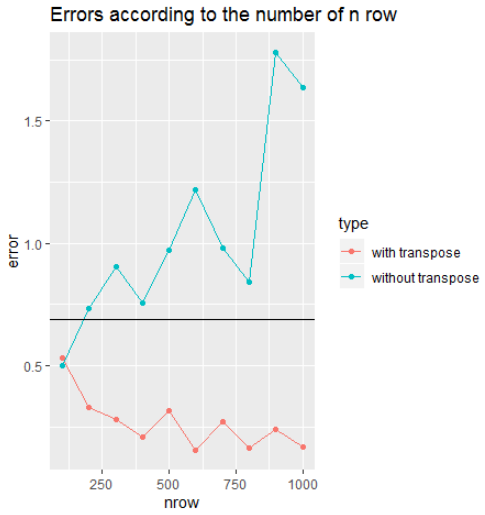

Figure 2: With default norm

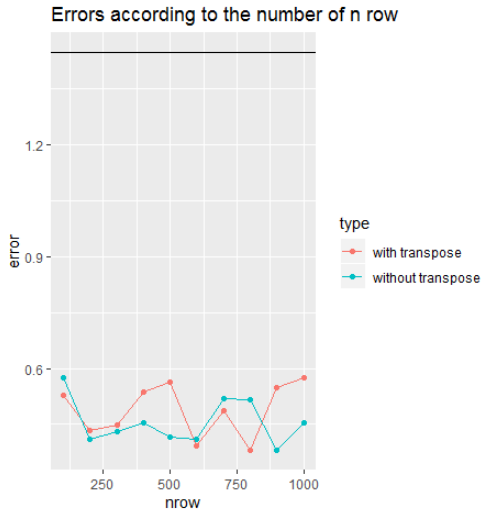

Figure 3: With Frobenius norm

I made mistake for this simulation. 'norm' function in R gives maximum column sum norm by default that's why I got much greater norms for skinny matrices than wide matrices. To make it clear, $\|(I - Q_1 Q_1^*)A\|_1$ becomes greater when the number of rows increase because this norm is computed by finding maximum absolute column sum. If I change the default norm into Forbenius norm I found there's no huge difference on errors between skinny and wide matrices.

3. Simulation procedure:

   (a) I made 1000×500 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.

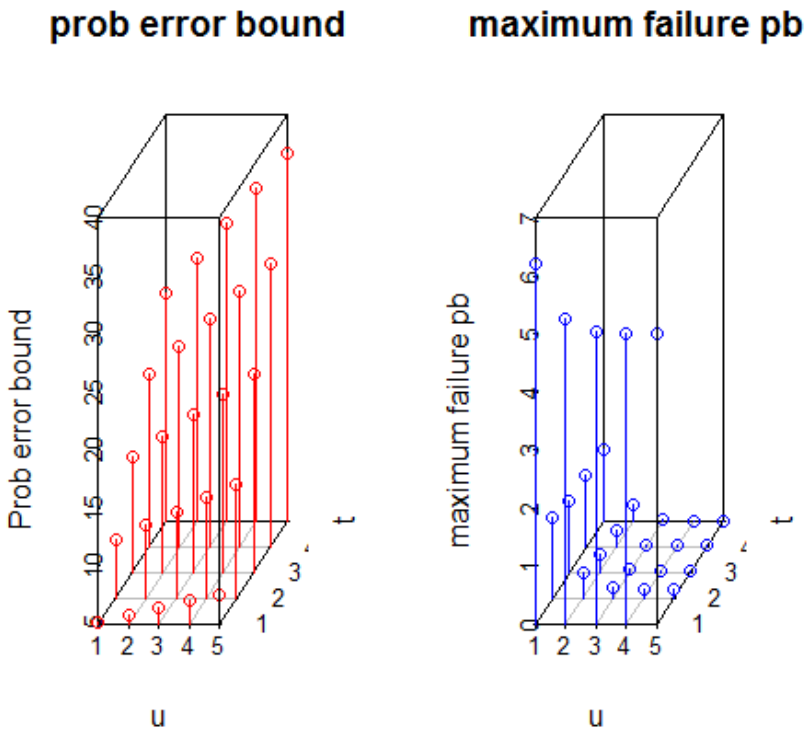

Figure 4: Prob error vs maximum failure rate according to u and t

4. For 1000×500 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

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

Algorithm 3 Approx tensor SVD 2

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

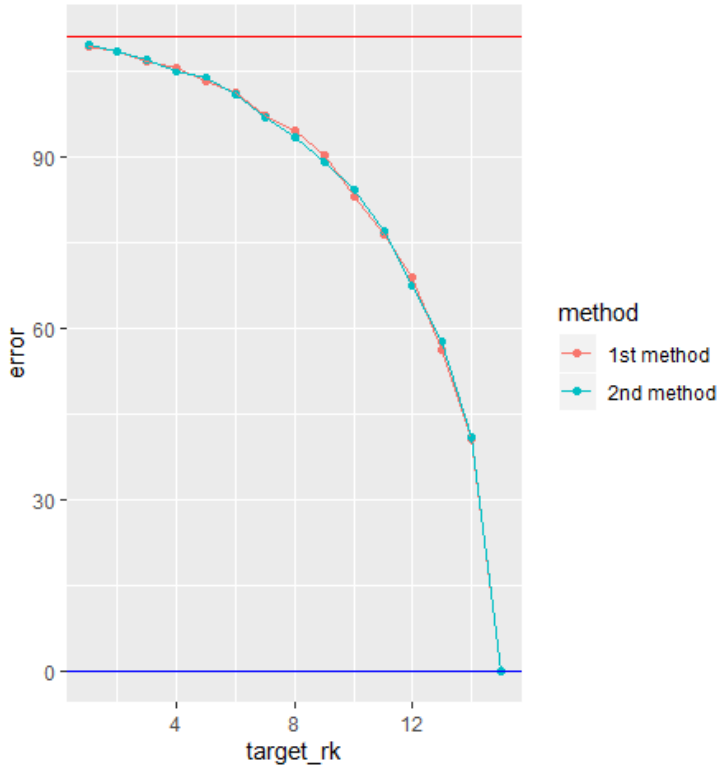

Figure 5: Approximation errors in tensor case

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. Comparison for 2 methods:
   To compare 2 methods, I made simulation in following order.
   (a) Make a $150 \times 150 \times 150$ tensor whose core tensor has $20 \times 20 \times 20$ non zero elements
   (b) Approximate tensor using 1st SVD algorithm and 2nd SVD algorithm.
   (c) Calculate norm difference from real tensor, time it took to approximate.
   (d) Repeat 30 times

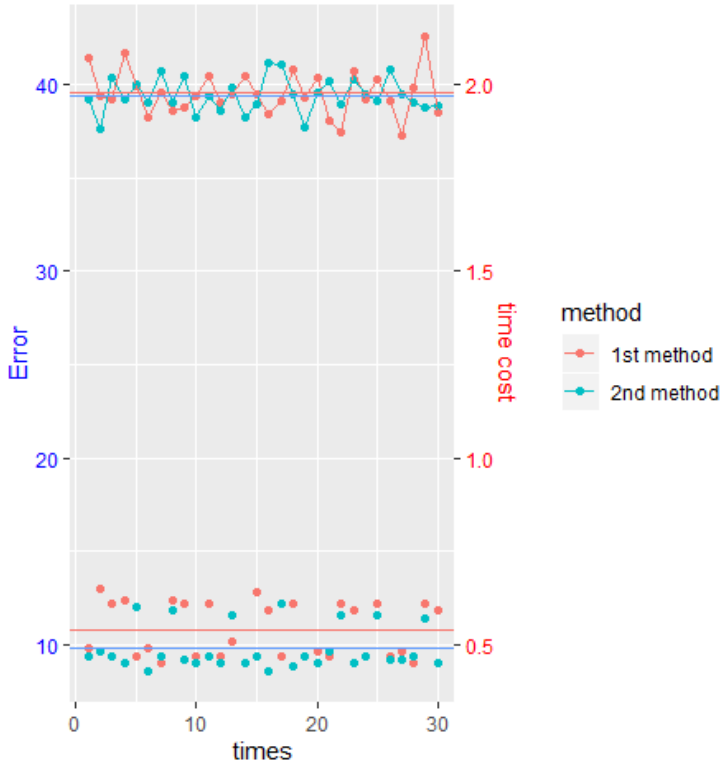

Figure 6: Approximation errors in tensor case

The result shows 2nd method is more efficient in computing cost than 1st method. This is because in Algorithm 2(First method), we need to generate $\Omega^{(1)}$ whose size is $I_1 \times I_2 I_3$,$\Omega^{(2)}$ whose size is $I_2 \times I_1 I_3$ and $\Omega^{(3)}$ whose size is $I_3 \times I_1 I_2$. Also because of huge random matrix size, heavy computations are needed to do matrix multiplication and QR decomposition.
On the other hand, in Algorithm 3 (Second method) we generate Ω_1, Ω_2 and Ω_3 whose sizes are $I_1 \times r_1, I_2 \times r_2$ and $I_3 \times r_3$ each. Even though we also need to generate $\Omega^{(1)}, \Omega^{(2)}$ and $\Omega^{(3)}$ to find out othonormal basis of image space, the number of elements in each matrix is only $r_1 r_2 r_3$. Therefore, even thought we generate more random matrix in Algorithm 3, the total number of random matrices' elements are less than in Algorithm 2. Also we don't need to compute multiplications of large matrices thanks to dimension reduction process in the line 5 at Algorithm 3.
With regards to accuracy, it seems that there is no huge difference. My conjecture for this result is if you check the theoretical error bound for matrix cases,

$$E\|(I - P_Y)A_F\| \leq (1 + \frac{k}{p-1})^{1/2}(\sum_{j>k} \sigma_j^2)^{1/2}$$

there's no terms related to sizes of random matrices we make. I think this result has something to do with the case of skinny and wide matrices in the section 1-2. Even though wide matrices need greater size of Gaussian matrices to capture range space,

there's no huge accuracy difference between fat and wide matrices. Therefore,I think matrix case phenomena can be applied to tensor case.

3. 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:

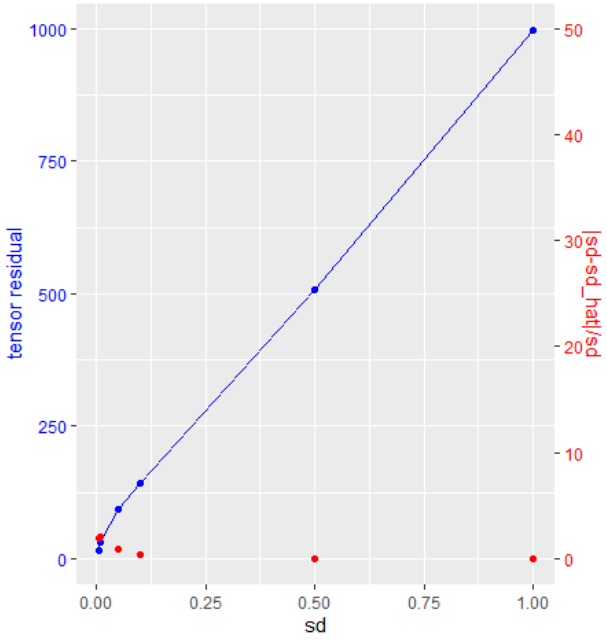

Figure 7: How good our approximation is with a noise

In Figure 7, 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.

Let's check residual analysis, I got residuals from above data, and I made model tensor data from drawing uniform distribution and made noise from drawing normal distribution. Since the noise follows normal distribution, residual histogram must follow normal distribution and you can check below Figure 8.

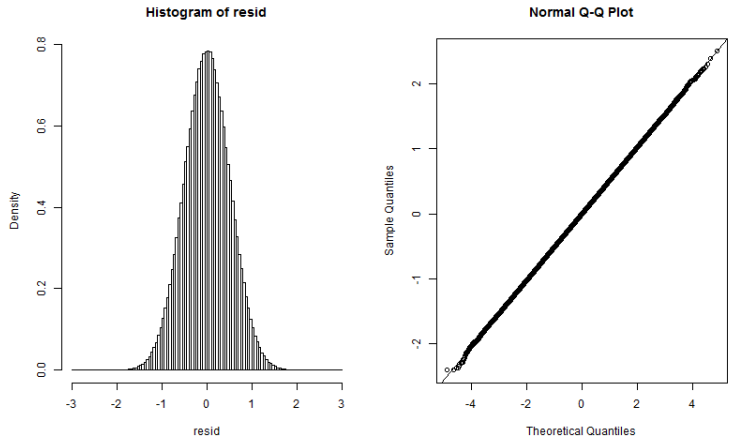

Figure 8: Residual histogram and QQ plot

4. How does oversampling work well on noises with different standard deviations.
While simulation to see relationship between oversampling size and accuracy, I found interesting phenomenon. If standard deviation of a noise is large enough, large oversampling size make approximation worse. My simulation procedure is as follows.

   (a) Make objective $100 \times 100 \times 100$ tensor having core tensor with $20 \times 20 \times 20$ non zero elements.

   (b) Make noise tensor drawing from normal distribution with mean 0 and different standard deviation.

   (c) Get given data by adding up two tensors.

   (d) Approximate tensor using 2nd SVD algorithm and calculate norm difference from objective tensor.

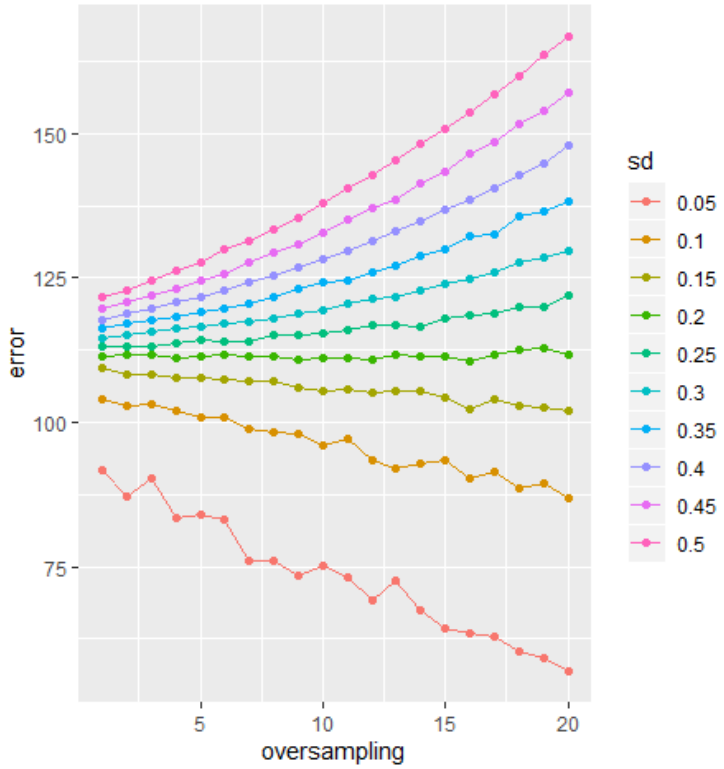

Figure 9: Oversampling sizes vs errors

Result: Above figure shows that if size of noise's standard deviation becomes greater than 0.2, oversampling size doesn't work well anymore. A reason for this phenomenon is the greater oversampling size gets, the more accurate approximation we can get for a given tensor with a noise. Therefore, error size of approximation become more and more close to Frobenious norm of a noise. However, If noise effect is small enough, we can get really good approximation for target tensor from given data having noise.
To sum up this phenomenon, let $\mathcal{A}$ be a tensor we want to estimate,$\mathcal{D} = \mathcal{A} + \mathcal{E}$ where $\mathcal{E}$ is our noise tensor and $\hat{\mathcal{A}}$ be our estimate. Then, as we increase oversampling size,

$$\|\mathcal{A} - \hat{\mathcal{A}}\| \to \|\mathcal{E}\| \quad \text{but} \quad \|\mathcal{D} - \hat{\mathcal{A}}\| \to 0$$

3 Modeling categorical data/ordinal data using tensor

1. For the category tensor $\mathcal{Y} = [y_{i_1,\cdots,i_N}] \in \{1, 2, \cdots, K\}^{d_1 \times \cdots \times d_N}$, I assume it's entries are realization of multinomial random variables. My model for those kinds of tensors is as follows.

$$P(y_{i_1,\cdots,i_N} = c) = [\mathbf{f}_\beta(\theta_{i_1,\cdots,i_N})]_c \quad c \in \{1, 2, \cdots, K\} \quad \beta \in \mathcal{R}^{K-1}$$

where $\mathbf{f}_\beta$ is a link function such that.

$$\mathbf{f}_\beta(x) = \left[\frac{e^{\beta_1^* x}}{\sum_{t=1}^K e^{\beta_t^* x}}, \cdots, \frac{e^{\beta_K^* x}}{\sum_{t=1}^K e^{\beta_t^* x}}\right]^T = \left[\frac{e^{\beta_1 x}}{1 + \sum_{t=1}^{K-1} e^{\beta_t x}}, \cdots, \frac{e^{\beta_{K-1} x}}{1 + \sum_{t=1}^K e^{\beta_t x}}, \frac{1}{1 + \sum_{t=1}^K e^{\beta_t x}}\right]^T$$

Also, we assume the parameter tensor $\Theta = \sum_{r=1}^{R} \lambda_r \mathbf{a_r}^{(1)} \otimes \cdots \otimes \mathbf{a_r}^{(N)}$ To sum up we have the following model.

$$\mathcal{Y} = \arg\max_c [\mathbf{f}_\beta(\Theta + \mathcal{E})]_c$$

Where $\mathcal{E}$ is a noise tensor which follows logistic distribution.

2. For the ordinary tensor $Y = [y_{i_1,\cdots,i_N}] \in \{1, 2, \cdots, K\}^{d_1 \times \cdots \times d_N}$ we will use cumulative logit model. Our model is as follows.

$$P(y_{i_1,\cdots,i_N} \leq j|x) = \pi_1(x) + \cdots + \pi_j(x)$$

$$logit(P(y_{i_1,\cdots,i_N} \leq j|x)) = \log \frac{P(y_{i_1,\cdots,i_N} < j|x)}{1 - P(y_{i_1,\cdots,i_N} \leq j|x)} = \log \frac{\pi_1(x) + \cdots + \pi_j(x)}{\pi_{j+1} + \cdots + \pi_K(x)} = \alpha_j + \beta x$$

where α_j is non-decreasing with regards to j

$$P(y_{i_1,\cdots,i_N} = j) = [\mathbf{f}_{\alpha\beta}(\theta_{i_1,\cdots,i_N})]_j \quad j \in \{1, 2, \cdots, K\} \quad \alpha \in \mathcal{R}^K \quad \beta \in \mathcal{R}$$

where

$$\mathbf{f}_{\alpha,\beta}(x) = \left[\frac{e^{\alpha_2+\beta x}}{1 + e^{\alpha_2+\beta x}} - \frac{e^{\alpha_1+\beta x}}{1 + e^{\alpha_1+\beta x}}, \cdots, \frac{e^{\alpha_K+\beta x}}{1 + e^{\alpha_K+\beta x}} - \frac{e^{\alpha_{K-1}+\beta x}}{1 + e^{\alpha_{K-1}+\beta x}}, \right]^T$$

Like in the category tensor case, we assume the parameter tensor $\Theta = \sum_{r=1}^{R} \lambda_r \mathbf{a_r}^{(1)} \otimes \cdots \otimes \mathbf{a_r}^{(N)}$ Finally, we have the following model.

$$\mathcal{Y} = \arg\max_j [\mathbf{f}_{\alpha,\beta}(\Theta + \mathcal{E})]_j$$

Where $\mathcal{E}$ is a noise tensor which follows logistic distribution.

4 Miscellaneous

Theorem 1. *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* $\|M_i - \hat{M}_i\| \leq \epsilon$ *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 $\|\hat{A} - \mathcal{A}\|$ *?*

Proof. First, notice that for each i,

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

Main proof is as follows

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

□

To get better error bound let's define principal angles.

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*

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

Then, our new error bound becomes as in following Theorem 2.

Theorem 2. *Under the same condition in Theorem 1 but* $\sin(\theta(span(M_i), span(\hat{M}_i^t))) < \epsilon$ *with matrix norm,*

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

Proof. It suffices to show $\|M_i M_i^t - \hat{M}_i \hat{M}_i^t\| \leq 2\epsilon$ because we can apply this last inequality in the proof of Theorem 1.

$$\|\mathcal{A} - \hat{A}\| \leq (\|M_1 M_1^t - \hat{M}_1 \hat{M}_1{}^t\| + \|M_2 M_2^t - \hat{M}_2 \hat{M}_2{}^t\| + \|M_3 M_3^t - \hat{M}_3 \hat{M}_3{}^t\|)\|\mathcal{A}\| \leq 6\epsilon\|\mathcal{A}\|$$

Then we are done.
Proof of the above inequality is as follows.

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

Last inequality follows from combining following 2 lemmas. □

Lemma 1. *If $P_R$and P_N are the orthogonal projectors onto $R$and N , respectively, then*

$$\cos\theta = \|P_N P_R\| = \|P_R P_N\|.$$

Proof. For vectors x and y such that $\|x\| = \|y\| = 1$, we have $P_R x \in \mathcal{R}$ and $P_N y \in \mathcal{N}$ Then

$$\cos\theta = \max_{u\in\mathcal{R}, v\in\mathcal{N}\|u\|=\|v\|=1} v^t u = \max_{u\in\mathcal{R}, v\in\mathcal{N}, \|u\|\leq 1\|v\|\leq 1} v^t u = \max_{\|x\|\leq 1\|y\|\leq 1} y^t P_{\mathcal{N}} P_{\mathcal{R}} x = \|P_R P_N\|$$

.

□

Lemma 2. *Under the same condition on Lemma 1,*

$$\|P_N(I - P_R)\| \leq \sin(\theta)$$

Proof.

$$\begin{aligned}
\|P_N(I - P_R)\|^2 &= \max_{u\in\mathcal{R}, \|u\|=1} u^t P_N (I - P_R) u = \max_{u\in\mathcal{R}, \|u\|=1} u^t P_N u - u^t P_N P_R u \\
&\leq 1 - \|P_N P_R\|^2 = \sin^2(\theta)
\end{aligned}$$

□

4.1 Question.

1. What is a noise tensor's role? In a paper *Learning from Binary Multiway Data: Probabilistic Tensor Decomposition and its Statistical Optimality*,a noise tensor was put inside of link function. What is a reason for that? My thought is that helps us to find out MLE so we can analyze it's performance better. Am I got right?

2. CP decomposition reduces overall parameters to estimate. However, it seems to me that Tucker decomposition doesn't make any advantage in respect of reducing parameter size.

4.2 To do list

1. I think my method for category tensor or ordinary tensor is simple and classical. So I want to find another way to analyze those kinds of data for this weekend and add to this report.

2. Also, I will think about how to estimate parameters well in my method.

3. I have read the paper *Learning from Binary Multiway Data: Probabilistic Tensor Decomposition and its Statistical Optimality* I will wrap up this paper this weekend.

4. Also, I will wrap up the paper *Tensor on tensor regression*

5 Appendix: My R codes

5.1 Codes for SVD with randomness

```
1
2  normf = function(y){
3    return(as.numeric(sqrt(sum(y^2))))
4  }
5
6  inner = function(x,y){
7    return(as.numeric(t(x)%*%y))
8  }
9
10 E_errorbound = function(k,p,sigma){
11   return((1+k/(p-1))^{1/2}*(sqrt(sum(sigma[(k+1):length(sigma)]^2))))
12 }
13
14 P_errorbound = function(k,p,sigma,u,t){
15   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)
16   fp = 5*t^-p +2*exp(-u^2/2)
17   return(c(errbd,fp))
18 }
19
20
21 StepAm = function(A,r,p){
22   m = nrow(A); n = ncol(A)
23   q = 3
24   l = r+p
25   omega = matrix(rnorm(n*l),nrow = n, ncol = l)
26   Y = A%*%omega
27   Q = qr.Q(qr(Y))
28   return(Q)
29 }
30
31 StepB = function(Q,A){
32   B = svd(t(Q)%*%A)
33   B$u = Q%*%B$u
```

```
  return(B)
}
```

5.2 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")
  )
```

5.3 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
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] <- normf(A-C$u%*%diag(C$d)%*%t(C$v))
```

```
  result[i,3] <- "without transpose"
  print(normf(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] <- normf(A-C$u%*%diag(C$d)%*%t(C$v))
  result[i+10,3] <- "with transpose"
  print(normf(A-C$u%*%diag(C$d)%*%t(C$v)))
  toc()
}

library(ggplot2)
ggplot(data = result, aes(x = nrow,y = error,colour = type))+
  geom_point(aes(y = error))+geom_line(aes(y = error))+
  geom_hline(yintercept  =E_errorbound(42,5,sigma))+
  ggtitle("Errors according to the number of n row")
```

5.4 Simulation: Comparisons for theoretical error bound

```
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)
```

5.5 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
```

5.6 Higher order tensor algorithm 1.

```
## 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)
}
```

5.7 Higher order tensor algorithm 2.

```
tensor_svd2 = function(tnsr,k1,k2,k3,p){
  App = list(Z=NULL,U=NULL)
  rk = c(k1,k2,k3)
  a = c(1,2,3,1,2,3)
  Omega = list()
  Q = list()
  for (i in 1:3) {

    Omega[[i]] <- matrix(rnorm(tnsr@modes[i]*(rk[i]+p)),ncol = rk[i]+p)
  }
  for (i in 1:3) {
    ing <- matrix(rnorm(prod(rk[-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)
}
```

5.8 Simulation for tensor part 1

```
## 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)
```

```
## 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")
```

5.9 Comparing 2 different methods

```
## comapring accuracy and efficiency

result = as.data.frame(matrix(nrow = 60,ncol = 3))
names(result) <- c("times","error","method")

for(i in 1:30){
  tic()
  result[i,1] <- i
  result[i,2] <- tensor_resid(B,tensor_svd(B,4,4,4,4))
  result[i,3] <- "1st method"
  toc(log = T, quiet = T)
}
log.lst <- tic.log(format = F)
timings1 <- unlist(lapply(log.lst, function(x) x$toc - x$tic))
tic.clearlog()
for(i in 1:30){
  tic()
  result[i+30,1] <- i
  result[i+30,2] <- tensor_resid(B,tensor_svd2(B,4,4,4,4))
  result[i+30,3] <- "2nd method"
  toc(log = T,quiet = T)
}
log.lst <- tic.log(format = F)
timings2 <- unlist(lapply(log.lst, function(x) x$toc - x$tic))
tic.clearlog()

result <- cbind(result,ct)
result[,3] <- as.factor(result[,3])
ggplot(result, aes(x=times,col = method)) +
  geom_point(aes(y=error)) +
```

```
  geom_line(aes(y=error)) +
  geom_point(aes(y=20*(ct))) +
  scale_y_continuous("Error", sec.axis = sec_axis(~(.)/20, name = "time
    cost")) +
  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")
  )+geom_hline(yintercept =20* mean(ct[1:30]),color='#F8766D')+
  geom_hline(yintercept = 20*mean(ct[31:60]),color ="#619CFF")+
  geom_hline(yintercept = mean(result[1:30,2]),color='#F8766D')+
  geom_hline(yintercept = mean(result[31:60,2]),color ="#619CFF")
```

5.10 Simulation model 1

```
################################
## 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")
  )
```

5.11 Oversampling effect according to different noise's standard deviations

```

## Oversampling simulation: does it really good?##############
## Tensor construction.
tnsr = rand_tensor(runif(1000000),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)
e = as.tensor(array(rnorm(1000000,mean =0,sd = 0.1),dim = c(100,100,100)))
D = B+e


ovs <- as.data.frame(matrix(nrow = 200,ncol =3))
names(ovs) <- c("oversampling","error","sd")
for (j in 1:10) {
  e = as.tensor(array(rnorm(1000000,mean =0,sd = 0.05*j),dim = c
   (100,100,100)))
  D = B+e
  for (i in 1:20) {
    ovs[i+20*(j-1),1] <- i
    ovs[i+20*(j-1),2] <- tensor_resid(B,tensor_svd2(D,20,20,20,i))
    ovs[i+20*(j-1),3] <- 0.05*j
  }
}
ovs[,3] <- as.factor(ovs[,3])
ggplot(data = ovs,aes(x = oversampling,y=error,color = sd))+
  geom_point()+geom_line()
```

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.