Solving 3-D Puzzles Using Tensor Decomposition and Application to Education of Multidimensional Data Analysis

Akio Ishida, Naoki Yamamoto, Jun Murakami, and Nobuhiro Oishi

***Abstract*—We have been conducting research on multi-dimensional data analysis. Because it is difficult to teach students of our laboratory the concept of multidimensional data processing, we are developing teaching materials to make them easier to understand that. In this paper, we described a method to solve 3-d puzzles using HOSVD, which is one of the methods of tensor decomposition, and proposed utilizing it to education. Specifically, we took up several kinds of 3-d puzzles and showed their solutions and scripts of R language.**

***Index Terms*—Multidimensional data processing, tensor decomposition, HOSVD, 3-d puzzles, understanding support.**

I. Introduction

We have been worked to develop a precise and rapid decomposition method of multidimensional array [1] and to analyze multidimensional data, such as clinical data provided from a rehabilitation hospital, by using higher-order singular value decomposition (HOSVD) [2] and other decomposition methods for many years [3], [4]. Recently, it recognized in Japan that data analysis is a skill which needs to all students to learn [5]. Under this situation, we have started to teach our students multidimensional data analysis by using R language [6] in addition to basics of statistics and multivariate analysis. However, since the concept of multidimensional data and operations between the data are fairly complicated and difficult, it takes considerable time for students to understand. Moreover, because there are few reference books and teaching materials of those contents written in Japanese, we are instructing students earlier to perform calculations such as tensor decomposition of HOSVD using R language libraries. Therefore, we have created and proposed a system that displays the process of decomposition of HOSVD by CG and makes it easier for students to understand [7].

This time we got inspiration from the fact that 3-d puzzle creation, such as cube Sudoku, becomes popular [8] and then devised a method to teach multidimensional data processing to students by solving the puzzle using HOSVD. By using this method for the education, it is thought that students can easily understand the concepts and calculations of multidimensional data.

Manuscript received August 10, 2018; revised September 20, 2018. This work was supported by the JSPS KAKENHI (Grants-in Aid for Scientific Research) Grant Number JP18K11596.

A. Ishida is with the Faculty of Liberal Studies, Kumamoto College, National Institute of Technology, Koshi, 861-1102 Japan (e-mail: ishida@kumamoto-nct.ac.jp).

N. Yamamoto and J. Murakami is with the Department of Human-Oriented Information Systems Engineering, Kumamoto College, National Institute of Technology, Koshi, 861-1102 Japan.

N. Oishi is with the Department of Information, Communication and Electronic Engineering, Kumamoto College, National Institute of Technology, Koshi, 861-1102 Japan.

II. HOSVD

A. 3-D Puzzle and Its Tensor Representation

Firstly, we describe tensor representation of a 3-d puzzle. As shown in Fig. 1, the 3-d puzzle is a rectangular solid consisted of pilling up $I_1 \times I_2 \times I_3$ cubes with a size of $1\times1\times1$. An example of a question for the puzzle is as follows [9]:

[Question 1 (Drill Hole Puzzle)]

In Fig. 1, digits on the visible faces, that is, front, top, and right face, of the puzzle denote the number of the $1\times1\times1$ cubes with holes drilled into the puzzle from the faces. Note that the holes are not drilled from opposite (invisible) faces of the puzzle. Find the number of undrilled $1\times1\times1$ cubes in the puzzle.

(End of Question)

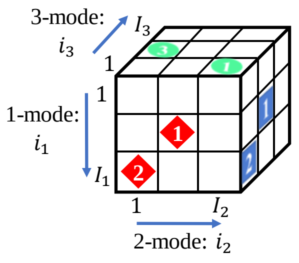


Fig. 1. Example of 3-d puzzle.

In this paper, to solve the questions of the above kind, we represent the 3-d puzzles as the higher-order tensors that means the multi-dimensional (more than or equal to three dimensions) arrays, where from the first- to the third-order tensors correspond to vectors, matrices, and 3-d arrays, respectively.

Now, we explain the notation of third-order tensor by using Fig. 1. The puzzle of this figure can be expressed as a third-order tensor whose size is $I_1 \times I_2 \times I_3 = 3\times3\times3$. Each arrow illustrated in the figure shows so-called a mode, that is, from the 1-mode to the 3-mode represent the vertical, horizontal, and depth direction, respectively. Let $\mathcal{A}$ be this third-order tensor, then it can be represented as $\mathcal{A} = (a_{i_1 i_2 i_3})$, $(i_1 = 1, \cdots, I_1; i_2 = 1, \cdots, I_2; i_3 = 1, \cdots, I_3)$ by using indices i_n

related to the *n*-modes $(n = 1, 2, 3)$, where $a_{i_1 i_2 i_3}$ denotes an (i_1, i_2, i_3) th element of the tensor. Since each element $a_{i_1 i_2 i_3}$ is an $1\times1\times1$ cube, in order to count the drilled cubes, the value given to each element is decided as follows: $a_{i_1 i_2 i_3} = 0$ (for drilled cube) or $a_{i_1 i_2 i_3} = 1$ (otherwise).

For convenience of explanation, the front face with red diamonds on it is referred to as the 1-2 face in Fig. 1. Similarly, the top and right faces are expressed as the 2-3 face and the 1-3 face, respectively. For these faces, the 1-2 face can be correctly expressed by the matrix $(a_{i_1 i_2 i_3})$, $(i_1 = 1, \cdots, I_1;$ $i_2 = 1, \cdots, I_2;$ i_3 is fixed to 1) and the same expression for the others.

B. HOSVD and Its Algorithm

Higher-order singular value decomposition (HOSVD) is an extension of the matrix (second-order tensor) SVD to the higher-order tensor. This decomposition method is widely applied in the fields of signal processing, image processing, pattern recognition, data analysis, etc. [e.g., 10, 11]. This time, we thought that the above-mentioned 3-d puzzles can be solved by applying the *n*-mode matrix unfolding, which is an important part of the procedure for computing the HOSVD, and that this calculation process is effective for teaching students the principle of that decomposition. Now, we describe an algorithm for the HOSVD, but the details of the *n*-mode matrix unfolding will be done in the next subsection. Please note that here, since the 3-d puzzles are expressed by third-order tensors, we only describe the case of those tensors in this paper.

HOSVD of a third-order tensor $\mathcal{A}$ is a method that decomposes $\mathcal{A}$ into $\mathcal{A} = \mathcal{B} \times_1 U^{(1)} \times_2 U^{(2)} \times_3 U^{(3)}$, where $U^{(1)}$, $U^{(2)}$, and $U^{(3)}$ are orthonormal matrices, and $\mathcal{B}$ is a core tensor, which can be briefly thought as the characteristic tensor of $\mathcal{A}$ with similar size (possible to reduce, if needed). The operator $\times_n$ is called an *n*-mode product that multiplies a tensor and a matrix. Although the concept of this product is also important in the HOSVD algorithm, this paper omits the explanation of it, due to limitations of space (please see Ref. [2]). HOSVD algorithm is as follows:

[Algorithm 1 (HOSVD)]

Input: Third-order tensor $\mathcal{A}$ with a size of $I_1 \times I_2 \times I_3$.

Output: Orthonormal matrices $U^{(1)}$, $U^{(2)}$, and $U^{(3)}$ and core tensor $\mathcal{B}$.

(Step 1) Apply the *n*-mode matrix unfolding to the tensor $\mathcal{A}$ and form matrices $A_{(n)}$, $(n = 1, 2, 3)$.

(Step 2) Apply SVD to $A_{(n)}$ obtained in the Step 1, and decompose them into $A_{(n)} = U^{(n)} \Sigma^{(n)} V^{(n)}$, $(n = 1, 2, 3)$, where $U^{(n)}$ and $V^{(n)}$ is the left and right singular matrices, respectively, and $\Sigma^{(n)}$ is a diagonal matrix with singular values in diagonal.

(Step 3) Compute the core tensor $\mathcal{B}$ from the *n*-mode product of the input tensor $\mathcal{A}$ and the matrices $U^{(n)}$ obtained in the Step 2 as $\mathcal{B} = \mathcal{A} \times_1 U^{(1)^T} \times_2 U^{(2)^T} \times_3 U^{(3)^T}$.

(Step 4) Return the matrices $U^{(n)}$, $(n = 1, 2, 3)$ and the core tensor $\mathcal{B}$ which are obtained in the Step 2 and Step 3, respectively, to the caller.

(End of Algorithm)

C. n-Mode Matrix Unfolding and Its Algorithm

The *n*-mode matrix unfolding is an operation to convert a tensor into *n* matrices. The value *n* in the *n*-mode indicates how to unfold a higher-order tensor to a matrix. For example, using Fig. 1, the 1-mode unfolding matrix is obtained by cutting the cube along the horizontal (2-mode) direction with the width of $1\times1\times1$ cube and arranging them side by side next to the left face matrix in the depth direction. Therefore, the original tensor is transferred into the matrix without changing the size and subscript order of row. The 2-mode and 3-mode matrix unfolding transfer the tensor into the matrix, similarly.

Although there are several computation algorithms for the matrix unfolding, we use that of Ref. [2] in consideration of ease of implementation. The algorithm is as follows:

[Algorithm 2 (n-mode matrix unfolding)]

Input: Third-order tensor $\mathcal{A}$ with the size of $I_1 \times I_2 \times I_3$.

Output: *n*-mode matrix unfolding $A_{(n)}$, $(n = 1, 2, 3)$.

(Step 1) 1-mode matrix unfolding

Extract submatrices $A_{i_2} = (a_{* i_2 *})$, $(i_2 = 1, \cdots, I_2)$ from the tensor $\mathcal{A}$, then arrange them side by side as follows: $A_{(1)} = (A_1 \mid A_2 \mid \cdots \mid A_{I_2})$, where $(a_{* i_2 *})$ is an element of the matrix whose first subscript i_1 varies from 1 to I_1 and third one i_3 does 1 to I_3 with a fixed i_2. In the following steps, unfolding matrices are constructed in a manner similar to this step.

(Step 2) 2-mode matrix unfolding

Extract submatrices $A_{i_3} = (a_{** i_3})$, $(i_3 = 1, \cdots, I_3)$ from $\mathcal{A}$ and transpose each of them. And then, arrange obtained matrices side by side as follows: $A_{(2)} = (A_1^T \mid A_2^T \mid \cdots \mid A_{I_3}^T)$.

(Step 3) 3-mode matrix unfolding

Extract submatrices $A_{i_1} = (a_{i_1 **})$, $(i_1 = 1, \cdots, I_1)$ from $\mathcal{A}$ and transpose each matrix like the previous step. And then, arrange obtained matrices as $A_{(3)} = (A_1^T \mid A_2^T \mid \cdots \mid A_{I_1}^T)$.

(Step 4) Return the matrices $A_{(n)}$, $(n = 1, 2, 3)$ obtained from above steps to the caller.

(End of Algorithm)

Fig. 2 shows an image applying the 1-mode matrix unfolding to the 3-d puzzle of Fig. 1. Other mode matrix unfoldings can be thought of as like 1-mode case. By unfolding the 3-d puzzle to the 2-d matrix in this way, it is considered that the position of drilled hole distributed three-dimensionally becomes easier to find.

III. Solution of 3-D Puzzle

A. Solution Method of 3-D Puzzles

Generally, the 3-d puzzle of Question 1 in Section II-A is solved by using the brute-force method, but we use here the HOSVD. Two solution methods by this decomposition technique, where both methods use the *n*-mode matrix unfolding, are described below.

[Solution method 1]

(Step 1) Construct the third-order tensor $\mathcal{A}$ with binary values representing the presence or absence of holes drilled vertically from three faces in the 3-d puzzle.

(Step 2) Apply the 1-mode matrix unfolding to $\mathcal{A}$ and compute the unfolding matrix $A_{(1)}$.

(Step 3) The answer of the puzzle can be obtained by counting the number of elements of $A_{(1)}$ with the values representing the absence of a hole.

(End of Solution)

[Solution method 2]

(Step 1) Construct the third-order tensor $\mathcal{B}$ with binary values, similar to the previous solution method, but the holes are drilled only from the 1-2 face. Similarly, construct the third-order tensors $\mathcal{C}$ and $\mathcal{D}$ which are drilled from the 2-3 face and the 1-3 face, respectively.

(Step 2) Compute the matrix $B_{(1)}$, $C_{(1)}$, and $D_{(1)}$ by applying the 1-mode matrix unfolding to the third-order tensors $\mathcal{B}$, $\mathcal{C}$, and $\mathcal{D}$, respectively.

(Step 3) The answer of the puzzle can be obtained by counting the number of elements with the values representing the absence of a hole from the logical AND operation of the elements at the same position in the matrices $B_{(1)}$, $C_{(1)}$, and $D_{(1)}$.

(End of Solution)

The Solution method 1 seems to be easy to understand at a glance, because it simply counts the number of the empty (undrilled) elements with white color in the matrix $A_{(1)}$ illustrated in the right side of Fig. 2. In comparison with above method, the Solution method 2 requires slightly more time to count the empty elements, since this method needs to repeat the *n*-mode unfolding algorithm three times to create the matrix $B_{(1)}$, $C_{(1)}$, and $D_{(1)}$. Furthermore, it is necessary to take logical AND operation between three corresponding elements of those matrices.

In the next section, we implement the above two solution methods and show the results of applying them to the 3-d puzzles of Question 1.

B. *Example of Solution 1 for Question 1*

In this section, we describe the solutions to Question 1 in Section II-A. First, we solve this question using the Solution method 1.

1) *Example of solution 1-1 (using solution method 1)*

From now on we install the rTensor package [12] for tensor calculations and use the unfold function in the package for matrix unfolding. This function can be used as follows.

```
unfold( tnsr, row_idx, col_idx )
```

In the calling statement, the original higher-order tensor is given to the parameter tnsr, and the mode numbers which indicate the row and column directions of the unfolding matrix are given to the parameter *row_idx* and *col_idx*, respectively.

For example, when we compute the 1-mode matrix unfolding, the parameter values are given as *row_idx*=1 and *col_idx*=(2 and 3), because the 1-modes of the original tensor (*tnsr*) is transferred to as the row, and the 2- and 3-mode as the column as shown in Fig. 2. Note that in the latter case, these two modes are transferred to as a vector with the size of the product of the sizes of the 2- and 3-mode of the original tensor, and consequently the parameter *col_idx* is given as c(3,2) by using the function c in R. As for the reason why the order of *col_idx* (c(3,2)) descends, the subscript i_3 needs to be varied before i_2 in $A_{(1)}$ (please see the right side of Fig. 2).

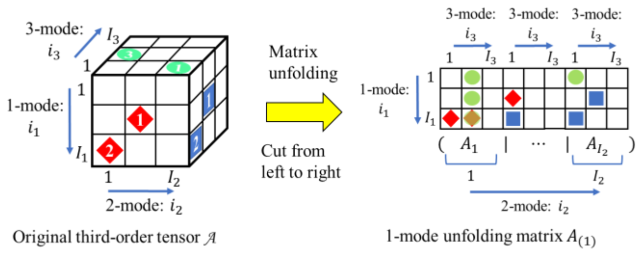


Fig. 2. Example of 1-mode matrix unfolding.

The R script for solving this example is shown below.

[R script for Example of solution 1-1]

```
# To solve Question 1 using Solution method 1
library(rTensor)  # install rTensor (only at the first time)
I1 <- 3; I2 <- 3; I3 <- 3  # set the size of 3rd order tensor
A <- array( 1, dim = c(I1,I2,I3) ) # initialize the tensor A
A <- as.tensor( A )   # convert A to tensor form
# construct 3rd order tensor of 3-d puzzle in Fig. 1
A[ 2,2,1 ] <- 0    # drill a hole from A(2,2,1) with length 1
A[ 3,1,1:2 ] <- 0  #          〃          A(3,1,1)        〃        2
A[ 1:3,1,2 ] <- 0  #          〃          A(1,1,2)        〃        3
A[ 1,3,1 ] <- 0    #          〃          A(1,2,1)        〃        1
A[ 2,3,2 ] <- 0    #          〃          A(2,3,2)        〃        1
A[ 3,2:3,1 ] <- 0  #          〃          A(3,2,1)        〃        2
# 1-mode matrix unfolding
A_1mode <- unfold( A, row_idx=1, col_idx=c(3,2) )
# count the number of elements without holes (empty ones)
ans1_1 <- sum( A_1mode @data )
```

(End of Script)

Table I shows the result of computing the 1-mode matrix unfolding (*A_1mode*) using this script. In this table, “1” and “0” represent the undrilled and drilled cubes, respectively. Besides, in this table, the cells with value “0” are color-coded according to the direction in which the holes are drilled. These colors are same as those of Fig. 2, except that the yellow color is used when two or three holes overlap. Comparing the table with $A_{(1)}$ in the right side of Fig. 2 for the location of the colors, we see that the puzzle data was correctly given to the obtained unfolding matrix *A_1mode*.

From the above the number of cubes without holes (undrilled) can be obtained in *ans1_1* by counting the elements with value “1” in the matrix *A_1mode*. The answer is as follows:

```
> ans1_1   # display the number of cubes without holes
[1] 18
```

Consequently, we can see that the number of undrilled cubes is eighteen and that the correct answer is obtained.

TABLE I: RESULT OF 1-MODE MATRIX UNFOLDING

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]	[,8]	[,9]
[1,]	**1**	**0**	**1**	**1**	**1**	**1**	**0**	**1**	**1**
[2,]	**1**	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**1**
[3,]	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**1**

2) *Example of solution 1-2 (using solution method 2)*

Next, the same example above is solved using the Solution method 2 described in section III-A. The R script for this question is shown below.

[R script for Example of solution 1-2]

```
# To solve Question 1 using Solution method 2
I1 <- 3; I2 <- 3; I3 <- 3
B <- array( 1, c( I1, I2, I3 ) )  # initialize the tensor B
C <- array( 1, c( I1, I2, I3 ) )  # initialize the tensor C
D <- array( 1, c( I1, I2, I3 ) )  # initialize the tensor D
# convert B, C, and D to tensor form
B <- as.tensor( B ); C <- as.tensor( C ); D <- as.tensor( D )
# give data in which holes are drilled from the 1-2, 2-3, and
# 1-3 faces to matrices B, C, D, respectively.
B[ 2,2,1 ] <- 0; B[ 3,1,1:2 ] <- 0   # from 1-2 face
C[ 1:3,1,2 ] <- 0; C[ 1,3,1 ] <- 0   # from 2-3 face
D[ 2,3,2 ] <- 0; D[ 3,2:3,1 ] <- 0   # from 1-3 face
# 1-mode matrix unfolding of  3rd order tensor B, C,  and D
B_1mode <- unfold( B, 1, c(3,2) )   # unfold B
C_1mode <- unfold( C, 1, c(3,2) )   # unfold C
D_1mode <- unfold( D, 1, c(3,2) )   # unfold D
# count the number of elements without holes (empty ones)
Temp<- B_1mode@data*C_1mode@data
                          *D_1mode@data
ans1_2 <- sum( temp )
```

(End of Script)

By executing this script, the data when the holes are drilled from each face of the original tensor is given to the 1-mode unfolding matrices *B_1mode*, *C_1mode*, and *D_1mode*, respectively. The 1-mode unfolding matrix *B_1mode* of the third-order tensor *B* is shown in Table II. From this table, it can be seen that the *B_1mode* correctly has the information about the red diamond marks on the 1-2 face of the tensor $\mathcal{A}$ in the left side of Fig. 2.

Similarly, the matrices *C_1mode* and *D_1mode*, which are the results of the 1-mode matrix unfolding of the tensors *C* and *D*, are shown in Table III and IV, respectively.

TABLE II: 1-MODE MATRIX UNFOLDING OF 3RD ORDER TENSOR B

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]	[,8]	[,9]
[1,]	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
[2,]	**1**	**1**	**1**	**0**	**1**	**1**	**1**	**1**	**1**
[3,]	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**

TABLE III: 1-MODE MATRIX UNFOLDING OF 3RD ORDER TENSOR C

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]	[,8]	[,9]
[1,]	**1**	**0**	**1**	**1**	**1**	**1**	**0**	**1**	**1**
[2,]	**1**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
[3,]	**1**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**

TABLE IV: 1-MODE MATRIX UNFOLDING OF 3RD ORDER TENSOR D

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]	[,8]	[,9]
[1,]	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
[2,]	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**1**
[3,]	**1**	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**1**

The number of the elements without holes (undrilled) in any of the unfolding matrices in Table II to IV is substituted into the variable temp in this script. Since this result is the same as that of the example of Solution 1-1 (shown in Table I), it is considered that the correct answer is obtained.

The Solution method 2 using three unfolding matrices requires more storage area than the Solution method 1 which uses only one unfolding matrix. On the other hand, it also has the following advantages. For example, looking at the (yellow-colored) element (3,2) in the Table I, the holes in the Table II and III also overlap there, so that we can see the direction in which the holes were drilled in the cube.

C. *Example of Solution 2 for Question 2*

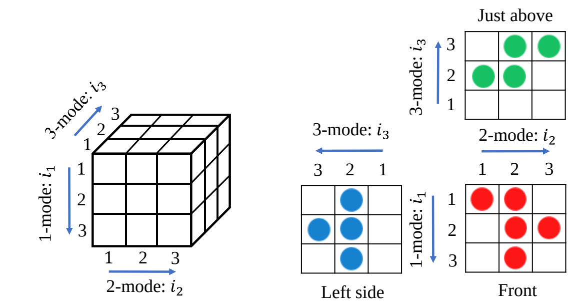
In this example, we deal with another type of 3-d puzzle [13] which is different from that described in Section II-A.

[Question 2 (Counting Ball Puzzle)]

Suppose that a cube with a size of 3×3×3 is made up by piling up small transparent cubes with a size of 1×1×1 as shown in Fig. 3 (a). Let some of these small cubes in the 3×3×3 cube have balls, and then its perspective view from front, top, and left side is given in Fig. 3 (b) (for example). Find the upper limit of the number of balls which are included in this 3×3×3 cube under the above conditions.

(End of Question)

We treat the puzzle shown in Fig. 3 (a) as a third-order tensor. Now note that the empty (transparent) elements in Fig. 3 (b) is continuously empty toward the rear and there are no balls. We can solve this question also by using the *n*-mode matrix unfolding.



(a) 3-d puzzle with size 3 × 3 × 3   (b) Perspective view from each side

Fig. 3. Example of another 3-d puzzle.

[R script to solve Quenstion 2]

```
# To solve Question 2
# initialization of tensor A (Suppose all small cubes have a
# ball, so the initial values are 1's.)
A <- array( 1, c(3,3,3) )
A <- as.tensor( A )
# construct a tensor A (empty element is 0)
A[1,3, ] <- 0   #  no ball from front (1,3,1) to backward
A[2,1, ] <- 0   #          〃          front (2,1,1)          〃
```

```
A[3,1, ] <- 0  #       〃      front (3,1,1)       〃
A[3,3, ] <- 0  #       〃      front (3,3,1)       〃
A[ ,1,1] <- 0  # no ball from top (1,1,1) to downward
A[ ,1,3] <- 0  #      〃      top (1,1,3)       〃
A[ ,2,1] <- 0  #      〃      top (1,2,1)       〃
A[ ,3,1] <- 0  #      〃      top (1,3,1)       〃
A[ ,3,2] <- 0  #     〃      top (1,3,2)       〃
A[1, ,1] <- 0  # no ball from left side (1,1,1) to rightward
A[1, ,3] <- 0  #      〃      left side (1,1,3)       〃
A[2, ,1] <- 0  #      〃      left side (2,1,1)       〃
A[3, ,1] <- 0  #      〃      left side (3,1,1)       〃
A[3, ,3] <- 0  #      〃      left side (3,1,3)       〃
 # 1-mode matrix unfolding of tensor A
A_1mode <- unfold( A, 1, c(3,2) )
# Count of the number of cubes including balls.
ans2 <- sum( A_1mode@data )
```

(End of Script)

Table V shows the 1-mode matrix unfolding of tensor *A*, which represents the existence of balls in the puzzle. We can make sure that this matrix unfolding expresses precisely the ball arrangement shown in the perspective view of Fig. 3 (b). From the execution result of this script, since the variable *ans2* for storing the answer is 6, we can find that the maximum number of the balls is 6 in the puzzle.

TABLE V: 1-MODE MATRIX UNFOLDING OF 3RD ORDER TENSOR A

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]	[,8]	[,9]
[1,]	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**0**
[2,]	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**1**
[3,]	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**

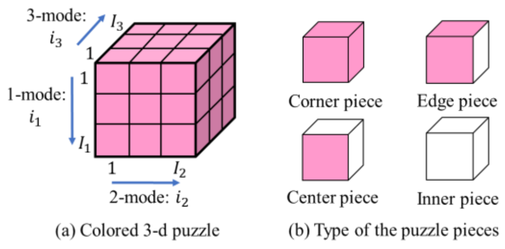


Fig. 4. Example of painted box puzzle.

D. Example of Solution 3 for Question 3

In this example, we deal with the painted box puzzle as shown in Fig. 4 (a) [14]. This puzzle is a rectangular solid with a size of $I_1 \times I_2 \times I_3$, where I_1, I_2, and I_3 are all assumed to greater than 2.

[Question 3 (Cutting Painted Box Puzzle)]

In Fig. 4 (a), the solid is constituted by piling up $I_1 \times I_2 \times I_3$ white cubes with a size of $1\times1\times1$, and the whole surfaces of the solid are painted. Count the number of $1\times1\times1$ cubes with two-sided painted and unpainted (white) $1\times1\times1$ cubes in the solid, respectively. Here, the size of the solid is assumed to be greater than $2\times2\times2$.

(End of Question)

The puzzle of Fig. 4 (a) is consisted of four types of pieces as shown in Fig. 4 (b). The corner, edge, and center pieces are the $1\times1\times1$ cubes with painted on three, two, and one sides, respectively, and the inner pieces are unpainted cubes.

In the same way as previous questions, we express the puzzle as a third-order tensor and solve by applying the *n*-mode matrix unfolding. Here, the element values of the tensor are the number of painted sides of each $1\times1\times1$ cube.

The following script is an example of the solution for this 3-d puzzle whose size is $I_1 \times I_2 \times I_3 = 3\times4\times5$. Note that the words "front layer" and "back layer" in the comment of the R script mean $I_1 \times I_2$ matrices when $i_3 = 1$ and $i_3 = I_3$, respectively.

[R script to solve Question 3]

```
# To solve Question 3
# substitution of 3-d puzzle size (each greater than 2)
I1 <- 3; I2 <- 4; I3 <- 5
# variables to specify the number of colors in each piece
inner <- 0; center <- 1; edge <- 2; corner <- 3
A <- array( 0, c(I1,I2,I3) )
A <- as.tensor( A )
# construction of tensor A
for( i3 in 1:I3 ){
if( i3==1 || i3==I3 ){   # processing of front and back layers
   # initialization of each layer. (The initial value is
   # the number of the colors of edge piece.)
      A[ , , i3] <- edge;
   # substitution of the number of colors in corner pieces
      A[1,1,i3] <- A[1,I2,i3] <- corner
      A[I1,1,i3] <- A[I1,I2,i3] <- corner
   # substitution of the number of colors in center pieces
      if( I1>=3 && I2>=3 ) A[2:(I1-1),2:(I2-1),i3]<- center
} else {   # processing of other layers
   # initialization of each layer (The initial value is
   # the number of the colors of center piece.)
      A[ , , i3] <- center;
   # substitution of the number of colors in edge pieces
      A[1,1,i3] <- A[1,I2,i3] <- edge
      A[I1,1,i3] <- A[I1,I2,i3] <- edge
   # substitution of the number of colors in inner pieces
      if ( I1>=3 && I2>=3 ) A[2:(I1-1),2:(I2-1),i3] <- inner
   }
}
# 1-mode matrix unfolding of tensor A.
A_1mode <- unfold( A, 1, c(3,2) )
A_1mode@data   # display of A_1mode
# count of the frequency of each puzzle piece
ans3 <- as.data.frame( table(A_1mode@data) )
colnames( ans3 ) <- c( "Number of cols.", "Freq." )
rownames( ans3 ) <- c( "Inner piece", "Center piece",
"Edge piece", "Corner piece" )
ans3   # display of results
```

(End of Script)

Fig. 5 shows the result of the 1-mode matrix unfolding *A_1 mode* obtained by executing the above script as a heat map. From the map, we can make sure that this script generates the unfolding matrix with correct number of colors as the element values.

The variable *ans3* in the script indicates the frequency of each piece in this puzzle, and it can be displayed by the following command:

```
> ans3   # display of the frequency of each piece
```

The results of the frequency are summarized in Table VI. The question is to find the number of edge pieces and inner pieces, and we can obtain the answers as 24 and 6, respectively.

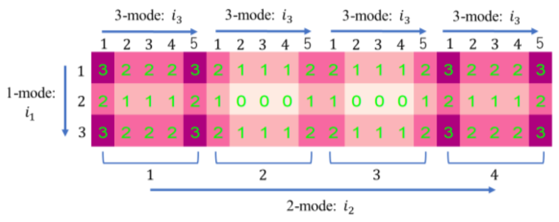


Fig. 5. 1-mode matrix unfolding of the painted box puzzle.

TABLE VI: FREQUENCY OF EACH PIECE

	Number of cols.	Freq.
Inner piece	**0**	**6**
Center piece	**1**	**22**
Edge piece	**2**	**24**
Corner piece	**3**	**8**

IV. RESULTS AND DISCUSSION

The approach to support understanding of HOSVD by using the 3-d puzzles described in the previous section was started by us in the creative experiment of 4th grade subjects in 2015 in our College. This subject is for students who completed basic education of information engineering to work on creative themes for half a year under the guidance of supervisors. We gave our student interested in data analysis the theme of devising a method to solve the puzzle in Ref. [9] (Question 1 in Section II-A) using HOSVD and its implementation to personal computer (PC). It was our aim to make this student familiar with HOSVD principle and computation through the experiment. The teaching procedure and the results carried out by the student are as follows:

1) First, a teacher (one of us) lectured to the student about the principles of SVD and HOSVD, and then gave him Refs. [2] and [3] for further study.
2) Next, the teacher gave the students an assignment of creating a program to represent the 3-d puzzle with a higher-order tensor and convert it into *n*-mode matrices.
3) As a next assignment, the teacher instructed the students to devise plural ways to solve this puzzle (counting the number of holes) using the *n*-mode matrices. As the result, Solution method 1 and 2 in Section 3-A were obtained.
4) Finally, the teacher instructed the student to program these solutions and implement them on PC. This student implemented them in C language.

The above is an actual example using the problem of solving 3-d puzzles to the education of HOSVD. In general, programs related to HOSVD can be written with simpler instructions by using R packages for tensor data analysis compared to not using it, but since this student used C language, he programed every detail, such as tensor data creation and *n*-mode matrix conversion (i.e. unfolding). From this, as he was able to do it, we saw that he had understood the principle of HOSVD very well. The student was able to smoothly connect the learning outcomes of this experiment to the application of HOSVD in graduation research of the 5th (final) grade subjects.

The solutions to this puzzle were simply to count the number of holes in the *n*-mode unfolding matrix, and the student asked us that he wanted to solve other 3-d puzzles as well. Thus, we worked on finding and creating further 3-d puzzles, and consequently devised solutions based on the *n*-mode matrix unfolding for Questions 2 and 3 mentioned in Section III-C and III-D. Those puzzles were added to the teaching materials for educational support of HOSVD. We continuously used these teaching materials after this student in our laboratory and got feedbacks from students that they could learn easily and fun. In addition, these puzzles were also demonstrated to junior high school students and their parents, and others at the Open Campus of our college, and many of them were interested in such puzzles and solution methods.

From the above, it is thought that the method of using 3-d puzzles for learning tensor data handling and fundamentals of tensor decomposition is effective.

V. FUTURE WORK

As further work, we are considering the following approach on application of 3-d puzzles to education of multi-dimensional data processing:

1) To devise new puzzles and their solutions that can be solved using HOSVD, where the dimensions of the puzzles may be three or more dimensions.
2) To implement solution algorithms for newly devised puzzles in 1) in R language.
3) To collect all the puzzles which we have devised and to have them be used widely by teachers and students at the educational sites, because we have confirmed that it is effective for the education to let students think about solutions using HOSVD, as a representative of the tensor decomposition methods, and solve in R language. Note that the language used may be not only R language but also C language, Java, Phython [15], Scilab [16], and others.

VI. CONCLUSION

In this paper, we described a new approach using 3-d puzzles for understanding tensor decomposition, which is widely known as an effective technique for big data analysis [17], especially taking up HOSVD. Data science education for science and engineering students is expected to become increasingly important in the future not only in Japan but also all over the world. In doing this education, our approach is considered very useful. This can be seen from our experiences taught students for recent few years.

Here we cited up a few examples of 3-d puzzles, but we could take up other various types of puzzles (though they are also future work). For examples, we could give magic cubic puzzles, which are the extension of magic squares to 3-d, as the numerical puzzles to solve [18]. On the other hand, those 3-d puzzles can also be thought of as non-numerical 3-d ones like the Rubik's cubes [19].

Since the R language is a very useful programming language for statistical analysis, the calculation programs that solve the 3-d puzzles was posted as a reference in this paper.

References

[1] J. Murakami, N. Yamamoto, and Y. Tadokoro, "High-speed computation of 3dtensor product expansion by the power method," *Electron. Commun. Jpn.*, vol. 85, pp. 63-72, 2002.

[2] L. D. Lathauwer, B. D. Moor, and J. Vandewalle, "A multilinear singular value decomposition," *SIAM J. Matrix Anal. Appl.*, vol. 21, pp. 1253-1278, 2000.

[3] A. Ishida, K. Kawakami, D. Furushima, N. Yamamoto, and J. Murakami, "Analysis of relationships between amount of physical activity of patients in rehabilitation and their ADL scores using Multidimensional PCA," *Adv. Intell. Syst. Comput.*, vol. 690, pp. 147-158, 2017.

[4] A. Ishida, K. Fujii, N. Yamamoto, J. Murakami, S. Saito, and N. Kano, "Application of nonnegative Tucker decomposition in medical data analysis," *Adv. Mater. Res.*, vols. 971-973, pp.1874-1883, 2014.

[5] H. Maruyama, N. Kamiya, T. Higuchi, and A. Takemura, "Developing data analytics skills in Japan: Status and challenge," *J. Jpn. Industr. Manag. Ass.*, vol. 65, no. 4E, pp. 334–339, 2015.

[6] R C. Team, "R: A language and environment for statistical computing," *R Foundation for Statistical Computing*, Vienna, Austria, 2012.

[7] C. Okuma, S. Nagaoka, J. Murakami, N. Yamamoto, and A. Ishida, "Development of understanding support system of higher order singular value decomposition by visualizing its calculation process," *J. Education in the Colleges of Technol.*, vol. 38, pp. 129-134, 2015.

[8] T. A. Lambert and P. A. Whitlock, "Generalizing Sudoku to three dimensions," *Monte Carlo Methods and Appl.*, vol. 16, nos. 3-4, pp. 251-263, 2010.

[9] The Mathematics Certification Institute of Japan, "Atama ga shibireru! Kono 1-mon! (This one question! Which stimulates the head!)," *MathMath+*, vol. 59, p. 15, 2015.

[10] L. D. Lathauwer and J. Vandewalle, "Dimensionality reduction in higher-order signal processing and rank-$(R_1, R_2, \cdots, R_N)$ reduction in multilinear algebra," *Linear Algebra Appl.*, vol. 391, pp. 31-55, 2004.

[11] S.-J. Wang, C.-G. Zhou, Y.-H. Chen, X.-J. Peng, H.-L. Chen, G. Wang, and X. Liu, "A novel face recognition method based on sub-pattern and tensor," *Neurocomputing*, vol. 74, no. 17, pp. 3553-3564, 2011.

[12] J. Li, J. Bien, and M. Wells. (Dec. 2015). *Package 'rTensor'*. [Online]. Available: http://cran.r-project.org/web/packages/rTensor/rTensor/pdf

[13] T. Shigeo, "Ura kara toku (Solve from the backside)," in *Proc. Sugaku Seminar*, 1984, vol. 23, no. 4.

[14] K. Okano, *Oyako de tanoshimu zukei Puzzle <2>*: *Rittai* (*Shape Puzzles that are enjoyable for children and parents <2>*: *Solid*), Tokyo, Kyoritsu Shuppan, 1986, pp. 6-7.

[15] Python Software Foundation. (2018). Welcome to Python.org. [Online]. Available: https://www.python.org/

[16] Scilab Enterprises. (2017). Scilab: home. [Online]. Available: https://www.scilab.org/

[17] A. Cichocki, "Era of big data processing: A new approach via tensor networks and tensor decompostions," in *Proc. Int. Work. Smart Info-Media Sys. Asia (SISA2013)*, Nagoya, Japan, Oct. 2013.

[18] M. Trenkler, "A construction of magic cubes," *The Math. Gazette*, vol. 84, pp. 36-41, 2000.

[19] Lubik's Brand Ltd. (2018). The home of Rubik's Cube. [Online]. Available: https://www.rubiks.com

A. Ishida received the M.S. and Ph.D. in s cience from Kumamoto University, Japan, in 2010 and 2014. He is currently an assistant professor of the Faculty of Liberal Studies, Kumamoto College, National Institute of Technology, Japan. His research interests are in the area of multi-dimensional data analysis, numerical calculation.

N. Yamamoto received the Ph.D. in engineering from Kyushu Institute of Technology, Japan, in 2001. He is currently a professor of the Department of Human-Oriented Information Systems Engineering, Kumamoto College, National Institute of Technology, Japan. His research interests are in the area of multidimensional data analysis and numerical calculation. He is a member of the Institute of Electronics, Information and Communication Engineers (IEICE) and Kyushu Society for Engineering Education (KSEE).

J. Murakami received the Ph.D. in engineering from Toyohashi University of Technology, Japan, in 2000. He is currently a professor of the Department of Human-Oriented Information Systems Engineering, the Co-Director of ICT Center, Kumamoto College, National Institute of Technology, Japan. His research area of interests includes statistical analysis, numerical calculation, and digital signal processing. He is a member of the Institute of Electronics, Information and Communication Engineers (IEICE), the Information Processing Society of Japan (IPSJ), and the Human Interface Society Japan (HISJ).

N. Oishi received the M.E. from Toyohashi University of Technology, Japan, in 1988 and Ph. D. in engineering from Kyushu Institute of Technology, Japan, in 2004. He is currently a professor and the dean of the Department of Information, Communication and Electronic Engineering, Kumamoto College, National Institute of Technology, Japan. His research area of interests includes numerical calculation, statistical analysis, and causal analysis of multidimensional data. He is a member of the Physical Society of Japan (JPS) and the Japan Society of Applied Physics (JSAP).