論　文

立体パズルを利用した多次元データ分解手法の理解支援の試み

山本　直樹*　　石田　明男**　　平田　将大*†　　村上　純***

A Trial to Support to Understand Decomposition Methods for Multidimensional Data by Solving 3-D Puzzles

Naoki Yamamoto*, Akio Ishida**, Shodai Hirata*†, Jun Murakami***

We have studied about multidimensional data analysis, and the related themes with this have been given to our students in their graduation research. Although the higher-order singular value decomposition (HOSVD) method is often used to treat multidimensional data, such as many big data processing problem, it is difficult for the students to understand the fundamental principle of it. In order to improve this situation, as our previous work, we developed an understanding support system of HOSVD by visualizing its calculation process. This time, we tried to give three-dimensional (3-D) puzzles to the students to solve by using HOSVD and to let them study that principle with interest. We describe our trial in this paper.

キーワード：多次元データ，テンソル分解，HOSVD，立体パズル，理解支援

Keywords：Multidimensional data, tensor decomposition, HOSVD, 3-D puzzles, understanding support

1．はじめに

我々の研究グループでは，多次元配列のような高次元のデータを低次元に分解する計算アルゴリズムの開発と，分解した結果を利用したデータ分析に関する研究を行っている[(1),(2)]．データ分析では，最近，ビッグデータ解析が盛んになっていることから，我々も医療データを対象とした多次元データの分析に主に取り組んでいる．多次元データ分析を行うに当たっては，行列の特異値分解（singular value decomposition, SVD）[(3)]として知られている手法を多次元へ拡張した高次特異値分解（higher-order singular value decomposition, HOSVD）[(4),(5)]を主に用いるが，多次元データの概念やそれに関する演算は複雑かつ難解で，理解するのに時間を要する．

5年生の卒業研究や4年生の学生実験（後期の創造実験）において，多次元データ分析に関するテーマを与えた場合，HOSVDを理解させるだけでかなりの時間が必要な上に，適当なテキストや演習課題などがないため，学習させながら実際に分解計算のプログラミングに取りかからせたりしていた．この状況を改善するため，数年前からは，学生が低学年時にプログラミング入門用に学んでいるProcessing[(6)]などの言語を用いて，HOSVDの分解過程のCG表示を課題として与え，分解手法を理解させることにしている[(7),(8)]．今回，立体数独などの立体パズル作成が近年盛んであること[(9)]に着目して，HOSVDを利用してパズルを解くアイディアを学生に考えさせることを通じて，この分解手法の理解につなげるという教育手法を試みた．

*　人間情報システム工学科
〒861-1102　熊本県合志市須屋2659-2
Dept. of Human-Oriented Information Systems Engineering, 2659-2 Suya, Koshi-shi, Kumamoto, Japan 861-1102
**　共通教育科
Faculty of Liberal Studies
*†　人間情報システム工学科（現：株式会社ジュピターテレコム）
Present affiliation: Jupiter Telecommunications Co., Ltd.
***　専攻科電子情報システム工学専攻
Advanced Course of Electronics and Information Systems Engineering

2．高次特異値分解とは

2.1　立体パズルとそのテンソル表現

本論文で取り扱う立体パズルは，サイズ$1\times1\times1$の立方体を図1のように積み重ねて作ったサイズ$I_1\times I_2\times I_3$の立体である．この立方体を用いたパズルの例[(10)]を次に示す．

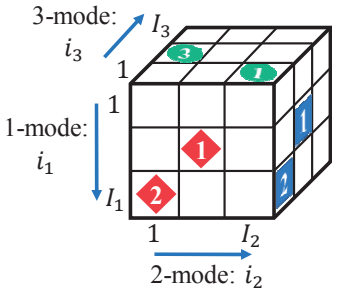


図1　立体パズルの例
(Fig.1 Example of 3-D puzzle)

［問題１］図 1 において見えている面に与えられた数字は，その面から穴を続けて開けた$1\times1\times1$の立方体の個数を表す．なお，見えていない面からは穴は開けていない．このとき，1 回も穴が開いていない$1\times1\times1$の立方体は何個存在するか求めよ．

（問題終わり）

本論文では，上記のような問題を解くために，立体パズルを高階テンソルで表現する．高階テンソルとは多次元配列を意味し，例えば，1 階テンソルはベクトル，2 階は行列，3 階は 3 次元配列に対応する．

ここで，図1の立体パズルを用いてテンソルの表記について説明する．このパズルは，図 1 のようにテンソルサイズが$I_1\times I_2\times I_3=3\times3\times3$の 3 階テンソルで表現することができる．図 1 の矢印で示されるテンソルの縦・横・奥方向はモードと呼ばれ，ここでは，テンソルの縦方向を 1-モード，横方向を 2-モード，奥方向を 3-モードとする．このパズルの 3 階テンソルを$\mathcal{A}$と表すと，$\mathcal{A}$は図 1 の各モードに関する変数i_1,i_2,i_3を用いて$\mathcal{A}=\left(a_{i_1i_2i_3}\right),(i_1=1,\cdots,I_1;\ i_2=1,\cdots,I_2;i_3=1,\cdots,I_3)$と表現できる．ただし，$a_{i_1i_2i_3}$は$\mathcal{A}$の$(i_1,i_2,i_3)$要素である．$a_{i_1i_2i_3}$は$1\times1\times1$の立方体に対応し，それぞれに要素値を与えることができるため，もし穴が開いた立方体ならば$a_{i_1i_2i_3}$=0，穴がなければ$a_{i_1i_2i_3}$=1 として穴の有無を区別できる．

便宜上，図 1 のような立体パズルにおいて，3-モードの変数を$i_3=1$と固定して，1-モードおよび 2-モードの変数が任意である面$\left(a_{i_1i_21}\right),(i_1=1,\cdots,I_1;\ i_2=1,\cdots,I_2)$，すなわち，赤色の菱形のある面を 1-2 面と呼ぶ．同様に，$i_1=1$と固定して緑色の丸のある面を 2-3 面，$i_2=I_2$と固定して青色の四角のある面を 1-3 面と呼ぶ．

2.2　高次特異値分解の概要とアルゴリズム

高次特異値分解（HOSVD）[4),(5)]は，第 1 章で述べたように行列の特異値分解（SVD）を 3 階以上の高階テンソルに拡張したもので，信号処理をはじめ画像処理，パターン認識，データ解析などで応用される多次元データ分解手法である．今回我々は，HOSVD アルゴリズムを構成する重要な処理である *n*-モード行列展開を利用して 2.1 節の立体パズルを解くことができることに着目し，4 年生の学生実験（創造実験）において「*n*-モード展開を利用した立体パズルの解法の作成」として実施した．

以下では，HOSVD アルゴリズムについて述べるが，特に *n*-モード行列展開については，次節で詳しく説明する．なお，本論文では，立体パズルは 3 階テンソルとして表現するため，3 階テンソルの HOSVD について述べるものとする．

3 階テンソル$\mathcal{A}$の HOSVD は正規直交行列$U^{(1)},U^{(2)},U^{(3)}$とコアテンソル$\mathcal{B}$により，$\mathcal{A}=\mathcal{B}\times_1U^{(1)}\times_2U^{(2)}\times_3U^{(3)}$と分解する手法のことである．ただし，演算子×$_n$はテンソルと行列の積を取る演算である *n*-モード積を表している（この概念も重要であるが，本論文では説明は割愛する）．コアテンソルは$\mathcal{A}$と同じサイズ（小さくすることも可能）の 3 階テンソルで，SVD で言えば特異値に対応するものである．以下にその計算アルゴリズムを示す．

［アルゴリズム１（HOSVD)］

入力： サイズ$I_1\times I_2\times I_3$の 3 階テンソル$\mathcal{A}$

出力： 正規直交行列$U^{(1)},U^{(2)},U^{(3)}$およびコアテンソル$\mathcal{B}$

（ステップ 1） 入力のテンソル$\mathcal{A}$を *n*-モード行列展開し，行列$A_{(n)},(n=1,2,3)$を計算する．

（ステップ 2） ステップ 1 で得られた行列$A_{(n)}$に SVD を適用して，$A_{(n)}=U^{(n)}\Sigma^{(n)}V^{(n)},(n=1,2,3)$と分解する．ただし，$U^{(n)}$と$V^{(n)}$は左および右特異行列，$\Sigma^{(n)}$は対角要素が特異値の対角行列を表す．

（ステップ 3） 入力テンソル$\mathcal{A}$とステップ 2 で得られた行列$U^{(n)}$の *n*-モード積により，コアテンソルを$\mathcal{B}=\mathcal{A}\times_1U^{(1)^{\mathrm{T}}}\times_2U^{(2)^{\mathrm{T}}}\times_3U^{(3)^{\mathrm{T}}}$として計算する．行列の右肩の T は行列の転置を表す．

（ステップ 4） ステップ 2 で得られた行列$U^{(n)},(n=1,2,3)$とステップ 3 で得られたコアテンソル$\mathcal{B}$を返す．

（アルゴリズム終わり）

2.3　*n*-モード行列展開の概要とアルゴリズム

n-モード行列展開の *n*-モードとは 2.1 節で述べたようにテンソルの方向を表している．例えば，1-モードはもとのテンソルの縦方向を表したが，1-モード行列展開は縦に並んでいるデータをそのまま行列の行となるように並べ替える操作である．したがって，1-モード行列展開によってできた行列の行には，もとのテンソルの 1-モードが移される．同様に，2-モードおよび 3-モード行列展開は，もとのテンソルの 2-モードおよび 3-モードが展開後の行列の行に移される．

これらの行列展開の方法は文献により流儀があって異なるが，本論文では文献(4)の方法を利用する．そのアルゴリズムを以下に示す．

［アルゴリズム 2.1（*n*-モード行列展開)］

入力： サイズ$I_1\times I_2\times I_3$の 3 階テンソル$\mathcal{A}$

出力： *n*-モード行列展開$A_{(n)},(n=1,2,3)$

以下のステップ 1～ステップ 3 では，$i_1=1,\cdots,I_1,i_2=1,\cdots,I_2,i_3=1,\cdots,I_3$のすべての場合について行う．

（ステップ 1） 1-モード行列展開

テンソル$\mathcal{A}$の要素$a_{i_1i_2i_3}$を行列$A_{(1)}$のi_1行$\{(i_2-1)I_3+i_3\}$列となるように展開する．

（ステップ 2） 2-モード行列展開

テンソル$\mathcal{A}$の要素$a_{i_1i_2i_3}$を行列$A_{(2)}$のi_2行$\{(i_3-1)I_1+i_1\}$列となるように展開する．

（ステップ 3） 3-モード行列展開

テンソル$\mathcal{A}$の要素$a_{i_1i_2i_3}$を行列$A_{(3)}$のi_3行$\{(i_1-1)I_2+i_2\}$列となるように展開する．

（ステップ 4） ステップ 1～ステップ 3 で得られた$A_{(n)},(n=$

1,2,3)を返す.

(アルゴリズム終わり)

このアルゴリズムのステップ 1 においては，もとのテンソルの 1-モードを表す添字i_1が，1-モード行列展開$A_{(1)}$の行のサイズとなっていることから，もとのテンソルの 1-モードが行列展開後の行列の行に移されることが分かる．他のモードについても同様に移される．

このアルゴリズムは，もとのテンソルから要素を 1 つずつ取り出して行列展開するが，実装上は複数の要素をベクトルや行列として取り出して展開する方が効率的で，アルゴリズムも理解しやすい．そこで，このような修正を加えたアルゴリズム 2.2 を次に示す.

[アルゴリズム 2.2（n-モード行列展開)]

入力： サイズ$I_1 \times I_2 \times I_3$の 3 階テンソル$\mathcal{A}$

出力： n-モード行列展開$A_{(n)}, (n = 1,2,3)$

（ステップ 1） 1-モード行列展開

テンソル$\mathcal{A}$から部分行列$A_{i_2} = (a_{*i_2*}), (i_2 = 1,\cdots,I_2)$を取り出し，横に並べて$A_{(1)} = (A_1|A_2|\cdots|A_{I_2})$とする．ただし，$(a_{*i_2*})$は$i_2$を固定した$i_1 = 1,\cdots,I_1$， $i_3 = 1,\cdots,I_3$の行列を表し，以下のステップにおいても同様である.

（ステップ 2） 2-モード行列展開

テンソル$\mathcal{A}$から部分行列$A_{i_3} = (a_{**i_3}), (i_3 = 1,\cdots,I_3)$を取り出し，転置して横に並べ$A_{(2)} = ({A_1}^{\mathrm{T}}|{A_2}^{\mathrm{T}}|\cdots|{A_{I_3}}^{\mathrm{T}})$とする.

（ステップ 3） 3-モード行列展開

テンソル$\mathcal{A}$から部分行列$A_{i_1} = (a_{i_1**}), (i_1 = 1,\cdots,I_1)$を取り出し，転置して横に並べ$A_{(3)} = ({A_1}^{\mathrm{T}}|{A_2}^{\mathrm{T}}|\cdots|{A_{I_1}}^{\mathrm{T}})$とする.

（ステップ 4）ステップ 1～ステップ 3 で得られた$A_{(n)}, (n = 1,2,3)$を返す.

(アルゴリズム終わり)

このアルゴリズムから，ステップ 1 では，もとのテンソルを左から右に輪切りにして，得られた部分行列をそのまま横に並べると 1-モード行列展開が得られることが分かる．図 2 は，3 階テンソルで表された立体パズルを 1-モード行列展開したイメージである．他のモード行列展開もこれと同様に考えることができる．このように，立体パズルを 2 次元的に展開することにより，3 次元的に存在する穴の有無や穴の分布がより分かりやすくなると考えられる.

3． 立体パズルの求解

3.1 立体パズルの解法

2.2 節に示した問題 1 のパズルの解法は，本来はしらみつぶしに考えるのであろうが，今回対象とした学生実験では HOSVD を用いて解かせたところ，n-モード行列展開を用いた以下の 2 つの解法が得られた.

[解法 1]

（ステップ 1） 立体パズルから，穴の有無の情報を与えた 3 階テンソル$\mathcal{A}$を作成する.

（ステップ 2） $\mathcal{A}$から 1-モード行列展開$A_{(1)}$を求める.

（ステップ 3）$A_{(1)}$から穴のない情報を持った要素数をカウントすれば，それが解である.

(解法終わり)

[解法 2]

（ステップ 1） 与えられた立体パズルで，1-2 面からのみ穴が開けられた情報を持つ 3 階テンソル$\mathcal{B}$を作成する．同様に，2-3 面，1-3 面からの情報のみを持つ 3 階テンソル$\mathcal{C},\mathcal{D}$をを作成する.

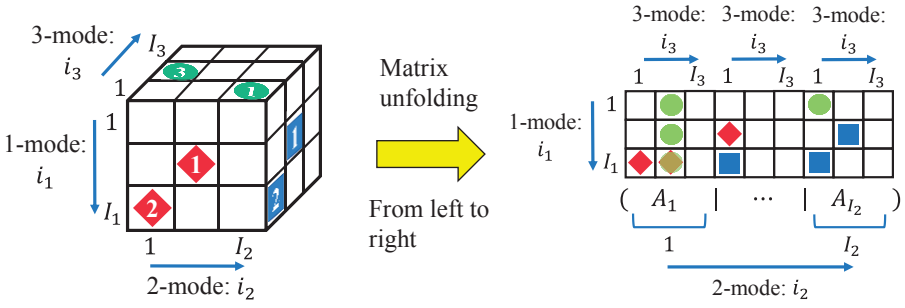
（ステップ 2）3 階テンソル$\mathcal{B},\mathcal{C},\mathcal{D}$について，それぞれの 1-モード展開行列$B_{(1)}, C_{(1)}, D_{(1)}$を求める.

（ステップ 3）1-モード展開行列$B_{(1)}, C_{(1)}, D_{(1)}$の 3 行列の同じ要素に関して，1 つも穴のない情報を持つ要素数をカウントすると，それが解である.

(解法終わり)

解法 1 は図 2 のイメージの処理を忠実に行うもので，図中右側の$A_{(1)}$の色の付いていない要素をカウントするだけでよい．解法 2 は 1 面からのみ開けられた穴の情報をテンソルに与えて解法 1 と同じカウントを行い，これを 3 面分繰り返すもので，解法 1 より手間数は増えるが，分かりやすさはある方法である.

次節では，上述の 2 つの解法を R 言語で実装し，いくつかの立体パズルに適用した結果を示す.



Original 3rd-order tensor $\mathcal{A}$　　1-mode matrix unfolding $A_{(1)}$

図 2　1–モード行列展開の例

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

3.2　求解例 1（問題 1：図 1 のパズル例）

本節では 2.1 節の問題 1 を解くことにする．まず，解法 1 で解く場合について説明する．

(1) 求解例 1.1（解法 1 を利用）

これ以降の求解例では，R にテンソル計算用のパッケージである rTensor を導入し，*n*-モード行列展開には rTensor の関数 unfold を用いているので，その利用法について簡単に説明する[11]．この関数の形式は次の通りである．

unfold(tnsr, row_idx, col_idx)

関数内の引数の tnsr には，もとの高階テンソルを与え，row_idx と col_idx には行列展開の行および列に移す tnsr のモードをそれぞれ指定する．

1-モード行列展開を求める場合は，図 2 に示したように，もとのテンソルのモードが行列展開の行と列として移されるから，引数 row_idx には tnsr の 1-モードが移されるので row_idx=1 を，col_idx には tnsr の 2-モードと 3-モードが移されるので 2 と 3 を指定する．ただし，後者の場合は関数 c でベクトル化し col_idx=c(3,2)とする．なお，col_idx では 3-モード，2-モードの順番で指定しているが，これは，図 2 右側の行列展開$A_{(1)}$の列において，3-モードの添字i_3を 2-モードの添字i_2より先に変化させる必要があるためである．

この例は，次の R スクリプトにより解くことができる．

［求解例 1-1 の R スクリプト］

```
# 問題 1 の解法 1 による求解
 library(rTensor)    # rTensor を利用する（初回時のみ必要）
 I1 ← 3; I2 ← 3; I3 ← 3    # 3 階テンソルのサイズ指定
 A ← array( 1, dim = c(I1,I2,I3) ) # 3 階テンソル A の初期化
 A ← as.tensor( A )    # A を rTensor のオブジェクトに変換
# 図 1 の立体パズルの 3 階テンソルを作成
 A[ 2,2,1 ] ← 0      # A(2,2,1)に 1 つ穴を開ける
 A[ 3,1,1:2 ] ← 0    # A(3,1,1)から連続して 2 つ穴を開ける
 A[ 1:3,1,2 ] ← 0    # A(1,1,2)から連続して 3 つ穴を開ける
 A[ 1,3,1 ] ← 0      # A(1,3,1)に 1 つ穴を開ける
 A[ 2,3,2 ] ← 0      # A(2,3,2)に 1 つ穴を開ける
 A[ 3,2:3,1 ] ← 0    # A(3,3,1)から連続して 2 つ穴を開ける
# 1-モード行列展開
 A_1mode ← unfold( A, row_idx=1, col_idx=c(3,2) )
# 穴の開いていない要素数をカウント
 ans1_1 ← sum( A_1mode @data )
```

（スクリプト終わり）

このスクリプトを実行して求めた 1-モード行列展開 A_1mode の結果を表 1 に示す．表 1 において，1 は穴が開いていない立方体，0 は穴が開いたそれを示す．また，表中の赤，青，緑色は図 2 の各面から開けた穴に対応しており（穴が重なった場合は黄色にしている），それらは図 2 右側の$A_{(1)}$と同じ配置となっているので，正しい結果が得られていることが分かる．

表 1　1-モード行列展開の結果

（Table 1 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

そこで，得られた A_1mode の全要素について 1 の要素をカウントすれば，穴の開いていない立方体の数が分かり，解は変数 ans1_1 に得られ，

```
 > ans1_1    # 穴の開いていない立方体の数の表示
 [1] 18
```

となる．これより，穴の開いていない立方体は全部で 18 個あると求めることができた．

(2) 求解例 1-2（立体パズルの解法 2 を利用）

次に，同じ問題例について，3.1 節の立体パズルの解法 2 により解く．その R スクリプトを次に示す．

［求解例 1-2 の R スクリプト］

```
# 問題 1 の解法 2 による求解
 I1 ← 3; I2 ← 3; I3 ← 3
 B ← array( 1, c( I1, I2, I3 ) )    # 3 階テンソル B の初期化
 C ← array( 1, c( I1, I2, I3 ) )    # 3 階テンソル C の初期化
 D ← array( 1, c( I1, I2, I3 ) )    # 3 階テンソル D の初期化
# B,C,D を rTensor のオブジェクトに変換
 B ← as.tensor( B ); C ← as.tensor( C ); D ← as.tensor( D )
# 1-2 面の穴の情報を B，2-3 面を C，1-3 面を D に与える
 B[ 2,2,1 ] ← 0; B[ 3,1,1:2 ] ← 0    # 1-2 面の情報
 C[ 1:3,1,2 ] ← 0; C[ 1,3,1 ] ← 0    # 2-3 面の情報
 D[ 2,3,2 ] ← 0; D[ 3,2:3,1 ] ← 0    # 1-3 面の情報
# 3 階テンソル B,C,D の 1-モード行列展開
 B_1mode ← unfold( B, 1, c(3,2) )    # テンソル B の展開
 C_1mode ← unfold( C, 1, c(3,2) )    # テンソル C の展開
 D_1mode ← unfold( D, 1, c(3,2) )    # テンソル D の展開
# 穴の開いていない要素数をカウント
 temp ← B_1mode@data*C_1mode@data*D_1mode@data
 ans1_2 ← sum( temp )
```

（スクリプト終わり）

このスクリプトにより，1-モード行列展開 B_1mode，C_1mode，D_1mode には，それぞれもとのテンソルの各面からのみ開けられた穴の情報が反映される．このときの 3 階テンソル B の 1-モード行列展開 B_1mode を表 2 に示す．この表から，B_1mode は図 2 における 1-2 面の赤の菱形で示された穴の情報のみを持っていることが確かめられる．

同様に，表 3 と表 4 にテンソル C および D の 1-モード行列展開の結果 C_1mode と D_1mode を示す．

表 2　3 階テンソル B の 1-モード行列展開
（Table 2 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

表 3　3 階テンソル C の 1-モード行列展開
（Table 3 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

表 4　3 階テンソル D の 1-モード行列展開
（Table 4 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

変数 temp には，表 2〜表 4 の 3 つの行列展開すべてについて穴のない要素が求められ，これは求解例 1-1 による表 1 の 1-モード行列展開 A_1mode の結果と一致している．さらに，最終的な解も両者一致することを確認した．

解法 2 は，表 1 のみを利用する解法 1 と比べ，表 2〜表 4 のようにもとのテンソルの各面ごとの穴の情報を用いるので記憶領域が多く必要となる．しかし次のような利点もある．表 1 の(3,2)要素は 0 で，そこには穴が開いているという情報を持っているが，同じ要素の表 2 と表 3 を見ると，1-2 面と 2-3 面から開けられた穴が重なっていることが分かる（黄色）．このように，重なった穴が開けられた方向を知るためには有用である．

3.3　求解例 2（問題 2：問題 1 の拡張）

次に，問題 1 を任意のサイズの立体（3 次元とする）の任意の位置に任意の個数の穴を開けるパズルに拡張して，この問題を既述の解法 1 および解法 2 により解いてみる．

［問題 2］ 図 1 のようなサイズ$1\times1\times1$の小立方体を積み重ねて作ったサイズ$I_1\times I_2\times I_3$の立体がある．この立体において，次の条件に合った立体パズルを作成し，1 つも穴が開いていない小立方体の個数を求めよ．

（条件 1）立体のサイズは任意とする．

（条件 2）立体の見えている面において，穴を開ける位置を任意に複数個選び，その各位置から任意の個数の小立方体に連続して穴を開けることができること．ただし，1-2 面はI_3個，2-3 面はI_1個，1-3 面はI_2個が、それぞれの面から連続して開けられる穴の最大値である．

（条件 3）立体の見えていない面からは穴は開けないものとする．

（問題終わり）

この問題を解くスクリプトは次のようになる．ただし，ここでは立体パズルのサイズは$3\times2\times4$とし（スクリプトの 3 行目），各面において穴を開ける位置と個数，各位置からて連続して開ける個数を，R の関数 sample を用いてランダムに設定している．

［求解例 2 の R スクリプト］

```
#　問題 2 の解法 1 による求解
 I1 ← 3; I2 ← 2; I3 ← 4　　# 3 階テンソルのサイズ指定
 A ← array( 1, c( I1, I2, I3 ) )　　# 3 階テンソル A の初期化
 B ← array( 1, c( I1, I2, I3 ) )　　# 3 階テンソル B の初期化
 C ← array( 1, c( I1, I2, I3 ) )　　# 3 階テンソル C の初期化
 D ← array( 1, c( I1, I2, I3 ) )　　# 3 階テンソル D の初期化
# A,B,C,D を rTensor のオブジェクトに変換
 A ← as.tensor( A ); B ← as.tensor( B )
 C ← as.tensor( C ); D ← as.tensor( D )
# 各面に開ける穴の最大個数をランダムに選択
 no12 ← sample( 1:(I1*I2), 1 )　　# 1-2 面
 no23 ← sample( 1:(I2*I3), 1 )　　# 2-3 面
 no13 ← sample( 1:(I1*I3), 1 )　　# 1-3 面
# 1-2 面の穴の情報を B，2-3 面を C，1-3 面を D に与える
 for( loop in 1:no12 ){　　# 1-2 面の情報作成
     i1 ← sample( 1:I1, 1 )　　#　添字 i1 をランダムに選択
     i2 ← sample( 1:I2, 1 )　　#　添字 i2 をランダムに選択
     # (i1,i2)から開ける穴の個数をランダムに選択
     hn ← sample( 1:I3, 1 )
     B[ i1, i2, 1:hn ] ← 0　　# 1-2 面に穴を開ける
 }
 for( loop in 1:no23 ){　　# 2-3 面の情報作成，以下同様
     i2 ← sample( 1:I2, 1 ); i3 ← sample( 1:I3, 1 )
     hn ← sample( 1:I1, 1 ); C[ 1:hn, i2, i3 ] ← 0
 }
 for( loop in 1:no13 ){　　# 1-3 面の情報作成，以下同様
     i1 ← sample( 1:I1, 1 ); i3 ← sample( 1:I3, 1 )
     hn ← sample( 1:I2, 1 ); D[ i1, (I2-hn+1):I2, i3 ] ← 0
 }
#　立体パズルの 3 階テンソル作成
 A@data ← B@data * C@data * D@data
# 解法 1 で解く
# 3 階テンソル A の 1-モード行列展開
 A_1mode ← unfold( A, 1, c(3,2) )
#　穴の開いてない要素数をカウント
 ans2_1 ← sum( A_1mode@data )
# 解法 2 による求解
# 3 階テンソル B,C,D の 1-モード行列展開
 B_1mode ← unfold( B, 1, c(3,2) )　　#　テンソル B の展開
```

```
C_1mode ← unfold( C, 1, c(3,2) )    # テンソル C の展開
D_1mode ← unfold( D, 1, c(3,2) )    # テンソル D の展開
# 穴の開いてない要素数をカウント
temp ← B_1mode@data*C_1mode@data*D_1mode@data
ans2_2 ← sum( temp )
```

（スクリプト終わり）

このスクリプトを実行して作成した立体パズルの例を図 3 に示す．この図は，立体の各面に開ける穴の位置とそこから連続して穴を開ける個数を，1-2 面を中心とした展開図で表現している．立体パズルのサイズは変数 I1，I2，I3 に任意の整数値を与えることで自在に変えることができる．

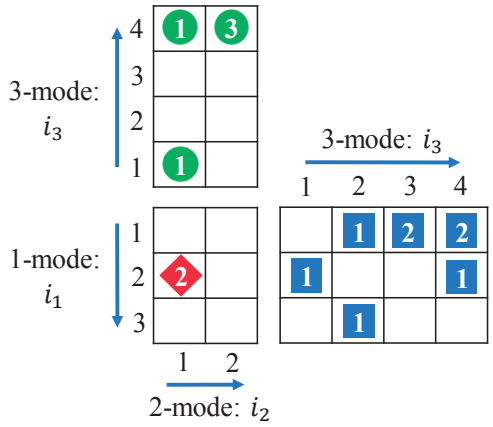


図 3　作成した立体パズルの例
(Fig.3 Example of created 3-D puzzle)

表 5 はすべての穴の情報を持つ立体パズルの 3 階テンソル A を 1-モード行列展開した A_1mode の結果を示すものである．穴のある所は 0 で，色付けは表 1 の場合と同様である．この表から，(1,4)要素，(1,8)要素および(2,8)要素は 2-3 面と 1-3 面から開けた穴が重なっていることが分かる（黄色）が，図 3 の穴の情報は正しく表されている．解法 1 では表 5 の穴のない情報である 1 の要素をカウントして解を求め，変数 ans2_1 に代入する．その値は，

```
> ans2_1
[1] 12
```

となり，立体パズル中で穴のない小立方体は 12 個あることが分かる．

表 6〜表 8 は 1-2 面，2-3 面および 1-3 面のうちの各面のみから開けられた穴の情報を持つ 3 階テンソル B，C，D をそれぞれ 1-モード行列展開した B_1mode，C_1mode，D_1mode を示す．表 6 は 1-2 面のみの穴の情報，つまり図 3 の赤の菱形で示された穴が開けられた情報を，表中の(2,1)要素と(2,2)要素に 0 として持っている．表 7 および表 8 はそれぞれ 2-3 面，1-3 面に関する同様の結果である．解法 2 はこれら 3 つの表を統合した情報を利用して，変数 temp に穴の開いていない要素数をカウントする．その結果，解法 1 と同じ値 12 が得られた．

表 5　3 階テンソル A の 1-モード行列展開
(Table 5 1-mode matrix unfolding of 3rd order tensor A)

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

表 6　3 階テンソル B の 1-モード行列展開
(Table 6 1-mode matrix unfolding of 3rd order tensor B)

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

表 7　3 階テンソル C の 1-モード行列展開
(Table 7 1-mode matrix unfolding of 3rd order tensor C)

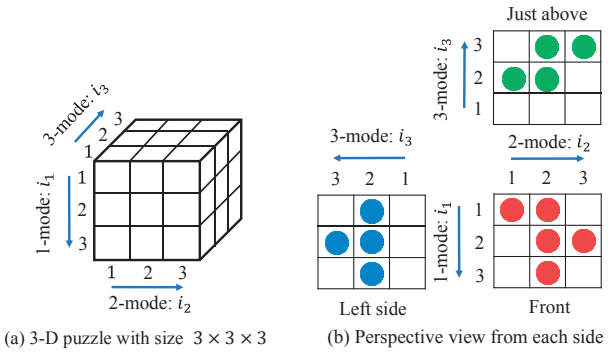
	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]	[,8]
[1,]	0	1	1	0	1	1	1	0
[2,]	1	1	1	1	1	1	1	0
[3,]	1	1	1	1	1	1	1	0

表 8　3 階テンソル D の 1-モード行列展開
(Table 8 1-mode matrix unfolding of 3rd order tensor D)

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

3.4　求解例 3（問題 3：別の立体パズルの例）

ここでは，2.1 節の図 1 の立体パズルおよび 3.3 節で述べたその拡張とは別タイプの立体パズルを考える[12]．



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

図 4　別の立体パズルの例
(Fig.4 Example of another 3-D puzzle)

［問題 3］ 図 4(a)のようなサイズ1 × 1 × 1の透明な小立方体の箱を積み重ねて作ったサイズ3 × 3 × 3の立方体がある．これらの小立方体の箱のいくつかに玉が入っているとき，この立方体を正面，真上および左側面から見た透視図

は図 4(b)のようになる. この立方体中に存在する玉の最大数を求めよ.

（問題終わり）

この問題は, 図 4(a)の立方体を 3 階テンソルとして取り扱い，図 4(b)で各面から見た空白の要素には玉がないことと，*n*-モード行列展開を利用することにより次の R スクリプトで解くことができる.

［問題 3 を解く R スクリプト］

```
# 問題 3 の求解
#A の初期値：全ての箱に玉があるとし 1 を入れる
 A ← array( 1, c(3,3,3) )
 A ← as.tensor( A )    #A を rTensor のオブジェクトに変換
#A の各要素に箱に玉の無い情報（0）を入れる
 A[1,3, ] ← 0   # 正面(1,3,1)から奥に玉がない．以下同様
 A[2,1, ] ← 0   # 正面(2,1,1)から奥
 A[3,1, ] ← 0   # 正面(3,1,1)から奥
 A[3,3, ] ← 0   # 正面(3,3,1)から奥
 A[ ,1,1] ← 0   # 真上(1,1,1)から縦に玉がない．以下同様
 A[ ,1,3] ← 0   # 真上(1,1,3)から縦
 A[ ,2,1] ← 0   # 真上(1,2,1)から縦
 A[ ,3,1] ← 0   # 真上(1,3,1)から縦
 A[ ,3,2] ← 0   # 真上(1,3,2)から縦
 A[1, ,1] ← 0   # 左面(1,1,1)から横に玉がない．以下同様
 A[1, ,3] ← 0   # 左面(1,1,3)から横
 A[2, ,1] ← 0   # 左面(2,1,1)から横
 A[3, ,1] ← 0   # 左面(3,1,1)から横
 A[3, ,3] ← 0   # 左面(3,1,3)から横
# テンソル A の 1-モード行列展開
 A_1mode ← unfold( A, 1, c(3,2) )
# 玉の入った箱の数をカウント
 ans3 ← sum( A_1mode@data )
```

（スクリプト終わり）

表 9 に玉の有無を表すテンソル A の 1-モード行列展開 A_1mode を示す．これから，図 4(b)の透視図の玉の配置を正しく表現していることが確認できる．また，解を与える変数 ans3 の値は 6 となり，玉の最大数は 6 個であることが分かる.

表 9　3 階テンソル A の 1-モード行列展開
（Table 9 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**

4．教育への利用例

4.1　これまでの取り組み

第 1 章で述べたように, これまでに HOSVD や多次元データ処理の入門として，卒業研究や短期留学生のプロジェクト実習に CG で分解過程を表示させるシステム作成に取り組んだ．本論文で述べた立体パズルによる HOSVD の *n*-モード行列展開および多次元データ取り扱いに関する理解支援の取り組みは，平成 27 年度の 4 年生の創造実験において行ったものである.

対象学生は 1 名で，SVD と HOSVD の原理を説明した後，文献(2),(3)を与えて，HOSVD を利用してパズルを解く解法を考案することをテーマとした．学生はそれらの原理を学習，理解した上で，立体パズルを高階テンソルでモデル化し，それを *n*-モード行列展開して解く方法を 2 通り考案した．本論文ではこれらの解法を R 言語のテンソル計算用パッケージを利用して実装したが，当学生は C 言語で，パッケージは利用せずにプログラミングしており，高階テンソルの取り扱いと *n*-モード行列展開についてよく理解できたものと考えられる．学生はこの実験における経験に継続して，5 年生の卒業研究における HOSVD 応用課題にスムースにつなげることができた.

なお，他の立体パズルも解いてみたい旨の感想が学生から聞かれたので，本論文では，当該パズルの拡張版を 1 つと他のタイプのパズルを追加例として挙げた.

4.2　今後の取り組み

前節で述べた創造実験での取り組み，および本論文でまとめた結果を踏まえ，今後は次のような立体パズルの利用を考えている.

(1)　立体パズルとその解法の考案

立体パズルを多次元データ（高階テンソル）と捉え，*n*-モード行列展開やそれ以外の HOSVD を利用して解くことができる新たな立体パズルおよびその解法を考案することをテーマとしたプロジェクトや実験を行う．得られた成果が蓄積されれば，課題集としてまとめて公開したいと考えている.

(2)　既存の立体パズルの解法の学習

本論文で取り上げた立体パズルを課題として与え, *n*-モード行列展開を用いて R 言語により解かせる．これは，本論文で述べた試みが有効であるとして，教育での利用を定着化させることである．多次元データの取り扱いや HOSVD, さらに R 言語を，パズルを解くことで学生に面白く学ばせる取り組みと考えられる. C 言語や Java, Phython[13], Scilab[14] などの言語を用いてもよい.

(3)　任意サイズ立体パズルの作成と求解

3.3 節で述べた問題 2 の課題を与え，さらに，この系統の任意パズルを数パターン作成し，求解プログラムを上記各種プログラム言語で実装させる. 3 階テンソル内の穴の情報が *n*-モード行列展開上でどのように分布するか，サイズを

変えながら観察し，3 次元データの性質や取り扱いに慣れさせる．4 階テンソルへの拡張を考えさせるのも面白いテーマと思われる．

５．まとめ

本論文では立体パズルを用いた HOSVD の理解支援の取り組みについて述べた．AI（人工知能，artificial intelligence）および IoT（Internet of things）とビッグデータ解析の組み合わせは今後さらに重要性を増すと思われ，我々もさらに高度な研究に取り組むことになる．その基礎知識を学生に咀嚼させるためには，ICT（information and communication technology）ツールを利用したり，アクティブ・ラーニングなどの教育手法を用いたりすることが考えられるが，その際の課題としてここで述べた立体パズルは有効であると考えられる．

ここで紹介したパズルは 3 次元のものであるが，n-モード展開では平面に展開されるので，4.2 節で述べたように 4 次元以上の問題を考えることもできる．また，第 1 章で述べた立体数独のように数字を扱うものでは，従来から立体魔法陣も知られており，その解法も詳しく調べられている[15]．これらの立体パズルの解法を考えさせるのも学生にとって興味のある課題であろう．課題や解法が蓄積されれば，公開や配布により，教育において広く利用可能である．

使用した R 言語は統計解析用に開発されたフリーソフトウェアのプログラム言語[16]であるが，統計用のみならず充実した計算関数やグラフィックス関数，ベクトル処理など実用的にも注目されており，Processing 言語と同様に教育上も有用なもので，参考のために本文中に立体パズルの解法のスクリプトを詳しく記述した．

（平成 29 年 9 月 25 日受付）
（平成 29 年 12 月 6 日受理）

参考文献

(1) J. Murakami, N. Yamamoto, and Y. Tadokoro : “High-Speed Computation of 3D Tensor Product Expansion by the Power Method”, Electronics and Communications in Japan, Part 3, Vol.85, pp.63-72 (2002).
(2) A. Ishida, T. Noda, J. Murakami, N. Yamamoto, and C. Okuma : “Calculation of Fourth-Order Tensor Product Expansion by Power Method and Comparison of it with Higher-Order Singular Value Decomposition”, Applied Mechanics and Materials, Vols.444-445, pp.703-711 (2014).
(3) G. H. Golub and C. Reinsch : “Singular Value Decomposition and Least Square Solutions”, Numerische Mathematik, Vol.14, pp.403-420 (1970).
(4) L. De Lathauwer, B. De Moor, and J. Vandewalle : “A Multilinear Singular Value Decomposition”, SIAM Journal on Matrix Analysis and Applications, Vol.21, No.4, pp.1253-1278 (2000).
(5) 森垣潤一，片山薫：「高次特異値分解の画像分類への応用」，第 19 回データ工学ワークショップ，DEWS2008，E10-2 (2008).
(6) “Processing”, https://processing.org/.
(7) 大隈千春，長岡翔，村上純，山本直樹，石田明男：「計算過程の可視化による高次特異値分解の理解支援システムの開発」，高専教育，Vol.38，pp.129-134 (2015).
(8) 石田明男，村上純，山本直樹，タンティップ・チャイッタナガン，シワラック・ソンソムパン：「Processing を用いた HOSVD 計算法の視覚化に関する研究」，第 25 回九州沖縄地区高専フォーラム講演要旨集，p.10 (2015).
(9) 田中貴拓，新谷幹夫，岩穴戸貴祥，白石路雄：「立体数独アプリケーションの開発」，芸術科学会論文誌，Vol.14，No. 6, pp.257-263 (2015).
(10) 日本数学検定協会：「頭がシビれる！この 1 問！」，マスマスプラス，Vol.59，秋冬号, p.15 (2015).
(11) 村上純，日野満司，山本直樹，石田明男：「統計ソフト R による 多次元データ処理入門－仮説検定・分散分析・主成分分析」，4.3 節，日新出版 (2017).
(12) 高木茂男：「裏から解く」，数学セミナー，Vol.23，No.4，表紙 (1984).
(13) “python”, https://www.python.org/.
(14) 川田昌克：「Scilab で学ぶわかりやすい 数値計算法」，森北出版 (2008).
(15) 大森清美：「新編 魔法陣」，第 9 章，冨山房 (1992).
(16) “The R Project for Statistical Computing”, https://www.r-project.org/.