MATLAB algorithms for TP transformation

*Pèter Baranyi and **Yeung Yam

*Integrated Intelligent Systems Japanese–Hungarian Laboratory
Budapest University of Technology and Economics
**Dept. Automation and Computer Aided Engineering
The Chinese University of Hong Kong, Shatin, N.T., Hong Kong

1 Introduction

Higher Order Singular Value Decomposition (HOSVD) based Tensor Product (TP) transformation technique has recently been proposed [1] in the field of Linear Matrix Inequality (LMI) based control theories [2]. The main objective of the TP transformation is that it is capable of transforming a dynamic model to the convex combination of linear systems, whereupon various LMI controller design techniques can readily be executed. The resulting convex combination of linear systems can be immediately viewed as Takagi-Sugeno inference based fuzzy model (TS fuzzy model) or multiple-model. The main objective of this paper is to discuss some programming details of the TP transformation and to propose MATLAB programs as possible implementations. The matrix TP transformation is utilized in the case of dynamic systems. For the sake of simplicity this paper discusses a scalar TP transformation. There are two types of TP transformations. One transforms to minimal basis. Its extended version transforms to convex basis. This paper focuses attention on the one that transforms to minimal basis.

One of the directions of nonlinear control design theories deals with dynamic models given in the state-space form as:

$$s\mathbf{x}(t) = \mathbf{A}(\mathbf{p}(t))\mathbf{x}(t) + \mathbf{B}(\mathbf{p}(t))\mathbf{u}(t) \tag{1}$$

$$\mathbf{y}(t) = \mathbf{C}(\mathbf{p}(t))\mathbf{x}(t) + \mathbf{D}(\mathbf{p}(t))\mathbf{u}(t);$$

where the nonlinear system matrix is:

$$\mathbf{S}(\mathbf{p}(t)) = \begin{pmatrix} \mathbf{A}(\mathbf{p}(t)) & \mathbf{B}(\mathbf{p}(t)) \\ \mathbf{C}(\mathbf{p}(t)) & \mathbf{D}(\mathbf{p}(t)) \end{pmatrix} \in R^{O\times I} \tag{2}$$

and vectors $\mathbf{x}(t)$, $\mathbf{u}(t)$ and $\mathbf{y}(t)$ respectively are the state, input and output vectors; and where $s\mathbf{x}(t) = \dot{\mathbf{x}}(t)$ is for a continuous-time system or $s\mathbf{x}(t) = \mathbf{x}(t+1)$ is for a discrete-time system. Further, $\mathbf{p}(t)$ is time varying and is bounded by the N-dimensional space $\mathbf{p}(t) \in \Omega : [a_1, b_1] \times [a_2, b_2] \times .. \times [a_N, b_N] \subset R^N$. $\mathbf{P}(t)$ may, for instance, include some elements of $\mathbf{x}(t)$ or $\mathbf{u}(t)$. A variety of Linear Matrix Inequality based design techniques have been proposed to the Tensor Product (TP) form of

(1) [2, 3]. Therefore a numerical TP transformation has been proposed in [1] that is capable of transforming (1) to TP form.

The input of the TP transformation is the dynamic system matrix (2) and the transformation space Ω. Another input is λ which is responsible for the transformation accuracy, see later in Section II and III. The output of the transformation is the possible components of the TP form of (2):

$$\{\mathbf{S}_{r=1..R}, \overline{w}_{r=1..R}(\mathbf{p}(t))\} = TPtransf(\mathbf{S}(\mathbf{p}(t)), \Omega, \lambda), \tag{3}$$

where matrices $\mathbf{S}_r \in R^{O \times I}$ are constant linear vertex system matrices. Functions $w_r(\mathbf{p}(t))$ are the basis functions. The line over the basis function, such as $\overline{w}_r(\mathbf{p}(t))$, denotes that the basis is convex. Actually, the TP transformation finds a fixed polytope, where the system varies in: $\mathbf{S}(\mathbf{p}(t)) \in \{\mathbf{S}_1, \mathbf{S}_2, \ldots, \mathbf{S}_R\}$. Having these components the TP form of (2) can be defined as:

$$\mathbf{S}(\mathbf{p}(t)) \underset{\varepsilon}{\approx} \sum_{r=1}^{R} \overline{w}_r(\mathbf{p}(t)) \mathbf{S}_r, \tag{4}$$

where ε is the approximation error which cames form the fact, shown in [4], that the TP models (4), given with bounded number of components, are nowhere dense in the modelling space. This implies that if R, the number of the vertex systems $\mathbf{S}_{r=1..R}$, is bounded then it is impossible to find a TP representation where ε reaches zero in general case. The main objective of the paper is to discuss some programming details of the transformation and to give MATLAB programs as an implementation of the TP transformation. In order to facilitate further reading this paper discusses a simplified version of the above TP transformation. Two ways of the simplification is done. A) The output of equ. (3) and (4) is matrix. The paper focuses on scalar TP transformation instead. The scalar version of (4) approximates a scalar function by a TP form:

$$f(\mathbf{x}) \underset{\varepsilon}{\approx} \sum_{r=1}^{R} w_r(\mathbf{x}) b_r, \tag{5}$$

where b_r and $f(\mathbf{x})$ are scalar. B) The paper discuses a TP transformation that results in a minimal basis instead of convex basis.

2 Scalar TP transformation to minimal basis and its properties

Before going into details let us a have a brief digression here and define a tensor product form where the basis functions are decomposed to all dimensions:

$$f(\mathbf{x}) \underset{\varepsilon}{\approx} \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} .. \sum_{i_N=1}^{I_N} \prod_{n=1}^{N} w_{n,i_n}(x_n) b_{i_1,i_2,..,i_N}, \tag{6}$$

where $\mathbf{x} \in R^N$ is the vector of variables x_n. $w_{n,i}(x_n)$ is the i-th basis function of $x_n \in X_n$. Observe that this can always be given in the form of (5): $y = f(\mathbf{x}) = \sum_{r=1}^{R} \omega_r(\mathbf{x})\beta_r$, where the basis is simply computed by the product of

$\omega_r(\mathbf{x}) = \prod_{n=1}^{N} w_{n,i_n}(x_n)$, and $\beta_r = b_{i_1,i_2,..,i_N}$, where $r = ordering\{i_1, i_2, .., i_3\}$. The function "ordering" results in the linear index equivalent of N dimensional array's index $i_1, i_2, .., i_N$, when the size of the array is $I_1 \times I_2 \times .. \times I_N$. Thus $R = \prod_{n=1}^{N} I_n$. Let us turn back to the main objective of this section.

Transformation 1 *TP transformation functions with $f(\mathbf{x})$ given in the bounded space $\mathbf{x} \in \Omega$ and results in the basis functions $w_{n,i}(x_n)$ and the values $b_{i_1,i_2,..,i_N}$ of the TP form (6):*

$$\{w_{n=1..N,i_n=1..I_n}, b_{i_1,i_2,..,i_N}\} = TPtransf(f(\mathbf{x}), \Omega, \lambda).$$

The TP transformation is computed numerically, which obviously brings numerical error in the results. The error of values $b_{i_1,i_2,..,i_N}$ is about 10^{-15}. It is the typical computer error which comes from the finite digital computation. Another error, let it be denoted by α, is obtained on the basis functions. It could be considerable larger than 10^{-15} and comes from the nature of the TP transformation. In order to control α we should have a few words about λ which is a technical programming parameter of the transformation, see Section III. The greater λ we set the more precise basis we obtain. Let α simply be the error between the basis functions obtained by the TP transformation in ideal case, when $\lambda = \infty$, and the basis functions obtained in real case, when $\lambda < \infty$:

$$\alpha = \max_{n,i_n,x_n} |w_{n,i_n}^{(\lambda=\infty)}(x_n) - w_{n,i_n}^{(\lambda<\infty)}(x_n)|.$$

Therefore, if $\lambda \to \infty$ then $\alpha \to 0$, see Section III. The increase of λ suffers from the fact that it exponentially explodes the computation requirement of the TP transformation. Therefore, the restriction of λ is the computational power at hand. Practically, we set the value of λ as great as possible.

At this point it is worth introducing ε of (6) as another error indicator of the TP transformation. Work [4] proofs that if the number of basis functions is bounded then the TP approximator, equ. (6), is nowhere dense in the space of approximation functions. This means that there are infinite number of functions that cannot exactly be represented in TP form by bounded number of basis functions. This implies that the TP transformation cannot generate exact TP representation in general, because it results in a finite number of basis functions. In ideal case, when we have infinite computational power, it is possible to handle infinite number of basis functions by computer. In reality, we can only say that the more basis functions we can compute the better TP representation can be generated [4]. Even in case, when the given function can be represented by a finite number of basis functions, it is still in question wether we have enough computational capacity to handle sufficient number of them. We can conclude that if we use less number of basis functions than required by the given function $f(\mathbf{x})$ then the TP approximation, equ. (6) becomes only an

approximation $\widehat{f}(\mathbf{x})$ of $f(\mathbf{x})$ (even in case when we assume that $\alpha = 0$). Let this approximation error ε of (6) be defined as:

$$\varepsilon = \max_{\mathbf{x}} |f(\mathbf{x}) - \widehat{f}(\mathbf{x})|.$$

We should remark here that ε and α have strong relation. If the function $f(\mathbf{x})$ cannot be represented in TP form with infinite number of basis functions then ε is not zero even in case when $\alpha = 0$. Let us see the opposite case: if the function $f(\mathbf{x})$ can be represented by TP form with finite number of basis functions then ε is zero only if λ is infinite, namely, $\alpha = 0$.

Let us have a few words about the properties of the basis:

i) (**Orthonormality and normality**) The TP transformation results in orthogonal and normalized basis functions:

$$\lim_{\lambda\to\infty}\left(\int_{X_n} w_{n,i}(x_n)w_{n,i}(x_n)\right) = 1 \quad \text{and} \quad \lim_{\lambda\to\infty}\left(\int_{X_n} w_{n,i}(x_n)w_{n,j}(x_n)\right) = 0,$$

where $n = 1..N; \quad i \neq j; \quad i, j = 1..I_n$.

ii) (**Minimal basis system**) The TP transformation gives the minimal number of basis functions, in the sense that the resulted TP approximation has no variant with less number of basis functions. Namely, the following equality:

$$\sum_{i_1=1}^{I_1}\sum_{i_2=1}^{I_2}..\sum_{i_N=1}^{I_N}\prod_{n=1}^{N} w_{n,i_n}(x_n)b_{i_1,i_2,..,i_N} = \sum_{j_1=1}^{J_1}\sum_{j_2=1}^{J_2}..\sum_{j_N=1}^{J_N}\prod_{n=1}^{N} v_{n,i_n}(x_n)c_{i_1,i_2,..,i_N},$$

where $w_{n,i_n}(x_n)$ and $b_{i_1,i_2,..,i_N}$ are resulted by TP transformation, has no solution for basis $v_{n,i_n}(x_n)$ and values $c_{i_1,i_2,..,i_N}$ if $\exists n : J_n < I_n$.

iii) (**Error bound**) The TP transformation helps with decreasing the number of basis functions subject to minimal reduction error. The transformation relates one value $\sigma_{n,i} \in R^+$ to all basis function $w_{n,i_n}(x_n)$. When the basis functions are discarded then the resulted error is bounded by the sum of the corresponding values $\sigma_{n,i}$. For instance, assume that we have

$$f_1(\mathbf{x}) = \sum_{i_1=1}^{I_1}\sum_{i_2=1}^{I_2}..\sum_{i_N=1}^{I_N}\prod_{n=1}^{N} w_{n,i_n}(x_n)b_{i_1,i_2,..,i_N}.$$

Then, let $w_{1,I_1}(x_1)$ and $w_{2,I_2}(x_2)$ be discarded:

$$f_2(\mathbf{x}) = \sum_{i_1=1}^{I_1-1}\sum_{i_2=1}^{I_2-1}..\sum_{i_N=1}^{I_N}\prod_{n=1}^{N} w_{n,i_n}(x_n)b_{i_1,i_2,..,i_N}.$$

Then

$$\|f_1(\mathbf{x}) - f_2(\mathbf{x})\|_{L_2} = \sigma_{1,I_1}(x_1) + \sigma_{2,I_2}(x_2), \quad \text{and} \quad \max_{\mathbf{x}} |f_1(\mathbf{x}) - f_2(\mathbf{x})| \leq \sigma_{1,I_1}(x_1) + \sigma_{2,I_2}(x_2).$$

3 Computational Details

In order to catch the key idea, let the TP transformation be discussed for two variable functions $f(\mathbf{x})$, $\mathbf{x} \in R^2$, first.

3.1 Two variable case

The steps of the TP transformation:

$$(w_{1,i=1..I}, w_{2,j=1..J}, b_{i,j}) = TPtransf(f(x_1,x_2),\Omega,\lambda)$$

are:

Step 1) Set the transformation space $\Omega : [a_1,b_1] \times [a_2,b_2]$.

Step 2) Define a rectangular grid by values: $a_1 \le g_{1,1} < g_{1,2} < .. < g_{1,M_1} \le b_1$ and $a_2 \le g_{2,1} < g_{2,2} < .. < g_{2,M_2} \le b_2$ which values define the location of grid lines. Values g_{n,m_n} can be arbitrary, but if there is no any specific purposes in mind then it is recommended to locate the grid lines equidistantly. The number of the grid lines (M_1 and M_2) let be defined by λ as: $M_1 = M_2 = \frac{\lambda}{2}$. The locations of the grid lines can automatically be generated as:

$$g_{1,m_1} = a_1 + \frac{b_1 - a_1}{M_1 - 1}(m_1 - 1); g_{2,m_2} = a_2 + \frac{b_2 - a_2}{M_2 - 1}(m_2 - 1).$$

Step 3) Sample the given function $f(\mathbf{x})$ over the grid points:

$$b^s_{m_1,m_2} = f(g_{1,m_1}, g_{2,m_2}); \quad m_1 = 1..M_1; \quad m_2 = 1..M_2,$$

where superscript "s" means "sampled". Construct matrix $\mathbf{B}^s \in R^{M_1 \times M_2}$ from sampled values $b^s_{m_1,m_2}$.

Step 4) Execute Singular Value Decomposition (SVD) on matrix $\mathbf{B}^s$:

$$\mathbf{B}^s \underset{SVD}{=} \mathbf{U}_1 \mathbf{D} \mathbf{U}_2^T. \tag{7}$$

Let us briefly recall the theorem of SVD:

Theorem 1 (Matrix singular value decomposition (SVD)) *Every real valued $(I \times J)$-matrix $\mathbf{B}$ can be written as the product of $\mathbf{B} = \mathbf{U}_1 \cdot \mathbf{D} \cdot \mathbf{U}_2^T$, in which*

1. $\mathbf{U}_1 = \begin{bmatrix} \mathbf{u}_{1,1} & \mathbf{u}_{1,2} & \cdots & \mathbf{u}_{1,I} \end{bmatrix}$ *is a unitary $(I \times I)$-matrix,*
2. $\mathbf{U}_2 = \begin{bmatrix} \mathbf{u}_{2,1} & \mathbf{u}_{2,2} & \cdots & \mathbf{u}_{2,J} \end{bmatrix}$ *is a unitary $(J \times J)$-matrix,*
3. $\mathbf{D}$ *is an $(I \times J)$-matrix with the properties of*

(i) *pseudo-diagonality:*

$\mathbf{D} = diag(\sigma_1, \sigma_2, \ldots, \sigma_{\min(I,J)})$

(ii) *ordering:* $\sigma_1 \ge \sigma_2 \ge \cdots \ge \sigma_{\min(I,J)} \ge 0$. *The σ_i are the singular values of $\mathbf{B}$, and the vectors $\mathbf{U}_{1,i}$ and $\mathbf{U}_{2,j}$ are, respectively, an i-th left and an j-th right singular vector.*

(iii) *rank: the number of the non zero singular values equals the rank of matrix* $\mathbf{B}$.

Therefore, the diagonal matrix $\mathbf{D}$ of equ.: (7) contains the singular values $\sigma_{k=1..min(M_1,M_2)}$ in decreasing order, for instance (assume that $M_1 < M_2$):

$$\mathbf{D} = \begin{pmatrix} \sigma_1 & 0 & \cdots & 0 & \cdots & 0 \\ 0 & \sigma_2 & \dots & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \cdots & 0 \\ 0 & 0 & \cdots & \sigma_{M_1} & \cdots & 0 \end{pmatrix} \in R^{M_1 \times M_2}.$$

Let us partition the resulted matrices as:

$$\mathbf{B}^s \underset{SVD}{=} \begin{pmatrix} \mathbf{U}_1^r & \mathbf{U}_1^d \end{pmatrix} \begin{pmatrix} \mathbf{D}^r & \mathbf{0} \\ \mathbf{0} & \mathbf{D}^d \end{pmatrix} \begin{pmatrix} \mathbf{U}_2^r & \mathbf{U}_2^d \end{pmatrix}^T$$

Superscript "r" means "retained" and superscript "d" means "discarded". Let us retain I number of singular values and discard the rest, namely, matrix $\mathbf{D}^d \in R^{(M_1-I)\times(M_2-I)}$ and they corresponding singular vectors contained in matrices $\mathbf{U}_1^d$ and $\mathbf{U}_2^d$. If we discard only zero singular values then we can write: $\mathbf{B}^s = \mathbf{U}_1^r \mathbf{D}^r \left(\mathbf{U}_2^r\right)^T$, where the size of $\mathbf{D}^r$ is $I \times I$. If matrix $\mathbf{D}^d$ contains nonzero singular values then:

$$\mathbf{B}^s \approx \mathbf{U}_1^r \mathbf{D}^r \left(\mathbf{U}_2^r\right)^T, \quad \text{where the error is:} \quad \|\mathbf{B}^s - \mathbf{U}_1^r \mathbf{D}^r \left(\mathbf{U}_2^r\right)^T\|_{L_2} = \sum_{k=I+1}^{min(M_1,M_2)} \sigma_k.$$

Matrices $\mathbf{U}_1^r$ and $\mathbf{U}_2^r$ are orthogonal and normalized. Therefore, the maximum error is bounded by:

$$\max_{elements} |\mathbf{B}^s - \mathbf{U}_1^r \mathbf{D}^r \left(\mathbf{U}_2^r\right)^T| \leq \sum_{k=I+1}^{min(M_1,M_2)} \sigma_k.$$

Step 5) Determine the basis from matrices $\mathbf{U}_1^r$ and $\mathbf{U}_2^r$. Each column $\mathbf{u}_{1,i=1..I}$ of matrix $\mathbf{U}_1^r \in R^{M_1 \times I}$ determines one basis function of variable x_1. Along in the same line, the columns of matrix $\mathbf{U}_2^r \in R^{M_2 \times I}$ define the basis functions of variable x_2. The values $u_{1,m_1,i}$ of one column define the values of the basis function $w_{1,i}(x_1)$, over $x_1 = g_{1,m1}$:

$$w_{1,i}(g_{1,m1}) = u_{1,m_1,i}.$$

In order to get continuous basis function $w_{1,i}(x_1)$, let the above values $u_{1,m_1,i}$ be connected by straight lines. The less distance we have between points $g_{1,m1}$ the better approximation of the basis function is obtained. In other words, the greater λ we set, namely, the more dense points $g_{1,m1}$ we define the more precise basis we obtain.

Having the basis functions we can write the TP form:

$$\widehat{f}(\mathbf{x}) = \mathbf{w}_1(x_1) \mathbf{D}^r \mathbf{w}_2^T(x_2), \tag{8}$$

where row vectors $\mathbf{w}_n(x_n)$, $n = 1..2$, contain the basis functions as:

$$\mathbf{w}_n(x_n) = \begin{pmatrix} w_{n,1}(x_n) & w_{n,2}(x_n) & \cdots & w_{n,I}(x_n) \end{pmatrix}.$$

Note that (8) is equivalent with the 2 dimensional version of (6):

$$\widehat{f}(\mathbf{x}) = \sum_{i=1}^{I} \sum_{j=1}^{J} w_{1,i}(x_1) w_{2,j}(x_2) d_{i,j},$$

where $I = J$ in the present case and $d_{i,j}$ are the elements of $\mathbf{D}^r$. The piece-wise linear basis defines bi-linear approximation, in the sense that over the rectangulars of the grid we obtain bi-linear functions. We can conclude that if $\lambda \to \infty$ and $\mathbf{D}^d$ has only zero singular values then the error between $\widehat{f}(\mathbf{x})$ and $f(\mathbf{x})$ tends to zero, which is in full accordance with the remarks of the previous section. It is straightforward to see that the features i)..iii) of the TP Transformation is inherited from the SVD. For instance, i) comes from the fact that $\mathbf{U}_n^T \mathbf{U}_n = \mathbf{I}$, where $\mathbf{I}$ is the identity matrix. ii) is guaranteed by the SVD, since the number of the nonzero singular values is the rank of $\mathbf{B}^s$. $\mathbf{B}^s$ can not be generated by a product resulting in a matrix of lower rank. iii) is also ensured by the SVD and piece-wise linear basis. The singular values represent the minimal changes, in the sense of norm L_2, when we decrease the rank of the given matrix. Each singular value is assigned to one column of matrix $\mathbf{U}_n$, hence, to one basis function.

3.2 MATLAB program and numerical example for two variable case

This example is intended to illustrate how to extract the tensor product form of function:

$$y = f(x_1, x_2) = (1 + x_1^{-2} + x_2^{-1.5})^2. \tag{9}$$

Let the transformation space Ω be defined as $x_1 \in [1,5]$ and $x_2 \in [1,5]$. Let the grid density be $\lambda = 25 = 5 \times 5$. Let us discuss a MATLAB algorithm that computes the steps of the TP transformation.

```
clear; M1=5; a1=1; b1=5; M2=5; a2=1; b2=5;
%sampling
for m1=1:M1
  for m2=1:M2
    x1=a1+(b1-a1)/(M1-1)*(m1-1);
    x2=a2+(b2-a2)/(M2-1)*(m2-1);
    Bs(m1,m2)=(1+x1^(-2)+x2^(-1.5))^2;
  end
end
% SVD
[U1,D,U2]=svd(Bs);   diag(D)
input(' number of singular values to keep = '), ns=ans;
U1=U1(:,1:ns);   U2=U2(:,1:ns);   D=D(1:ns,1:ns);
%creating basis functions
figure(1);   plot(U1);   figure(2);   plot(U2);
```

%The resulted approximation
B=U1*D*U2'; figure(3); mesh(B);
% Reduction error
figure(4); mesh(Bs-B);

Executing this program we find that the rank of the sampled matrix $\mathbf{B}^s$ is three. This program results in rough basis functions, see Figure 1, and, hence, in a rough bi-linear approximation of the given function, see Figure 2. When we discard non zero singular values during executing the program, we see that additional reduction error is obtained.

Let us execute the above program with $\lambda = 640000 = 800 \times 800$, so as the program starts as:

clear; M1=800; a1=1; b1=5; M2=800; a2=1; b2=5;
%sampling .....

When we execute the program, we see that the number of singular values is 800. The number of nonzero singular value is three again (in order to see the first ten singular values type "diag(D(1:10,1:10))" instead of "diag(D)"). We also can see that the basis functions are defined by 800 points, which is a considerable improvement, see Figure 3. Figure 4 shows that the approximation is much improved. Note that the number of nonzero singular values will be three even in case when we increase λ to infinity. This means that the given function can exactly be given by 3×3 basis system; we may find the basis via analytic derivations as well.

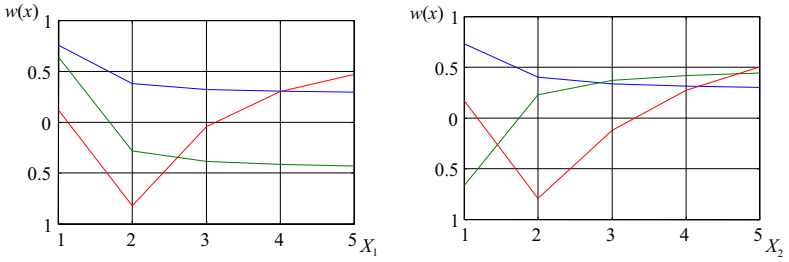

Figure 1: (Extracted basis functions over 5 grid lines

3.3 Multi variable case

This subsection discusses how to calculate the transformation when the number of variables is greater than two, $N > 2$. Before going into details, let as have a brief digression here and introduce some basic tensor operations and notations. Tensor $\mathcal{A} \in R^{I_1 \times I_2 \times .. \times I_N}$ is an N dimensional array of values $a_{i_1,i_2,..,i_N}$, where $i_n = 1..I_n$ and $n = 1..N$.

Definition 1 (n-mode matrix of tensor $\mathcal{A}$) *Assume an Nth order tensor $\mathcal{A} \in R^{I_1 \times I_2 \times \ldots \times I_N}$. The n-mode matrix $\mathbf{A}_{(n)} \in R^{I_n \times J}$, $J = \prod_k I_k$, where $k = 1, \ldots, N$ and $k \neq n$, contains*

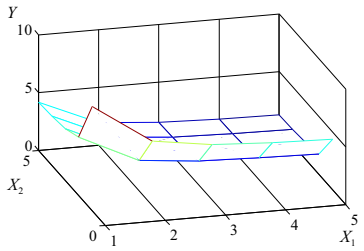

Figure 2: (Approximation over 5×5 grid

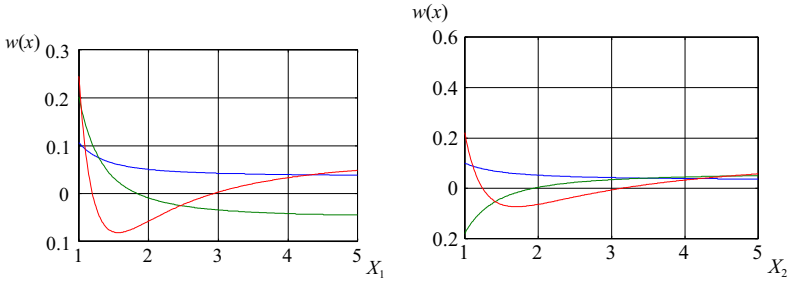

Figure 3: (Extracted basis functions over 800 grid lines

all the vectors in the nth dimension of tensor $\mathcal{A}$. The ordering of the vectors is arbitrary in $\mathbf{A}_{(n)}$. This ordering shall, however, be consistently used later on. $(\mathbf{A}_{(n)})_j$ is called an jth n-mode vector.

Let us discuss the following example: Let $\mathcal{A}$ be a 3 dimensional tensor:

$$a_{1..2,1..2,1} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}; \quad a_{1..2,1..2,2} = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix};$$

A first mode matrix of $\mathcal{A}$ is: $\mathbf{A}_{(1)} = \begin{pmatrix} 1 & 2 & 5 & 6 \\ 3 & 4 & 7 & 8 \end{pmatrix}$. A third mode matrix of $\mathcal{A}$ is: $\mathbf{A}_{(3)} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{pmatrix}$. Let us define a MATLAB program that functions with 3 dimensional tensors. Variable "d" denotes the dimension:

```
function B=layout(d,A)
K=size(A); l=0;
for n=1:3
  if n ~ =d
    l=l+1; p(l)=n;
  end
end
```

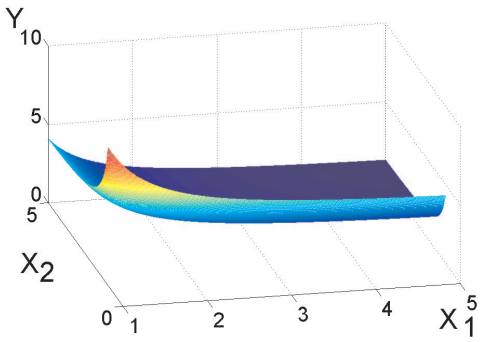

Figure 4: (Example 2/B) Approximation over 800×800 grid

```
I=K; I(d)=1;
for t=1:K(p(1))*K(p(2))
   [i(p(1)),i(p(2))]=ind2sub([K(p(1)) K(p(2))],t);
   for k=1:K(d)
      i(d)=k; B(i(d),t)=A(i(1),i(2),i(3));
   end
end
```

This can easily be extended to N dimension. For instance to 4 dimension:

type "for n=1:4" instead of "for n=1:3";

type "t=1:K(p(1))*K(p(2))*K(p(3))" instead of "t=1:K(p(1))*K(p(2))"

type "[i(p(1)),i(p(2)),i(p(3))]=ind2sub([K(p(1)) K(p(2))],K(p(3)),t)" instead of "[i(p(1)),i(p(2))]=ind2sub([K(p(1)) K(p(2))],t)"

type "B(i(d),t)=A(i(1),i(2),i(3),i(4))" instead of "B(i(d),t)=A(i(1),i(2),i(3))"

Note that any matrix of which the columns are given by n-mode vectors $(\mathbf{A}_{(n)})_j$ can readily be restored to become tensor $\mathcal{A}$. The restoration can be executed even in case when some rows of $\mathbf{A}_{(n)}$ are discarded since the value of I_n has no role in the ordering of $(\mathbf{A}_{(n)})_j$. The restoration in MATLAB can be done by the program below. The restoration has no unique solution. Therefore, an additional parameter of the function is introduced and termed "example". This parameter defines the size of the restored tensor on all dimension except d.

```
function B=restore(d,A,example)
K=size(example);
[s,d]=size(A); l=0;
for n=1:3
   if n ~=d
      l=l+1; p(l)=n;
   end
end
I=K; I(d)=1;
```

```
for t=1:K(p(1))*K(p(2))
   [i(p(1)),i(p(2))]=ind2sub([K(p(1)) K(p(2))],t);
   for k=1:s
      i(d)=k; B(i(1),i(2),i(3))=A(i(d),t);
   end
end
```

Definition 2 (*n*-mode matrix-tensor product) *The n-mode product of tensor* $\mathcal{A} \in R^{I_1 \times I_2 \times \cdots \times I_N}$ *and a matrix* $\mathbf{U} \in R^{J \times I_n}$*, as denoted by* $\mathcal{A} \times_n \mathbf{U}$*, is an* $(I_1 \times I_2 \times \cdots \times I_{n-1} \times J \times I_{n+1} \times \cdots \times I_N)$*-tensor of which the entries are given by* $\mathcal{A} \times_n \mathbf{U} = B$*, where* $B_{(n)} = \mathbf{U} \cdot \mathbf{A}_{(n)}$*. Let* $\mathcal{A} \times_1 \mathbf{U}_1 \times_2 \mathbf{U}_2 \times \cdots \times_N \mathbf{U}_N$ *be denoted as* $\mathcal{A} \bigotimes_{n=1}^{N} \mathbf{U}_n$ *for brevity.*

This product can be given in MATLAB as:
function C=product(d,A,B)
H=layout(d,A); F=B*H; C=restore(d,F,A);

Having these operations we can easily define the multi variable TP transformation.

Note that equ. (6) can be equivalently written by tensor operations:

$$f(\mathbf{x}) = \mathcal{B} \bigotimes_{n=1}^{N} \mathbf{w}_n(x_n),$$

where row vector $\mathbf{w}_n(x_n)$ contains the basis functions $w_{n,i=1..I_n}(x_n)$ as: $\mathbf{w}_n(x_n) = \begin{pmatrix} w_{n,1}(x_n) & w_{n,2}(x_n) & \cdots & w_{n,I_n}(x_n) \end{pmatrix}$. The next part shows how to transform the multi variable function $f(\mathbf{x})$, where $\mathbf{x} \in R^N$.

The computational steps of TP transformation:

Step 1) Set the transformation space: $\Omega : [a_1, b_1] \times [a_2, b_2] \times .. \times [a_N, b_N]$.

Step 2) Define a hyper rectangular grid. Let the grid density of the dimensions be defined by λ: $M_n = \frac{\lambda}{N}$, $n = 1..N$. The equidistantly located grid lines are: $g_{n,m_n} = a_n + \frac{b_n - a_n}{M_n - 1}(m_n - 1)$, $m_n = 1..M_n$.

Step 3) Sampling the given function:

$$b^s_{m_1,m_2,..,m_N} = f(g_{1,m_1}, g_{2,m_2}, .., g_{N,m_N}); \quad m_n = 1..M_n.$$

Step 4) Store the sampled values $b^s_{m_1,m_2,..,m_N}$ into tensor $\mathcal{B}^s$ then execute HOSVD on it: $\mathcal{B}^s = \mathcal{B} \otimes_{n=1}^{N} \mathbf{U}_n$. In order to understand this product, let us recall the definition of HOSVD:

Theorem 2 (Higher Order SVD (HOSVD)) *Every tensor* $\mathcal{A} \in R^{I_1 \times I_2 \times \cdots \times I_N}$ *can be written as the product*

$$\mathcal{A} = \mathcal{S} \bigotimes_{n=1}^{N} \mathbf{U}_n \tag{10}$$

in which

1. $\mathbf{U}_n = \begin{bmatrix}\mathbf{u}_{1,n} & \mathbf{u}_{2,n} & \dots & \mathbf{u}_{I_N,n}\end{bmatrix}$ *is a unitary* $(I_N \times I_N)$*-matrix called n-mode singular matrix.*

2. *tensor* $\mathcal{S} \in R^{I_1 \times I_2 \times \dots \times I_N}$ *whose subtensors* $\mathcal{S}_{i_n=\alpha}$ *have the properties of*

(i) *all-orthogonality: two subtensors* $\mathcal{S}_{i_n=\alpha}$ *and* $\mathcal{S}_{i_n=\beta}$ *are orthogonal for all possible values of* n, α *and* $\beta : \langle \mathcal{S}_{i_n=\alpha}, \mathcal{S}_{i_n=\beta} \rangle = 0$ *when* $\alpha \neq \beta$,

(ii) *ordering:* $\|\mathcal{S}_{i_n=1}\| \geq \|\mathcal{S}_{i_n=2}\| \geq \dots \geq \|\mathcal{S}_{i_n=I_n}\| \geq 0$ *for all possible values of n.*

The Frobenius norm $\|\mathcal{S}_{i_n=i}\|$*, symbolized by* $\sigma_i^{(n)}$*, are n-mode singular values of A and the vector* $\mathbf{u}_{i,n}$ *is an ith singular vector. S is termed core tensor.*

The computation of the HOSVD can be done by executing SVD on each dimension of $\mathcal{A}$. Namely, $\mathbf{U}_n$ in Eq. (10) is determined by executing SVD (Theorem 1) on the n-mode matrix $\mathbf{A}_{(n)}$ of tensor $\mathcal{A}$. After discarding the singular values $\sigma_{n,i}$ one obtains: $\mathbf{A}_{(n)} = \mathbf{U}_{n,1}^r \mathbf{D}_n^r (\mathbf{U}_{n,2}^r)^T = \mathbf{U}_{n,1}^r \mathbf{S}_{(n)}$. Restoring $\mathbf{S}_{(n)}$ into tensor $\mathcal{S}_n$, we have: $\mathcal{A} = \mathcal{S}_n \times_n \mathbf{U}_n^r$. Determination of $\mathbf{U}_{n+1}$ is done in the same way. Again, let SVD be executed on $n+1$-mode matrix $(\mathbf{S}_n)_{(n+1)}$ of tensor $\mathcal{S}_n$ obtained in the above step. This results in: $(\mathbf{S}_n)_{(n+1)} = \mathbf{U}_{n+1,1}^r \mathbf{D}_{n+1}^r (\mathbf{U}_{n+1,2}^r)^T$. Then restoring the product of $\mathbf{S}_{(n+1)} = \mathbf{D}_{n+1}^r (\mathbf{U}_{n+1}^r)^T$ into tensor $\mathcal{S}_{n+1}$ we have: $\mathcal{A} = \mathcal{S}_{n+1} \times_n \mathbf{U}_n \times_{n+1} \mathbf{U}_{n+1}$. Repeating the above steps on each dimension leads to eq. (10). Like in the case of SVD we can give error bound if nonzero singular values are discarded. The error is computed by summing up all the discarded singular values in all dimension:

$$\|\mathcal{B}^s - \mathcal{B}\|_{L_2} = \sum_{n,k_n} \sigma_{n,k_n}, \tag{11}$$

where σ_{n,k_n} denotes the k_n-th discarded singular value obtained by SVD executed in dimension n. More detailed discussion of HOSVD and graphical demonstrations are given in paper [5]. Let us construct a simple MATLAB algorithm. First let us slightly specialize the SVD function of MATLAB:

```
function [U,S]=psvd(A)
[U1,D,U2]=svd(A); diag(D)
input(' number of singular values to keep = '), ns=ans;
U1=U1(:,1:ns); U2=U2(:,1:ns); D=D(1:ns,1:ns);
U=U1; S=D*U2';
```

Let us define the function of 3 dimensional HOSVD as:

```
function [U1,U2,U3,S]=hosvd(A)
H1=layout(1,A); [U1,HN1]=psvd(H1);
A1=restore(1,HN1,A);
H2=layout(2,A1); [U2,HN2]=psvd(H2);
A2=restore(2,HN2,A1);
H3=layout(3,A2); [U3,HN3]=psvd(H3);
S=restore(3,HN3,A2);
```

Step 5) Define the basis function $w_{n,i}$ from the columns of matrices $\mathbf{U}_n$ like in the two variable case.

Having the basis we can write that:

$$f(\mathbf{x}) \underset{\varepsilon}{\approx} \mathcal{B} \overset{N}{\underset{n=1}{\otimes}} \mathbf{w}_n(x_n).$$

Transformation error bound (see property iii) in Section III) Let function $\widehat{f}(\mathbf{x})$ be the TP representation where only zero singular values are discarded. Let $\widehat{\widehat{f}}(\mathbf{x})$ be derived from $\widehat{f}(\mathbf{x})$ where nonzero singular values rae discarded as well. Then

$$\max_{\mathbf{x}} |\widehat{f}(\mathbf{x}) - \widehat{\widehat{f}}(\mathbf{x})| \leq \sum_{n,k_n} \sigma_{n,k_n},$$

where σ_{n,k_n} is from (11). Other form of the error bound is: $\|\widehat{f}(\mathbf{x}) - \widehat{\widehat{f}}(\mathbf{x})\|_{L_2} = \sum_{n,k_n} \sigma_{n,k_n}$. Both error bounds directly come form the error bound of HOSVD.

4 Conclusion

This paper presents the fundamentals of scalar TP transformation and some MATLAB programs as possible implementation. We have not set any restrictions how the function $f(\mathbf{x})$ is defined. This implies that the scalar transformation is capable of transforming, functions given by soft computing tools, for instance, the *Rudas's* type operators [6, 7, 8] based functions can be transformed as well. This would help in fuzzy control with carrying the advantages of the *Rudas's* general operators over the fuzzy control techniques. **Acknowledgement** P.Baranyi is supported by Zoltán Magyary scholarship. This work is supported by FKFP 180/2001.

References

[1] P. Baranyi. HOSVD based TP model transformation as a way to LMI based controller design. *IEEE Trans. IE.* (in press).

[2] K. Tanaka and H. O. Wang. *Fuzzy Control Systems Design and Analysis - A Linear Matrix Inequality Approach*. Hohn Wiley and Sons, Inc., 2001, 2001.

[3] S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan. Linear matrix inequalities in system and control theory. *Philadelphia PA:SIAM*, 1994.

[4] D. Tikk, P. Baranyi, and R.J.Patton. Polytopic and TS models are nowere dense in the approximation model space. *IEEE Int. Conf. System Man and Cybernetics (SMC'02)*, 2002. Proc. on CD.

[5] L. D. Lathauwer, B. D. Moor, and J. Vandewalle. A multi linear singular value decomposition. *SIAM Journal on Matrix Analysis and Applications*, 21(4):1253–1278, 2000.

[6] I. J. Rudas and O. Kaynak. New types of generalized operations. computational intelligence: Soft computing and fuzzy-neuro integrationwith applications. *Springer NATO ASI Series. Series F: Computer and Systems, Sciences*, 192, 1998.

[7] I. J. Rudas and O. Kaynak. Entropy-based operations on fuzzy sets. *IEEE Trans. on Fuzzy Systems*, 6, 1998.

[8] I. J. Rudas and O. Kaynak. Minimum and maximum fuzziness generalized operators. *Fuzzy Sets and Systems*, 98, 1998.