SANDIA REPORT

SAND2004-3487
Unlimited Release
Printed July 2004

A Preliminary Report on the Development of MATLAB Tensor Classes for Fast Algorithm Prototyping

Brett W. Bader and Tamara G. Kolda

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94AL85000.

Approved for public release; further dissemination unlimited.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865)576-8401
Facsimile: (865)576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.doe.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800)553-6847
Facsimile: (703)605-6900
E-Mail: orders@ntis.fedworld.gov
Online order: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2004-3487
Unlimited Release
Printed July 2004

A Preliminary Report on the Development of MATLAB Tensor Classes for Fast Algorithm Prototyping

Brett W. Bader
Computational Sciences Department
Sandia National Laboratories
Albuquerque, NM 87185–0316
`bwbader@sandia.gov`

Tamara G. Kolda
Computational Sciences and Mathematics Research Department
Sandia National Laboratories
Livermore, CA 94551–9217
`tgkolda@sandia.gov`

ABSTRACT

We describe three MATLAB classes for manipulating tensors in order to allow fast algorithm prototyping. A tensor is a multidimensional or N-way array. We present a `tensor` class for manipulating tensors which allows for tensor multiplication and "matricization." We have further added two classes for representing tensors in decomposed format: `cp_tensor` and `tucker_tensor`. We demonstrate the use of these classes by implementing several algorithms that have appeared in the literature.

Keywords: higher-order tensors, n-way arrays, multidimensional arrays, MATLAB

This page intentionally left blank.

1. Introduction. A tensor is a multidimensional or N-way array of data; Figure 1.1 shows a 3-way array of size $I_1 \times I_2 \times I_3$. In this paper, we describe three MATLAB classes for manipulating tensors: `tensor`, `cp_tensor`, and `tucker_tensor`.

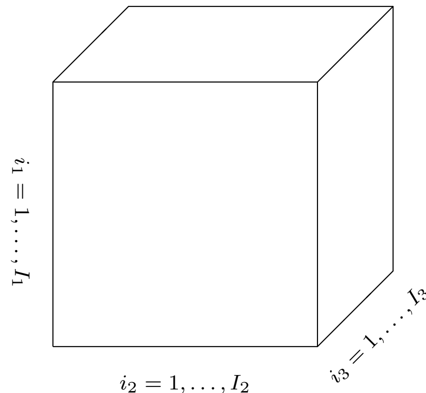

Fig. 1.1. *A 3-way array*

MATLAB is a high-level computing environment that allows users to develop mathematical algorithms using familiar mathematical notation. In terms of higher-order tensors, MATLAB R14 supports multidimensional arrays (MDAs). Allowed operations on MDAs include elementwise operations, permutation of indices, and most vector operations (like `sum` and `mean`) [9]. More complex operations, such as the multiplication of two MDAs, are not supported by MATLAB. This paper describes the use of MATLAB's class functionality [8] to create a `tensor` datatype that extends MATLAB's MDA functionality to support tensor multiplication and more.

Basic mathematical notation and operations for tensors, as well as related MATLAB commands, are described in §2. Tensor multiplication receives its own section, §3, in which we describe both notation and how to multiply a tensor times a vector, a tensor times a matrix, and a tensor times another tensor. Conversion of a tensor to a matrix and vice versa is described in §4.

A tensor may be stored in factored form as a sum of rank-1 tensors. There are two commonly accepted factored forms. The first was developed independently under two names: the CANDECOMP model of Carroll and Chang [2] and the PARAFAC model of Harshman [5]. Following the notation in Kiers [7], we refer to this decomposition as the CP model. The second decomposition is the Tucker [10] model. Both models, as well as the corresponding MATLAB classes `cp_tensor` and `tucker_tensor`, are described in §5.

We note that these MATLAB classes serve a purely supporting role in the sense that these classes do not contain algorithms—just data types. Thus, we view this work as complementary to those packages that provide algorithms, such as Andersson and Bro's N-way toolbox for MATLAB [1].

In general, we use the following notational conventions. Indices are denoted by lowercase letters and span the range from 1 to the uppercase letter of the index, e.g., $n = 1, 2, \ldots, N$. We denote vectors by lowercase boldface letters, e.g., $\mathbf{x}$; matrices by uppercase boldface, e.g., $\mathbf{U}$; and tensors by calligraphic letters, e.g., $\mathcal{A}$. Notation for tensor mathematics is still sometimes awkward. We have tried to be as standard as possible, relying on [6, 7] for some guidance in this regard.

2. Basic Notation & MATLAB Commands for Tensors. Let $\mathcal{A}$ be a tensor of dimension $I_1 \times I_2 \times \cdots \times I_N$. The *order* of $\mathcal{A}$ is N. The nth *dimension* or *mode* or *way* of $\mathcal{A}$ is of size I_n.

Just as an n-vector can be thought of as an $n \times 1$ matrix, an n-vector can be thought of as an order-1 tensor of size n, and an $m \times n$ matrix can be thought of as an order-2 tensor of size $m \times n$.

2.1. Creating a `tensor` object. In MATLAB, a higher-order tensor can be stored as an MDA. We introduce the `tensor` class to extend the capabilities of the MDA object. An array or MDA can be converted to a tensor as follows.

> `T = tensor(A)` or `T = tensor(A,DIM)` converts an array (vector, matrix, or MDA) to a tensor. Here `A` is the object to be converted and `DIM` specifies the dimensions of the object.
>
> `A = double(T)` converts a tensor to an array.

Figure 2.1 shows an example of creating a tensor.

2.2. Tensors and size. Out of necessity, the `tensor` class handles sizes in a different way than the MATLAB arrays. Every MATLAB array has at least 2 dimensions; for example, a scalar is an object of size 1×1 and a column vector is an object of size $n \times 1$. On the other hand, MATLAB drops trailing singleton dimensions for any object of order greater than 2. Thus, a $4 \times 3 \times 1$ object has a reported size of 4×3; see Figure 2.2. The MATLAB `tensor` class explicitly stores the size of its objects, allowing for as few as one dimension as well as for trailing singleton dimensions. Thus, `DIM` must be specified in the constructor whenever the order is one or there are trailing singleton dimensions.

2.3. General functionality. In general, a `tensor` object will behave exactly as an MDA for all functions that are defined for an MDA; see Figure 2.3.

2.4. Accessors. We denote the index of an element within a tensor by either subscripts or parentheses. Subscripts are generally used for indexing on matrices and vectors but can be confusing for the complex indexing that is sometimes required for tensors. For example, $\mathcal{A}(i_1, i_2, \ldots, i_N)$ may be easier to read than $\mathcal{A}_{i_1 i_2 \cdots i_N}$. Furthermore, the parentheses notation is consistent with MATLAB:

> `A(i1,i2,...,iN)` returns the $(i_1, i_2, \ldots, i_N)$ element of $\mathcal{A}$.

We may replace an index with a colon or a range of indices in the same way as is done in MATLAB. Thus, $\mathbf{U}_{i:}$ or $\mathbf{U}(i,:)$ denotes the ith row of $\mathbf{U}$, and $\mathbf{U}_{:j}$ denotes the jth column of $\mathbf{U}$. Likewise, $\mathcal{A}(:,:,k)$ denotes the kth submatrix along the third mode. The MATLAB notation is straightforward:

> `A(:,:,k)` returns the kth 3-mode submatrix of the tensor $\mathcal{A}$.

Figure 2.4 shows an example of accessors for a tensor.

```
% Create a random MDA
A = rand(3,4,2)
A(:,:,1) =
    0.2626    0.0211    0.8837    0.7377
    0.2021    0.0832    0.1891    0.3264
    0.7666    0.1450    0.4118    0.6331
A(:,:,2) =
    0.1501    0.0396    0.7307    0.4609
    0.2340    0.1489    0.6396    0.4528
    0.2955    0.4261    0.1215    0.1157
% Create a tensor
T = tensor(A)
T is a tensor of size 3 x 4 x 2
T.data =
(:,:,1) =
    0.2626    0.0211    0.8837    0.7377
    0.2021    0.0832    0.1891    0.3264
    0.7666    0.1450    0.4118    0.6331
(:,:,2) =
    0.1501    0.0396    0.7307    0.4609
    0.2340    0.1489    0.6396    0.4528
    0.2955    0.4261    0.1215    0.1157
```

FIG. 2.1. *Example of creating a tensor object from a multidimensional array.*

```
% Create an MDA of size 4 x 3 x 1
A = rand([4 3 1]);

% Matlab ignores trailing singleton dimensions
size(A)
ans =
     4     3
ndims(A)
ans =
     2

% Creating a tensor from A creates an order-2
% tensor of size 4 x 3
T = tensor(A);
size(T)
ans =
     4     3
ndims(T)
ans =
     2

% Specifing the dimensions explicitly creates
% an order-3 tensor of size 4 x 3 x 1
T = tensor(A,[4 3 1]);
size(T)
ans =
     4     3     1
ndims(T)
ans =
     3
```

FIG. 2.2. *The* `tensor` *class explicitly tracks the size of its data.*

- `A + B` or `plus(A,B)`
- `A - B` or `minus(A,B)`
- `-A` or `uminus(A)`
- `+A` or `uplus(A)`
- `A.*B` or `times(A,B)`
- `A./B` or `rdivide(A,B)`
- `A.\B` or `ldivide(A,B)`
- `A.^B` or `power(A,B)`
- `A < B` or `lt(A,B)`
- `A > B` or `gt(A,B)`
- `A <= B` or `le(A,B)`
- `A >= B` or `ge(A,B)`
- `A ~= B` or `ne(A,B)`
- `A == B` or `eq(A,B)`
- `A & B` or `and(A,B)`
- `A | B` or `or(A,B)`
- `~A` or `not(A)`

FIG. 2.3. *Functions that behave identically for tensors and multidimensional arrays.*

```
% Create a random 2 x 2 x 2 tensor
A = tensor(rand(2,2,2))
A is a tensor of size 2 x 2 x 2
A.data =
(:,:,1) =
    0.4021    0.8332
    0.6531    0.3029
(:,:,2) =
    0.5953    0.3480
    0.4503    0.3982

% Access the (2,1,1) element
A(2,1,1)
ans =
    0.6531

% Reassign a 2 x 2 submatrix to be
% the 2 x 2 identity matrix
A(:,1,:) = eye(2)
A is a tensor of size 2 x 2 x 2
A.data =
(:,:,1) =
    1.0000    0.8332
         0    0.3029
(:,:,2) =
         0    0.3480
    1.0000    0.3982
```

FIG. 2.4. *Accessors and assignment for a tensor object work the same as they would for a multidimensional array.*

3. Tensor Multiplication.

Notation for tensor multiplication is extremely difficult to understand, particularly because its use is often inconsistent. The issues have to do with defining which indices are to be multiplied and how the modes of the result should be ordered. We approached this problem by considering what could be expressed easily by MATLAB.

3.1. Multiplying a tensor times a matrix.

The first question we consider is how to multiply a tensor times a matrix. With matrix multiplication, the specification of which dimensions should be multiplied is automatic—it is always the rows of the first matrix with the columns of the second matrix. A transpose on an argument swaps the rows and columns. Because tensors may have an arbitrary number of dimensions, the situation is more complicated. In this case, we need to specify which mode of the tensor is multiplied by the columns of the given matrix.

The solution is the n-mode product [3]. Let $\mathcal{A}$ be an $I_1 \times I_2 \times \cdots \times I_N$ tensor. Let $\mathbf{U}$ be an $J_n \times I_n$ matrix. Then the n-mode product of $\mathcal{A}$ and $\mathbf{U}$ is denoted by

$$\mathcal{A} \times_n \mathbf{U}.$$

The result is a tensor of size $I_1 \times \cdots \times I_{n-1} \times J_n \times I_{n+1} \times \cdots \times I_N$. Note that the order of the result is the same as the original tensor. The entries are computed as follows:

$$(\mathcal{A} \times_n \mathbf{U})(i_1, \ldots, i_{n-1}, j_n, i_{n+1}, \ldots, i_N) = \sum_{i_n=1}^{I_n} \mathcal{A}(i_1, i_2, \ldots, i_N)\, \mathbf{B}(j_n, i_n).$$

To understand this notation in terms of matrices (i.e., order-2 tensors), suppose $\mathbf{A}$ is $m \times n$, $\mathbf{U}$ is $m \times k$, and $\mathbf{V}$ is $n \times k$. Then

$$\mathbf{A} \times_1 \mathbf{U}^T = \mathbf{U}^T \mathbf{A} \quad \text{and} \quad \mathbf{A} \times_2 \mathbf{V}^T = \mathbf{A}\mathbf{V}.$$

Similarly, the matrix SVD can be written as

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \mathbf{\Sigma} \times_1 \mathbf{U} \times_2 \mathbf{V}.$$

The following MATLAB commands can be used to calculate n-mode products.

`B = product(A,U,n)` calculates $\mathcal{B} = \mathcal{A} \times_n \mathbf{U}$.

`B = product(A,{U,V},[m,n])` calculates $\mathcal{B} = \mathcal{A} \times_m \mathbf{U} \times_n \mathbf{V}$.

The n-mode product satisfies the following property from [3]. Let $\mathcal{A}$ be a tensor of size $I_1 \times I_2 \times \cdots \times I_N$. If $\mathbf{U} \in \mathbb{R}^{J_m \times I_m}$ and $\mathbf{V} \in \mathbb{R}^{J_n \times I_n}$, then

$$\mathcal{A} \times_m \mathbf{U} \times_n \mathbf{V} = \mathcal{A} \times_n \mathbf{V} \times_m \mathbf{U}. \tag{3.1}$$

See Figure 3.1 for an example that demonstrates this property, and Figure 3.2 which revisits the same example but calculates the products using cell arrays.

It is often desirable to calculate the product of a tensor and a sequence of matrices. Let $\mathcal{A}$ be an $I_1 \times I_2 \times \cdots \times I_N$ tensor. Let $\mathbf{U}^{(n)}$ denote a $J_n \times I_n$ matrix for $n = 1, \ldots, N$. Then

$$\mathcal{B} = \mathcal{A} \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)} \cdots \times_n \mathbf{U}^{(N)} \tag{3.2}$$

```
A = tensor(rand(4,3,2));
U = rand(2,4);
V = rand(3,2);

% Computing A x_1 U x_3 V
B = product(A,U,1);
C = product(B,V,3)
C is a tensor of size 2 x 3 x 3
C.data =
(:,:,1) =
    1.9727    2.0380    2.6528
    1.6460    1.8647    2.4649
(:,:,2) =
    1.9051    2.0078    2.5385
    1.5881    1.8406    2.3523
(:,:,3) =
    0.3289    0.3437    0.4400
    0.2743    0.3148    0.4082

% Computing A x_3 V x_1 U
B = product(A,V,3);
C = product(B,U,1)
C is a tensor of size 2 x 3 x 3
C.data =
(:,:,1) =
    1.9727    2.0380    2.6528
    1.6460    1.8647    2.4649
(:,:,2) =
    1.9051    2.0078    2.5385
    1.5881    1.8406    2.3523
(:,:,3) =
    0.3289    0.3437    0.4400
    0.2743    0.3148    0.4082
```

FIG. 3.1. *Calculating n-mode products.*

```
% Compute the same thing using a cell array
W{1} = U;
W{2} = V;
C = product(A,W,[1 3])
C is a tensor of size 2 x 3 x 3
C.data =
(:,:,1) =
    1.9727    2.0380    2.6528
    1.6460    1.8647    2.4649
(:,:,2) =
    1.9051    2.0078    2.5385
    1.5881    1.8406    2.3523
(:,:,3) =
    0.3289    0.3437    0.4400
    0.2743    0.3148    0.4082
```

FIG. 3.2. *An alternate approach to calculating n-mode products.*

is of size $J_1 \times J_2 \times \cdots \times J_N$. We propose new, alternative notation for this operation that is consistent with the MATLAB command:

$$\mathcal{B} = \mathcal{A} \times \{\mathbf{U}\}.$$

This notation will prove useful in presenting some algorithms.

The following equivalent MATLAB commands can be used to calculate n-mode products with a sequence of matrices.

> `B = product(A,{U1,U2,...,UN}, [1:N])` calculates $\mathcal{B} = \mathcal{A} \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)} \cdots \times_n \mathbf{U}^{(N)}$. Here `Un` is a MATLAB matrix representing $\mathbf{U}^{(n)}$.
>
> `B = product(A,U)` calculates $\mathcal{B} = \mathcal{A} \times \{\mathbf{U}\}$. Here `U = {U1,U2,...,UN}` is a MATLAB cell array and `Un` is as described above.

Another frequently used operation is multiplying by *all but one* of a sequence of matrices:

$$\mathcal{B} = \mathcal{A} \times_1 \mathbf{U}^{(1)} \cdots \times_{n-1} \mathbf{U}^{(n-1)} \times_{n+1} \mathbf{U}^{(n+1)} \cdots \times_N \mathbf{U}^{(N)}.$$

We propose new, alternative notation for this operation that is consistent with the MATLAB command:

$$\mathcal{B} = \mathcal{A} \times_{-n} \{\mathbf{U}\}.$$

This notation will prove useful in presenting some algorithms in §6.

The following MATLAB commands can be used to calculate n-mode products with all but one of a sequence of matrices.

> `B = product(A,U,-n)` calculates $\mathcal{B} = \mathcal{A} \times_{-n} \{\mathbf{U}\}$. Here `U = {U1,U2,...,UN}` is a MATLAB cell array; the nth cell is simply ignored in the computation.

Note that `B = product(A,{U1,...,U4,U6,...,U9},[1:4,6:9])` is equivalent to `B = product(A,U,-5)` where `U={U1,...,U9}`; both calculate $\mathcal{B} = \mathcal{A} \times_{-5} \{\mathbf{U}\}$.

3.2. Multiplying a tensor times a vector. In our opinion, one source of confusion in n-mode multiplication is what to do when multiplying a tensor times a vector due to the introduction of a singleton dimension in mode n. If the singleton dimension is dropped (as is sometimes desired), then the commutativity of the multiplies (3.1) outlined in the previous section no longer holds because the order of the intermediate result changes and $\times_n$ or $\times_m$ applies to the wrong mode.

Although one can usually determine the correct order of the result via the context of the equation, it is impossible to do this automatically in MATLAB in any robust way. Thus, we propose an alternate name and notation in the case when the newly introduced singleton dimension should indeed be dropped.

Let $\mathcal{A}$ be an $I_1 \times I_2 \times \cdots \times I_N$ tensor, and let $\mathbf{b}$ be an I_n-vector. We propose that the *contracted* n-mode product, which drops the nth singleton dimension, be denoted by

$$\mathcal{A} \,\bar{\times}_n\, \mathbf{b}.$$

The result is of size $I_1 \times \cdots \times I_{n-1} \times I_{n+1} \times \cdots \times I_N$. Note that the order of the result is $N-1$, one less than the original tensor. The entries are computed as follows:

$$(\mathcal{A} \,\bar{\times}_n\, \mathbf{u})(i_1, \ldots, i_{n-1}, i_{n+1}, \ldots, i_N) = \sum_{i_n=1}^{I_n} \mathcal{A}(i_1, i_2, \ldots, i_N)\; \mathbf{u}(i_n).$$

The following MATLAB command computes the contracted n-mode product.

> `product(A,u,n,'vec')` computes $\mathcal{A} \,\bar{\times}_n\, \mathbf{u}$.

Observe that $\mathcal{A} \,\bar{\times}_n\, \mathbf{u}$ and $\mathcal{A} \times_n \mathbf{u}^T$ produce identical results except for the order and shape of the results; that is, $\mathcal{A} \,\bar{\times}_n\, \mathbf{u} \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times I_{n+1} \times \cdots \times I_N}$, whereas $\mathcal{A} \times_n \mathbf{u}^T \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times 1 \times I_{n+1} \times \cdots \times I_N}$. See Figure 3.3 for an example.

For the contracted n-mode product, it is no longer true that multiplication is commutative; i.e.,

$$(\mathcal{A} \,\bar{\times}_m\, \mathbf{u}) \;\bar{\times}_n\, \mathbf{v} \neq (\mathcal{A} \,\bar{\times}_n\, \mathbf{v}) \;\bar{\times}_m\, \mathbf{u}.$$

If we assume $m < n$, then

$$\mathcal{A} \,\bar{\times}_m\, \mathbf{u} \,\bar{\times}_n\, \mathbf{v} = (\mathcal{A} \,\bar{\times}_m\, \mathbf{u}) \;\bar{\times}_{n-1}\, \mathbf{v} = (\mathcal{A} \,\bar{\times}_n\, \mathbf{v}) \;\bar{\times}_m\, \mathbf{u}.$$

As before with matrices, it is often useful to calculate the product of a tensor and a sequence of vectors:

$$\mathcal{B} = \mathcal{A} \,\bar{\times}_1\, \mathbf{u}^{(1)} \,\bar{\times}_2\, \mathbf{u}^{(2)} \cdots \,\bar{\times}_N\, \mathbf{u}^{(N)}$$

or

$$\mathcal{B} = \mathcal{A} \,\bar{\times}_1\, \mathbf{u}^{(1)} \cdots \,\bar{\times}_{n-1}\, \mathbf{u}^{(n-1)} \,\bar{\times}_{n+1}\, \mathbf{u}^{(n+1)} \cdots \,\bar{\times}_N\, \mathbf{u}^{(N)}.$$

As in the matrix case, we propose the following alternative notation:

$$\mathcal{B} = \mathcal{A} \,\bar{\times}\, \{\mathbf{u}\}$$

or

$$\mathcal{B} = \mathcal{A} \,\bar{\times}_{-n}\, \{\mathbf{u}\},$$

respectively.

In practice, one must be careful when calculating a sequence of contracted products to perform the multiplications starting with the highest mode and proceed sequentially to the lowest mode. The following MATLAB commands automatically sort the modes in the correct order.

> `b = product(A,u,'vec')` computes $\mathcal{A} \,\bar{\times}\, \{\mathbf{u}\}$ where `u` is a cell array whose nth entry is the vector $\mathbf{u}^{(n)}$.
>
> `b = product(A,u,-n,'vec')` computes $\mathcal{A} \,\bar{\times}_{-n}\, \{\mathbf{u}\}$.

Note that the result of the first calculation is a scalar, and the result of the second is a vector of size I_n.

```
A = tensor(rand(3,4,2));
u = rand(3,1);

% Compute A x_1 u'
B = product(A,u',1)
B is a tensor of size 1 x 4 x 2
B.data =
(:,:,1) =
    0.5058    0.3319    0.1857    0.6210
(:,:,2) =
    0.9385    0.4829    0.5141    0.8288

% Compare to A \bar x_1 u
C = product(A,u,1,'vec')
C is a tensor of size 4 x 2
C.data =
    0.5058    0.9385
    0.3319    0.4829
    0.1857    0.5141
    0.6210    0.8288
```

FIG. 3.3. *Comparison of* $\mathcal{A} \times_n \mathbf{u}^T$ *and* $\mathcal{A} \bar{\times}_n \mathbf{u}$.

3.3. Multiplication of a tensor with another tensor. The last case of tensor multiplication to consider is the product of two tensors. If the tensors have equal dimensions and are of size $I_1 \times I_2 \times \cdots \times I_N$, then their product is given by

$$\langle \mathcal{A}, \mathcal{B} \rangle = \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_N=1}^{I_N} \mathcal{A}(i_1, i_2, \ldots, i_N)\, \mathcal{B}(i_1, i_2, \ldots, i_N).$$

In MATLAB, this is accomplished via the following command.

`product(A,B)` calculates $\langle \mathcal{A}, \mathcal{B} \rangle$; the result is a scalar.

Using the product definition, the Frobenius norm of a tensor is then given by

$$\|\mathcal{A}\|^2 = \langle \mathcal{A}, \mathcal{A} \rangle = \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_N=1}^{I_N} \mathcal{A}(i_1, i_2, \ldots, i_N)^2.$$

In MATLAB, the norm can be calculated as follows.

`norm(A)` calculates $\|\mathcal{A}\|$, the Frobenius norm of a tensor.

Next, suppose the two tensors are different sizes. Let $\mathcal{A}$ be of size $I_1 \times \cdots \times I_M \times J_1 \times \cdots \times J_N$ and let $\mathcal{B}$ be of size $I_1 \times \cdots \times I_M \times K_1 \times \cdots \times K_P$. We can multiply them along the first M modes, and the result is of size $J_1 \times \cdots \times J_N \times K_1 \times \cdots \times K_P$, given by

$$\langle \mathcal{A}, \mathcal{B} \rangle_{\{1,\ldots,M;1,\ldots,M\}} (j_1, \ldots j_N, k_1, \ldots, k_P) = \\ \sum_{i_1=1}^{I_1} \cdots \sum_{i_M=1}^{I_M} \mathcal{A}(i_1, \ldots, i_M, j_1, \ldots, j_N)\, \mathcal{B}(i_1, \ldots, i_M, k_1, \ldots, k_P).$$

Note that the modes to be multiplied are specified in the subscripts that follow the angle brackets. The remaining modes are ordered such that those from $\mathcal{A}$ come before $\mathcal{B}$, which is different from the tensor-matrix product case considered above because the leftover matrix dimension is inserted at I_m and not moved to the end. In MATLAB, the command is as follows.

`product(A,B,[1:M],[1:M])` computes $\langle \mathcal{A}, \mathcal{B} \rangle_{\{1,\ldots,M;1,\ldots,M\}}$.

4. Matricize: Transforming a Tensor into a Matrix. It is often useful to transform a tensor into a matrix such that all of the columns along a certain mode are rearranged to form a matrix. Following Kiers [7], we call this process "matricizing" because matricizing a tensor is analogous to vectorizing a matrix. De Lathauwer et al. [3] call this process "unfolding."

Typically, a tensor is matricized so that all of the columns associated with a particular dimension are aligned. De Lathauwer et al. [3] and Kiers [7] differ on how the columns should be arranged within the matrix; while both agree that the ordering

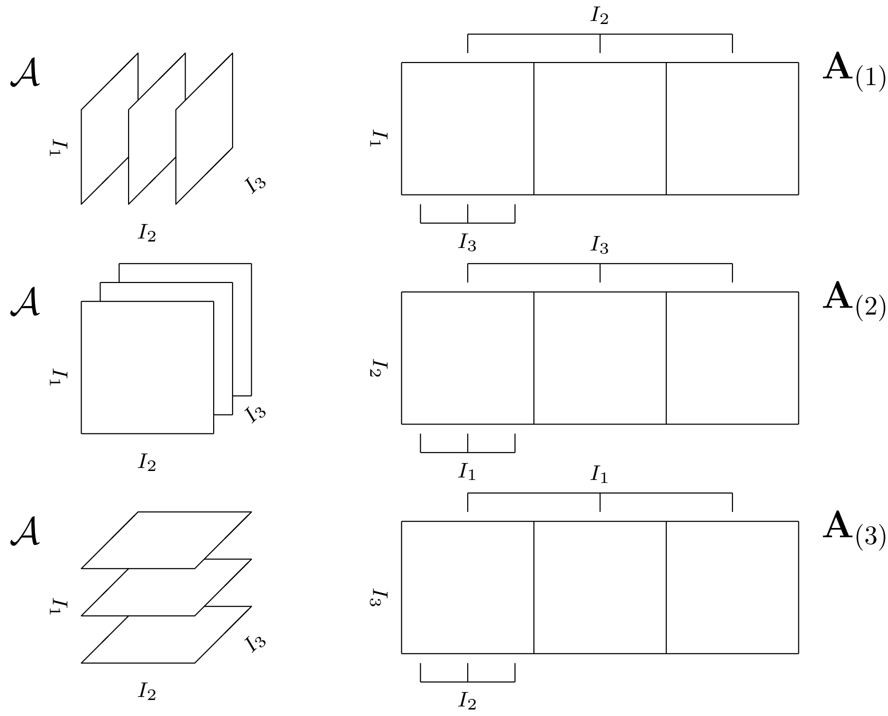

Fig. 4.1. *Matricizing a 3-way tensor according to De Lathauwer et al.*

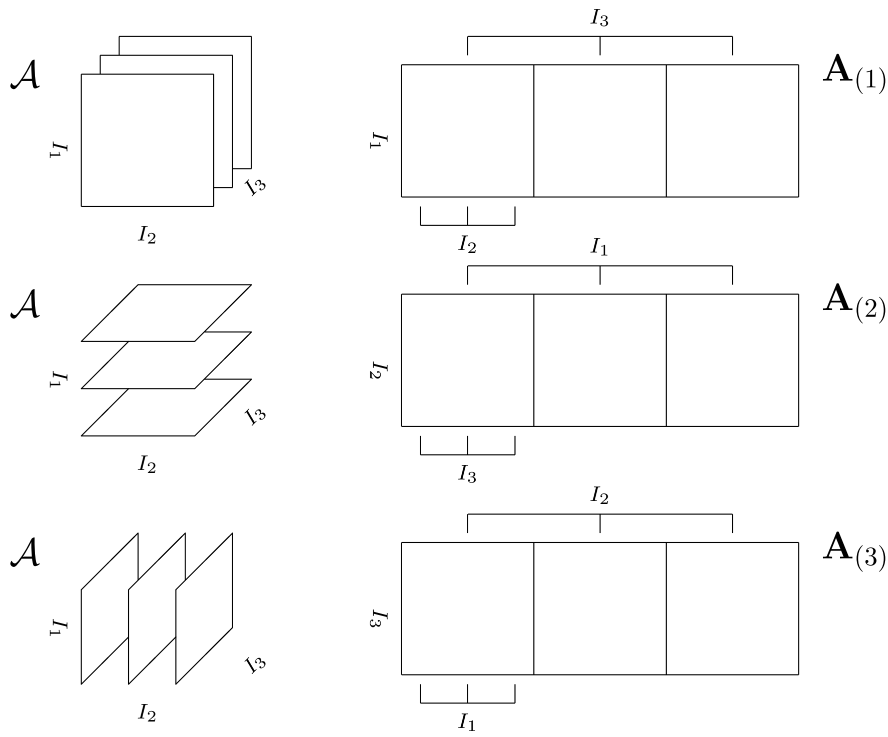

Fig. 4.2. *Matricizing a 3-way tensor according to Kiers.*

should by cyclic, the interpretation of that cyclic ordering is reversed. De Lathauwer et al.'s ordering is shown in Figure 4.1, and Kiers' ordering is shown in Figure 4.2.

In De Lathauwer et al.'s definition [3, Definition 1], $\mathbf{A}_{(n)}$ is of size $I_n \times (I_1 \cdots I_{n-1} I_{n+1} \cdots I_N)$ and contains entry $\mathcal{A}(i_1, \ldots, i_N)$ in position (k, ℓ) where $k = i_n$ and

$$\begin{aligned}
\ell = &(i_{n+1} - 1) I_{n+2} I_{n+3} \cdots I_N I_1 \cdots I_{n-1} \\
&+ (i_{n+2} - 1) I_{n+3} \cdots I_N I_1 \cdots I_{n-1} \\
&+ \cdots \\
&+ (i_N - 1) I_1 I_2 \cdots I_{n-1} \\
&+ (i_1 - 1) I_2 \cdots I_{n-1} \\
&+ \cdots \\
&+ (i_{n-2} - 1) I_{n-1} \\
&+ (i_{n-1} - 1) + 1.
\end{aligned}$$

In Kier's definition [7], $\mathbf{A}_{(n)}$ is of size $I_n \times (I_1 \cdots I_{n-1} I_{n+1} \cdots I_N)$ and contains entry $\mathcal{A}(i_1, \ldots, i_N)$ in position (k, ℓ) where $k = i_n$ and

$$\begin{aligned}
\ell = &(i_{n-1} - 1) I_{n-2} I_{n-3} \cdots I_1 I_N \cdots I_{n+1} \\
&+ (i_{n-2} - 1) I_{n-3} \cdots I_1 I_N \cdots I_{n+1} \\
&+ \cdots \\
&+ (i_2 - 1) I_1 I_N \cdots I_{n+1} \\
&+ (i_1 - 1) I_N \cdots I_{n+1} \\
&+ (i_N - 1) I_{N-1} \cdots I_{n+1} \\
&+ \cdots \\
&+ (i_{n+2} - 1) I_{n+1} \\
&+ (i_{n+1} - 1) + 1.
\end{aligned}$$

To distinguish between these two definitions of matricizations in our mathematical notations, we append the subscript phrase "DDV" or "Kiers"; e.g., $\mathbf{A}_{(n)_{\text{Kiers}}}$. In most cases, the distinction is unnecessary so long as the matricization method used on the right hand side of an equation is the same as the left hand side.

The `matricize` function supports either interpretation as an option but defaults to De Lathauwer et al.'s interpretation. The following MATLAB commands can convert a tensor to a matrix.

> `matricize(T,n)` or `matricize(T,n,'DDV')` computes $\mathbf{T}_{(n)_{\text{DDV}}}$.
>
> `matricize(T,n,'Kiers')` computes $\mathbf{T}_{(n)_{\text{Kiers}}}$.

Figure 4.3 shows two examples of matricizing a tensor.

We can also construct a tensor from a "matricized" tensor by specifying the mode of matricization and original tensor dimensions. The following MATLAB command can convert a matrix to a tensor.

> `tensor(A,n,DIMS,'DDV')` or `tensor(A,n,DIMS,'Kiers')` creates a tensor from a matrix $\mathbf{A}_{(n)}$. The dimensions of the resulting tensor are specified by `DIMS`.

Figure 4.4 shows such a conversion.

Tensors stored in matricized form may be manipulated as matrices, reducing some tensor-matrix operations, such as n-mode multiplication, to matrix-matrix operations. For example, the n-mode product of a tensor $\mathcal{A}$ by a matrix $\mathbf{M}$ may be expressed in the following two ways:

$$\mathcal{B} = \mathcal{A} \times_n \mathbf{M}$$

or, in terms of tensor matricizations,

$$\mathbf{B}_{(n)} = \mathbf{M}\mathbf{A}_{(n)}.$$

Moreover, the series of multiplications in (3.2), when written as a matrix formulation, is given by

$$\begin{aligned}
\mathbf{B}_{(1)_{\text{DDV}}} &= \mathbf{U}^{(1)}\mathbf{A}_{(1)_{\text{DDV}}}(\mathbf{U}^{(2)^T} \otimes \mathbf{U}^{(3)^T} \otimes \ldots \otimes \mathbf{U}^{(N)^T}) \\
&= \mathbf{U}^{(1)}\mathbf{A}_{(1)_{\text{DDV}}}(\mathbf{U}^{(2)} \otimes \mathbf{U}^{(3)} \otimes \ldots \otimes \mathbf{U}^{(N)})^T,
\end{aligned}$$

when using the definition by De Lathauwer et al. [3], or

$$\begin{aligned}
\mathbf{B}_{(1)_{\text{Kiers}}} &= \mathbf{U}^{(1)}\mathbf{A}_{(1)_{\text{Kiers}}}(\mathbf{U}^{(N)^T} \otimes \ldots \otimes \mathbf{U}^{(3)^T} \otimes \mathbf{U}^{(2)^T}) \\
&= \mathbf{U}^{(1)}\mathbf{A}_{(1)_{\text{Kiers}}}(\mathbf{U}^{(N)} \otimes \ldots \otimes \mathbf{U}^{(3)} \otimes \mathbf{U}^{(2)})^T,
\end{aligned}$$

when using the definition by Kiers [7].

5. Decomposed Tensors. As we mentioned previously, we have also created two additional classes to support the representation of tensors in decomposed form, that is, as the sum of rank-1 tensors. A rank-1 tensor is a tensor that can be written as the outer product of vectors, i.e.,

$$\mathcal{A} = \lambda\, \mathbf{u}^{(1)} \circ \mathbf{u}^{(2)} \circ \ldots \circ \mathbf{u}^{(N)},$$

where λ is a scalar and each $\mathbf{u}^{(n)}$ is an I_n-vector, for $n = 1, \ldots, N$. The $\circ$ symbol denotes the outer product; so, in this case, the $(i_1, i_2, \ldots, i_N)$ entry of $\mathcal{A}$ is given by

$$\mathcal{A}(i_1, i_2, \ldots, i_N) = \lambda\, \mathbf{u}^{(1)}_{i_1}\, \mathbf{u}^{(2)}_{i_2} \cdots \mathbf{u}^{(N)}_{i_N},$$

where $\mathbf{u}_i$ denotes the ith entry of vector $\mathbf{u}$. We focus on two different tensor decompositions: CP and Tucker.

5.1. CP tensors. Recall that "CP" is shorthand for CANDECOMP [2] and PARAFAC [5], which are identical models that were developed independently. The CP decomposition is a weighted sum of rank-1 tensors, given by

$$\mathcal{A} = \sum_{k=1}^{K} \lambda_k\, \mathbf{U}^{(1)}_{:k} \circ \mathbf{U}^{(2)}_{:k} \circ \cdots \circ \mathbf{U}^{(N)}_{:k}. \tag{5.1}$$

Here λ is a vector of size K and each $\mathbf{U}^{(m)}$ is a matrix of size $I_n \times K$, for $n = 1, \ldots, N$. Recall that the notation $\mathbf{U}^{(n)}_{:k}$ denotes the kth column of the matrix $\mathbf{U}^{(n)}$.

The following MATLAB command creates a CP tensor.

```
% Let T be a 3 x 4 x 2 tensor
T = tensor(rand(3,4,2))
T is a tensor of size 3 x 4 x 2
T.data =
(:,:,1) =
    0.5390    0.5256    0.3962    0.7578
    0.1358    0.3757    0.0959    0.1490
    0.5949    0.4611    0.8326    0.1960
(:,:,2) =
    0.1808    0.4731    0.8298    0.1779
    0.5269    0.5443    0.5672    0.6324
    0.9991    0.1961    0.1057    0.5835

% The De Lathauwer et al. matricization in mode-1
A1 = matricize(T,2,'DDV')
A1 =
    0.5390    0.1358    0.5949    0.1808    0.5269    0.9991
    0.5256    0.3757    0.4611    0.4731    0.5443    0.1961
    0.3962    0.0959    0.8326    0.8298    0.5672    0.1057
    0.7578    0.1490    0.1960    0.1779    0.6324    0.5835

% The Kiers matricization in mode-1
A2 = matricize(T,2,'Kiers')
A2 =
    0.5390    0.1808    0.1358    0.5269    0.5949    0.9991
    0.5256    0.4731    0.3757    0.5443    0.4611    0.1961
    0.3962    0.8298    0.0959    0.5672    0.8326    0.1057
    0.7578    0.1779    0.1490    0.6324    0.1960    0.5835
```

Fig. 4.3. *Two methods for converting a tensor to a matrix.*

```
% We can convert a matrix into a tensor (inverse matricize)
T = tensor(A1,2,[3,4,2],'DDV')
ans is a tensor of size 3 x 4 x 2
ans.data =
(:,:,1) =
    0.5390    0.5256    0.3962    0.7578
    0.1358    0.3757    0.0959    0.1490
    0.5949    0.4611    0.8326    0.1960
(:,:,2) =
    0.1808    0.4731    0.8298    0.1779
    0.5269    0.5443    0.5672    0.6324
    0.9991    0.1961    0.1057    0.5835
```

Fig. 4.4. *Constructing a tensor from a matrix by reshaping it.*

`T = cp_tensor(lambda,U)` creates a `cp_tensor` object. Here `lambda` is a K-vector and `U` is a cell array whose nth entry is the matrix $U^{(n)}$ with K columns.

A CP tensor can be converted to a dense tensor as follows.

`B = full(A)` converts a `cp_tensor` object to a `tensor` object.

See Figure 5.1 for an example.

Addition and subtraction of CP tensors is handled specially. The λ's and $\mathbf{U}^{(n)}$'s are concatenated. To add or subtract two CP tensors (of the same order and size), use the `+` and `-` signs.

`A + B` computes the sum of two CP tensors.

`A - B` computes the difference of two CP tensors.

An example is shown in Figure 5.2.

To determine the value of K for a CP tensor, execute the following MATLAB command.

`r = length(T.lambda)` returns the "rank" of the tensor `T`.

5.2. Tucker tensors. The *Tucker decomposition* [10], also called a Rank-$(K_1, K_2, \ldots, K_N)$ decomposition [4], is another way of summing decomposed tensors given by

$$\mathcal{A} = \sum_{k_1=1}^{K_1} \sum_{k_2=1}^{K_2} \cdots \sum_{k_N=1}^{K_N} \lambda(k_1, k_2, \ldots, k_N)\, \mathbf{U}^{(1)}_{:k_1} \circ \mathbf{U}^{(2)}_{:k_2} \circ \cdots \circ \mathbf{U}^{(N)}_{:k_N}. \tag{5.2}$$

Here λ is itself a tensor of size $K_1 \times K_2 \times \cdots \times K_N$, and each $\mathbf{U}^{(n)}$ is a matrix of size $I_n \times K_n$, for $n = 1, \ldots, N$. As before, the notation $\mathbf{U}^{(n)}_{:k}$ denotes the kth column of the matrix $\mathbf{U}^{(n)}$. The tensor λ is often called the "core array" or "core tensor."

A Tucker tensor can be created in MATLAB as follows.

`T = tucker_tensor(lambda,U)` where `lambda` is a $K_1 \times K_2 \times \cdots \times K_N$ tensor and `U` is a cell array whose nth entry is a matrix with K_n columns.

Figure 5.3 shows an example.

A Tucker tensor can be converted to a dense tensor as follows.

`B = full(A)` converts a `tucker_tensor` object to a `tensor` object.

5.3. Relations between CP and Tucker. Mathematically, a CP decomposition is a special case of a Tucker decomposition where $K = K_1 = K_2 = \cdots = K_N$ and $\lambda(k_1, k_2, \ldots, k_N)$ is zero unless $k_1 = k_2 = \cdots = k_N$ (i.e., only the diagonal entries of the tensor λ are non-zero). On the other hand, it is possible to express a Tucker decomposition as a CP decomposition where $K = \prod_{n=1}^{N} K_n$.

6. Examples.

We demonstrate the use of the `tensor`, `cp_tensor`, and `tucker_tensor` classes for algorithm development. De Lathauwer et al. have presented higher-order generalizations of the power method and orthogonal iteration in [4], which serve as our examples.

Our first example is the higher-order power method, Algorithm 3.2 from [4], which is a multilinear generalization of the best rank-1 approximation problem for matrices. In Figure 6.1, we have reproduced the algorithm using our notation. Figure 6.2 shows the MATLAB code that implements the algorithm. Figure 6.3 shows sample output using this method.

The higher-order orthogonal iteration is the multilinear generalization of the best rank-R approximation problem for matrices. Algorithm 4.2 in [4] finds the best rank-$(R_1, R_2, \ldots, R_N)$ approximation of a higher-order tensor. We have reproduced the algorithm in Figure 6.4 using our notation, and that MATLAB implementation is in Figure 6.5. Figures 6.6–6.8 show the algorithm applied to the same random tensor from Figure 6.3 for computing different rank-(R_1, R_2, R_3) approximations.

7. Conclusions.

We have described three new MATLAB classes for manipulating dense and factored tensors. These classes extend MATLAB's built-in capabilities for multidimensional arrays in order to facilitate rapid algorithm development.

The `tensor` class simplifies the algorithmic details for implementing numerical methods for higher-order tensors by hiding the underlying matrix operations. It was previously the case that users had to know how to appropriately reshape the tensor into a matrix, execute the desired operation using matrix commands, and then appropriately reshape the result into a tensor. This can be nonintuitive and cumbersome, and we believe using the `tensor` class will be much simpler.

The `tucker_tensor` and `cp_tensor` classes give users an easy way to store and manipulate factored tensors, as well as the ability to convert such tensors into non-factored (or dense) format.

At this stage, our MATLAB implementations are not optimized for performance or memory useage; however, we have striven for consistency and ease-of-use. In the future, we plan to further enhance these classes and add additional functionality.

Over the course of this code development effort, we have relied on published notation, especially from Kiers [7] and De Lathauwer et al. [4]. To address ambiguities that we discovered in the class development process, we have proposed extensions to the existing mathematical notation, particularly in the area of tensor multiplication, that we believe more clearly denote mathematical concepts that were difficult to write succinctly with the existing notation.

We have demonstrated our new notation and MATLAB classes by revisiting the higher-order power method and the higher-order orthogonal iteration method from [4]. In our opinion, the resulting algorithm (using our consolodated notation) and code (using our MATLAB classes) is more easily understood.

REFERENCES

```
A = cp_tensor(5, [2 3 4]', [1 2]', [5 4 3]')
A is a CP tensor of size 3 x 2 x 3
A.lambda =
     5
A.U{1} =
     2
     3
     4
A.U{2} =
     1
     2
A.U{3} =
     5
     4
     3

B = full(A)
B is a tensor of size 3 x 2 x 3
B.data =
(:,:,1) =
    50   100
    75   150
   100   200
(:,:,2) =
    40    80
    60   120
    80   160
(:,:,3) =
    30    60
    45    90
    60   120
```

FIG. 5.1. *An example of a CP tensor.*

```
A = cp_tensor(5, [2 3 4]', [1 2]', [5 4 3]');
B = A + A
B is a CP tensor of size 3 x 2 x 3
B.lambda =
     5
     5
B.U{1} =
     2     2
     3     3
     4     4
B.U{2} =
     1     1
     2     2
B.U{3} =
     5     5
     4     4
     3     3

C = full(B)
C is a tensor of size 3 x 2 x 3
C.data =
(:,:,1) =
   100   200
   150   300
   200   400
(:,:,2) =
    80   160
   120   240
   160   320
(:,:,3) =
    60   120
    90   180
   120   240
```

FIG. 5.2. *Adding two CP tensors.*

```
lambda = tensor(rand(4,3,1),[4 3 1]);
for n = 1 : 3
    U{n} = rand(5,size(lambda,n));
end
A = tucker_tensor(lambda,U)
A is a Tucker tensor of size 5 x 5 x 5
A.lambda =
Tensor of size 4 x 3 x 1
data =
    0.8939    0.2844    0.5828
    0.1991    0.4692    0.4235
    0.2987    0.0648    0.5155
    0.6614    0.9883    0.3340
A.U{1} =
    0.4329    0.6405    0.4611    0.0503
    0.2259    0.2091    0.5678    0.4154
    0.5798    0.3798    0.7942    0.3050
    0.7604    0.7833    0.0592    0.8744
    0.5298    0.6808    0.6029    0.0150
A.U{2} =
    0.7680    0.4983    0.7266
    0.9708    0.2140    0.4120
    0.9901    0.6435    0.7446
    0.7889    0.3200    0.2679
    0.4387    0.9601    0.4399
A.U{3} =
    0.9334
    0.6833
    0.2126
    0.8392
    0.6288

% The size of A
size(A)
ans =
     5     5     5

% The "rank" of A
size(A.lambda)
ans =
     4     3     1
```

Fig. 5.3. *Creating a Tucker tensor.*

Higher-Order Power Method

In: $\mathcal{A}$ of size $I_1 \times I_2 \times \ldots \times I_N$.

Out: $\mathcal{B}$ of size $I_1 \times I_2 \times \ldots \times I_N$, an estimate of the best rank-1 approximation of $\mathcal{A}$.

1. Compute initial values: Let $\mathbf{u}_0^{(n)}$ be the dominant left singular vector of $\mathbf{A}_{(n)}$ for $n = 2, \ldots, N$.
2. For $k = 1, 2, \ldots$ (until converged), do:

   For $n = 1, \ldots, N$, do:

   $\tilde{\mathbf{u}}_{k+1}^{(n)} = \mathcal{A} \,\bar{\times}_{-n} \{\mathbf{u}_k\}$.

   $\lambda_{k+1}^{(n)} = \left\| \tilde{\mathbf{u}}_{k+1}^{(n)} \right\|$

   $\mathbf{u}_{k+1}^{(n)} = \tilde{\mathbf{u}}_{k+1}^{(n)} / \lambda_{k+1}^{(n)}$
3. Let $\lambda = \lambda_K$ and $\{\mathbf{u}\} = \{\mathbf{u}_K\}$ where K is the index of the final result of the iterations.
4. Set $\mathcal{B} = \lambda\, \mathbf{u}^{(1)} \circ \mathbf{u}^{(2)} \circ \ldots \circ \mathbf{u}^{(n)}$.

Fig. 6.1. *Higher-order power method algorithm of [4] using the proposed notation. In this illustration, subscripts denote iteration number.*

```
function B = hopm(A,kmax)

A = tensor(A);
N = order(A);

% Default value
if ~exist('kmax','var')
    kmax = 5;
end

% Compute the dominant left singluar vectors
% of A_(n) (2 <= n <= N)
for n = 2:N
  [u{n}, lambda(n), V] = svds(matricize(A,n), 1);
end

% Iterate until convergence
for k = 1:kmax
  for n = 1:N
    u{n} = product(A, u, -n, 'vec');
    lambda(n) = norm(U{n});
    u{n} = double(u{n}./lambda(n));
  end
end

% Assemble the resulting tensor
B = cp_tensor(lambda(N), u);
```

Fig. 6.2. *MATLAB code for our implementation of the higher-order power method.*

```
T = tensor(rand(3,4,2))
T is a tensor of size 3 x 4 x 2
T.data =
(:,:,1) =
    0.8030    0.9159    0.8735    0.4222
    0.0839    0.6020    0.5134    0.9614
    0.9455    0.2536    0.7327    0.0721
(:,:,2) =
    0.5534    0.3358    0.3567    0.5625
    0.2920    0.6802    0.4983    0.6166
    0.8580    0.0534    0.4344    0.1133

norm(T)
ans =
    2.9089

T1 = hopm(T)
T1 is a CP tensor of size 3 x 4 x 2
T1.lambda =
    2.6206
T1.U{1} =
   -0.6717
   -0.5446
   -0.5023
T1.U{2} =
    0.5431
    0.4700
    0.5444
    0.4333
T1.U{3} =
   -0.8075
   -0.5899

T1f = full(T1)
T1f is a tensor of size 3 x 4 x 2
T1f.data =
(:,:,1) =
    0.7719    0.6680    0.7738    0.6159
    0.6258    0.5416    0.6274    0.4993
    0.5772    0.4996    0.5787    0.4606
(:,:,2) =
    0.5639    0.4880    0.5653    0.4499
    0.4572    0.3956    0.4583    0.3647
    0.4217    0.3649    0.4227    0.3364

norm(T1f)
ans =
    2.6206
```

Fig. 6.3. *Example of the higher-order power method.*

Higher-Order Orthogonal Iteration

In: $\mathcal{A}$ of size $I_1 \times I_2 \times \ldots \times I_N$ and desired rank of output.

Out: $\mathcal{B}$ of size $I_1 \times I_2 \times \ldots \times I_N$, an estimate of the best rank-$(R_1, R_2, \ldots, R_N)$ approximation of $\mathcal{A}$.

1. Compute initial values: Let $\mathbf{U}_0^{(n)} \in \mathbb{R}^{I_n \times R_n}$ be an orthonormal basis for the dominant R_n-dimensional left singular subspace of $\mathbf{A}_{(n)}$ for $n = 2, \ldots, n$.
2. For $k = 1, 2, \ldots$ (until converged), do:

   For $n = 1, \ldots, N$, do:
   $\tilde{\mathcal{U}} = \mathcal{A} \times_{-n} \{\mathbf{U}_k^T\}$
   Let $\mathbf{W}$ of size $I_n \times R_n$ solve:
   $\max \left\| \tilde{\mathcal{U}} \times_n \mathbf{W}^T \right\|$ subject to $\mathbf{W}^T\mathbf{W} = \mathbf{I}$.
   $\mathbf{U}_{k+1}^{(n)} = \mathbf{W}$.
3. Let $\{\mathbf{U}\} = \{\mathbf{U}_K\}$ where K is the index of the final result of the iterations.
4. Set $\lambda = \mathcal{A} \times \{\mathbf{U}^T\}$.
5. Set $\mathcal{B} = \lambda \times \{\mathbf{U}\}$.

Fig. 6.4. *Higher-order orthogonal iteration algorithm of [4] using the proposed notation. In this illustration, subscripts denote iteration number.*

```
function B = hooi(A,R,kmax)

A = tensor(A);
N = order(A);

% Default value
if ~exist('kmax','var')
    kmax = 5;
end

% Compute an orthonormal basis for the dominant
% Rn-dimensional left singular subspace of
% A_(n) (1 <= n <= N).  We store its transpose.
for n = 1:N
  [U, S, V] = svds(matricize(A,n), R(n));
  Ut{n} = U';
end

% Iterate until convergence
for k = 1:kmax
  for n = 1:N
    Utilde = product(A, Ut, -n, 'mat');

    % Maximize norm(Utilde x_n W') wrt W and
    % keeping orthonormality of W
    [W,S,V] = svds(matricize(Utilde, n), R(n));
    Ut{n} = W';
  end
end

% Create the core array
lambda = product(A, Ut, 'mat');

% Create cell array containing U from the cell
% array containing its transpose
for n = 1:N
  U{n} = Ut{n}';
end

% Assemble the resulting tensor
B = tucker_tensor(lambda, U);
```

Fig. 6.5. *MATLAB code for our implementation of the higher-order orthogonal iteration method.*

```
T2 = hooi(T,[1 1 1])
T2 is a Tucker tensor of size 3 x 4 x 2
T2.lambda =
Tensor of size 1 x 1 x 1
data =
    2.6206
T2.U{1} =
    0.6717
    0.5446
    0.5023
T2.U{2} =
   -0.5431
   -0.4700
   -0.5444
   -0.4333
T2.U{3} =
   -0.8075
   -0.5899

T2f = full(T2)
T2f is a tensor of size 3 x 4 x 2
T2f.data =
(:,:,1) =
    0.7719    0.6680    0.7738    0.6159
    0.6258    0.5416    0.6274    0.4993
    0.5772    0.4996    0.5787    0.4606
(:,:,2) =
    0.5639    0.4880    0.5653    0.4499
    0.4572    0.3956    0.4583    0.3647
    0.4217    0.3649    0.4227    0.3364

norm(T2f)
ans =
    2.6206
```

FIG. 6.6. *Example of the higher-order orthogonal iteration for computing the best rank-(1,1,1) tensor.*

```
T3 = hooi(T,[2 2 1])
T3 is a Tucker tensor of size 3 x 4 x 2
T3.lambda =
Tensor of size 2 x 2 x 1
data =
   -2.6206    0.0000
   -0.0000   -1.1233
T3.U{1} =
    0.6718   -0.0186
    0.5445    0.6903
    0.5023   -0.7233
T3.U{2} =
    0.5430   -0.6862
    0.4701    0.3773
    0.5445   -0.1259
    0.4332    0.6090
T3.U{3} =
   -0.8083
   -0.5888

T3f = full(T3)
T3f is a tensor of size 3 x 4 x 2
T3f.data =
(:,:,1) =
    0.7843    0.6625    0.7769    0.6061
    0.1962    0.7786    0.5491    0.8813
    1.0284    0.2523    0.6620    0.0610
(:,:,2) =
    0.5713    0.4826    0.5659    0.4415
    0.1429    0.5671    0.3999    0.6419
    0.7491    0.1838    0.4822    0.0444

norm(T3f)
ans =
    2.8512
```

Fig. 6.7. *Example of the higher-order orthogonal iteration for computing the best rank-(2,2,1) tensor.*

```
T4 = hooi(T,[3 4 2])
T4 is a Tucker tensor of size 3 x 4 x 2
T4.lambda =
Tensor of size 3 x 4 x 2
data =
(:,:,1) =
    2.6201   -0.0075   -0.0338   -0.0029
   -0.0176    1.1198   -0.0118   -0.0006
    0.0142    0.0854   -0.1634   -0.1214
(:,:,2) =
   -0.0126   -0.0529    0.2446   -0.1165
    0.1747    0.0187    0.2369    0.0055
   -0.2249   -0.1523   -0.2324   -0.0311
T4.U{1} =
    0.6774    0.0720   -0.7321
    0.5344   -0.7321    0.4224
    0.5055    0.6774    0.5343
T4.U{2} =
    0.5442    0.6798   -0.2895    0.3974
    0.4774   -0.3606    0.6648    0.4474
    0.5474    0.1200    0.2110   -0.8009
    0.4200   -0.6272   -0.6556    0.0203
T4.U{3} =
    0.8117    0.5841
    0.5841   -0.8117

T4f = full(T4)
T4f is a tensor of size 3 x 4 x 2
T4f.data =
(:,:,1) =
    0.8030    0.9159    0.8735    0.4222
    0.0839    0.6020    0.5134    0.9614
    0.9455    0.2536    0.7327    0.0721
(:,:,2) =
    0.5534    0.3358    0.3567    0.5625
    0.2920    0.6802    0.4983    0.6166
    0.8580    0.0534    0.4344    0.1133

norm(T4f)
ans =
    2.9089
```

FIG. 6.8. *Example of the higher-order orthogonal iteration for computing the best rank-(3,4,2) tensor.*

[1] C. A. Andersson and R. Bro, *The N-way toolbox for MATLAB*, Chemometrics & Intelligent Laboratory Systems, 52 (2000), pp. 1–4. http://www.models.kvl.dk/source/nwaytoolbox/.

[2] J. D. Carrol and J. J. Chang, *Analysis of individual differences in multidimensional scaling via an n-way generalization of 'Eckart-Young' decomposition*, Psychometrika, 35 (1970), pp. 283–319.

[3] L. De Lathauwer, B. De Moor, and J. Vandewalle, *A multilinear singular value decomposition*, SIAM J. Matrix Anal. Appl., 21 (2000), pp. 1253–1278.

[4] ———, *On the best rank-1 and rank-$(R_1, R_2, \ldots, R_N)$ approximation of higher-order tensors*, SIAM J. Matrix Anal. Appl., 21 (2000), pp. 1324–1342.

[5] R. A. Harshman, *Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-modal factor analysis*, UCLA working papers in phonetics, 16 (1970), pp. 1–84.

[6] ———, *An index formulism that generalizes the capabilities of matrix notation and algebra to n-way arrays*, J. Chemometrics, 15 (2001), pp. 689–714.

[7] H. A. L. Kiers, *Towards a standardized notation and terminology in multiway analysis*, J. Chemometrics, 14 (2000), pp. 105–122.

[8] The MathWorks, Inc., *Documentation: MATLAB: Programming: Classes and objects*. http://www.mathworks.com/access/helpdesk/help/techdoc/matlab_prog/ch11_mat.html, 2004.

[9] ———, *Documentation: MATLAB: Programming: Multidimensional arrays*. http://www.mathworks.com/access/helpdesk/help/techdoc/matlab_prog/ch_dat32.html#39663, 2004.

[10] L. R. Tucker, *Some mathematical notes on three-mode factor analysis*, Psychometrika, 31 (1966), pp. 279–311.