SIAM J. MATRIX ANAL. APPL.
Vol. 30, No. 3, pp. 1219–1232

A JACOBI-TYPE METHOD FOR COMPUTING ORTHOGONAL TENSOR DECOMPOSITIONS*

CARLA D. MORAVITZ MARTIN† AND CHARLES F. VAN LOAN‡

Abstract. Suppose $\mathcal{A} = (a_{ijk}) \in \mathbb{R}^{n\times n\times n}$ is a three-way array or third-order tensor. Many of the powerful tools of linear algebra such as the singular value decomposition (SVD) do not, unfortunately, extend in a straightforward way to tensors of order three or higher. In the two-dimensional case, the SVD is particularly illuminating, since it reduces a matrix to diagonal form. Although it is not possible in general to diagonalize a tensor (i.e., $a_{ijk} = 0$ unless $i = j = k$), our goal is to "condense" a tensor in fewer nonzero entries using orthogonal transformations. We propose an algorithm for tensors of the form $\mathcal{A} \in \mathbb{R}^{n\times n\times n}$ that is an extension of the Jacobi SVD algorithm for matrices. The resulting tensor decomposition reduces $\mathcal{A}$ to a form such that the quantity $\sum_{i=1}^{n} a_{iii}^2$ or $\sum_{i=1}^{n} a_{iii}$ is maximized.

Key words. multilinear algebra, tensor decomposition, singular value decomposition, multidimensional arrays

AMS subject classifications. 15A69, 65F30

DOI. 10.1137/060655924

1. Introduction.

The SVD of a matrix gives us important information about a matrix such as its rank, an orthonormal basis for the column or row space, and reduction to diagonal form. In applications, especially those involving multiway data analysis, information about the rank and reduction of tensors to have fewer nonzero entries are useful concepts to try to extend to higher dimensions.

Suppose $\mathcal{A} = (a_{ijk}) \in \mathbb{R}^{n\times n\times n}$ is a three-way array or third-order tensor. This paper is about computing a tensor decomposition of $\mathcal{A}$ such that $\mathcal{A}$ is written as a linear combination of rank-1 tensors of the form

$$w \otimes v \otimes u, \tag{1.1}$$

where $u, v, w \in \mathbb{R}^n$ and "$\otimes$" denotes the Kronecker product. The rank-1 tensor in (1.1) can also be denoted using the tensor outer product notation. In particular, (1.1) is equivalent to a vectorization of $u \circ v \circ w$.

The contribution of this paper is a higher-order generalization of the Jacobi SVD algorithm for matrices [14, p. 457] that works by solving small subproblems where $n = 2$. In the higher-order generalization, we find a tensor decomposition of the form

$$a = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} \sigma_{ijk}(w_k \otimes v_j \otimes u_i), \tag{1.2}$$

where u_i, v_j, and w_k are the ith, jth, and kth columns of orthogonal matrices $U, V, W \in \mathbb{R}^{n\times n}$, respectively, and σ_{ijk} is the (i, j, k)th element of a tensor $\Sigma \in$

*Received by the editors March 31, 2006; accepted for publication (in revised form) by P. Comon December 17, 2007; published electronically September 25, 2008. This work was supported by NSF grant CCR-9901988.

http://www.siam.org/journals/simax/30-3/65592.html

†Department of Mathematics and Statistics, James Madison University, Harrisonburg, VA 22807 (carlam@math.jmu.edu).

‡Department of Computer Science, Cornell University, Ithaca, NY 14853-7510 (cv@cs.cornell.edu).

$\mathbb{R}^{n\times n\times n}$. The orthogonal matrices U, V, W are chosen to maximize either the sums of squares of the diagonals ($\sum_{i=1}^{n} \sigma_{iii}^2$) or the "trace" ($\sum_{i=1}^{n} \sigma_{iii}$).

Several Jacobi-like procedures have been implemented in the area of tensor decompositions already. In the case of symmetric tensors, there exist Jacobi algorithms to compute tensor decompositions of real or complex tensors [6, 7, 8, 12]. In particular, [12] shows that the solution to the Jacobi subproblem for symmetric tensors has an SVD solution of a particular symmetric matrix. We use a similar method when we show how to maximize the trace of a tensor. A Jacobi-type algorithm in [24] is used to simultaneously diagonalize positive definite Hermitian matrices using nonorthogonal transformations. This is relevant to tensors since [13] shows that a nonorthogonal tensor decomposition can be rewritten in terms of a simultaneous diagonalization problem for matrices. Furthermore, [13] proposes a Jacobi approach to simultaneous diagonalization (see also [4]) and shows that maximizing the sums of squares of the diagonals of a $2 \times 2 \times 2$ tensor using orthogonal transformations is equivalent to finding the roots of a polynomial of degree eight [13].

Tensor decompositions are used in many applications to help explain interactions among multiway data. These applications include chemometrics [3, 27], psychometrics [22], computer image and human motion recognition [29, 30], signal processing [25, 26], and many other areas using multiway data analyses [9]. Sometimes the tensor decompositions have a minimal number of terms in the linear combination, therefore "condensing" $\mathcal{A}$ into fewer nonzero entries so that interactions can be better explained.

A large amount of work has already been devoted to creating algorithms to compute orthogonal tensor decompositions. The most widely used algorithm is TUCKER3 originally proposed by Tucker [28]. Many improvements have been made to the algorithm since its original introduction, and the current version has been implemented in a MATLAB toolbox [1, 2]. A greedy algorithm to compute an orthogonal tensor decomposition has been proposed by [21], and [10] uses TUCKER3 to describe a higher-order generalization of the SVD for tensors. Several methods have also been developed to compute a "compressed" third-order orthogonal tensor decomposition, i.e., maximum variance of squares [16], maximum sums of squares of the diagonals of each face of a tensor [22], and maximum sums of squares of the diagonals of a third-order tensor [18].

A special case of TUCKER3 is the CANDECOMP-PARAFAC algorithm simultaneously proposed by [5] and [15]. Algorithms have also been developed to compute the nearest rank-1 tensor to a given tensor $\mathcal{A}$ (see [11, 20, 21, 32]). Furthermore, since the CANDECOMP-PARAFAC representation is equivalent to simultaneously diagonalizing a set of matrices, there are a number of recent algorithms related to simultaneous diagonalization (see [13] and the references therein).

Our presentation is organized as follows. First, we describe some matrix tools necessary to describe tensors and tensor decompositions in section 2. In section 3 we describe different ways to represent tensors. In sections 4 and 5 we describe the higher-order generalization of the Jacobi SVD algorithm—first for tensors $\mathcal{A} \in \mathbb{R}^{2\times 2\times 2}$ and then for general tensors $\mathcal{A} \in \mathbb{R}^{n\times n\times n}$. We examine the algorithm cost in section 6, and describe a block version and an $\ell \times m \times n$ version in sections 7 and 8, respectively. Extending the algorithm to order-p tensors is discussed briefly in section 9. Finally, in section 10 we examine the performance of the algorithm.

2. Some properties of the Kronecker product. We review a few essential facts about the Kronecker product which are found in [31]. Computations that involve the Kronecker product require an understanding of the `vec` and `reshape` operators.

If $B \in \mathbb{R}^{m\times n}$, then $\texttt{vec}(B) \in \mathbb{R}^{mn}$ is the vector formed by "stacking" the columns of B.

The vec operator can be used to convert between matrix-vector and matrix-matrix products. For example, let $F \in \mathbb{R}^{m\times m}, G \in \mathbb{R}^{n\times n}, X \in \mathbb{R}^{n\times m}$. Then

$$Y = GXF^T \qquad \Longleftrightarrow \qquad \texttt{vec}(Y) = (F \otimes G)\texttt{vec}(X). \tag{2.1}$$

Another important property of vec involves outer products. For $x \in \mathbb{R}^m, y \in \mathbb{R}^n$, $\texttt{vec}(xy^T) = y \otimes x$.

The vec operator can be used in combination with the outer product to write certain matrix factorizations as vectors. For example, if $A \in \mathbb{R}^{m\times n}$ and $A = UBV^T$, where $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$, then

$$A = \sum_{i=1}^{m}\sum_{j=1}^{n} b_{ij}u_iv_j^T \qquad \Longleftrightarrow \qquad \texttt{vec}(A) = \sum_{i=1}^{m}\sum_{j=1}^{n} b_{ij}(v_j \otimes u_i), \tag{2.2}$$

where u_i, v_j are the ith and jth columns of U and V, respectively.

We can also write $\texttt{vec}(A)$ as a matrix-vector product. If $a = \texttt{vec}(A)$ and $b = \texttt{vec}(B)$, then by (2.1)

$$a = (V \otimes U) \cdot b. \tag{2.3}$$

The reshape operator is a more general way of rearranging the entries in a matrix (it is also a MATLAB function). If $b \in \mathbb{R}^{mn}$, then $\texttt{reshape}(b, m, n)$ creates an $m \times n$ matrix from b. This operator is useful when converting between $B \otimes C$ and $C \otimes B$. If the vec *permutation matrix* $\Pi_{n,mn} \in \mathbb{R}^{mn\times mn}$ is defined by

$$\Pi_{n,mn} = \begin{bmatrix} I_{mn}(1:m:mn,:) \\ I_{mn}(2:m:mn,:) \\ \vdots \\ I_{mn}(m:m:mn,:) \end{bmatrix},$$

then it can be shown that (see [31])

$$\Pi_{n,mn}^T(B \otimes C)\Pi_{n,mn} = C \otimes B.$$

3. Tensor decompositions. We use the accepted notation where a third-order tensor is indexed by three indices and can be represented as a "cube" of data [19]. While the cube orientation is not unique, here we say that the kth index indicates the face of the cube. See Figure 1 for an illustration when $n = 2$.

An $n \times n \times n$ tensor $\mathcal{A}$ is also a three-way array where the kth face is represented using MATLAB notation as $A(:,:,k)$. The entries $\texttt{vec}(\mathcal{A})$ can also be rearranged to correspond to viewing the cube in different ways (see Figure 2 for an illustration when $n = 2$). Viewing the tensor cube in different orientations corresponds to a rearrangement of the elements of $\mathcal{A}$. This can be better described by a permutation of the elements of $\texttt{vec}(\mathcal{A})$ by a reshape operation. For example, if $\mathcal{A} \in \mathbb{R}^{n\times n\times n}$ and $a = \texttt{vec}(\mathcal{A})$, then the different cuts can be represented in vector form as

$$\begin{aligned} a_1 &= \texttt{vec}(\texttt{reshape}(\Pi_{n,n^3}^T \cdot a, n, n^2)) && \text{(top-bottom)}, \\ a_2 &= \texttt{vec}(\texttt{reshape}(\Pi_{n^2,n^3}^T \cdot a, n, n^2)) && \text{(left-right)}, \\ a_3 &= \texttt{vec}(\texttt{reshape}(a, n, n^2)) && \text{(front-back)}, \end{aligned} \tag{3.1}$$

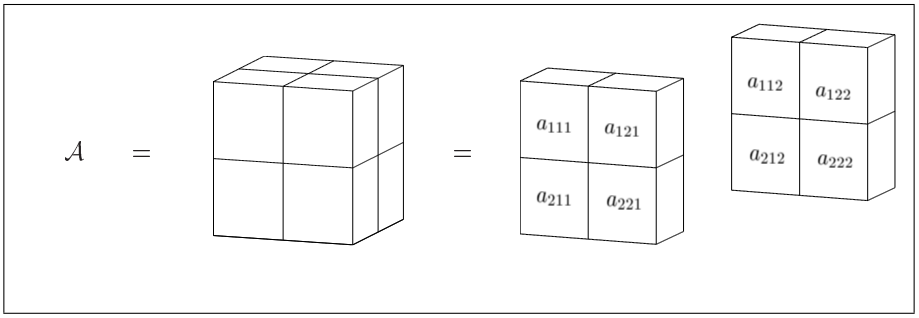

FIG. 1. *Illustration of a third-order tensor as a cube of data when* $n = 2$.

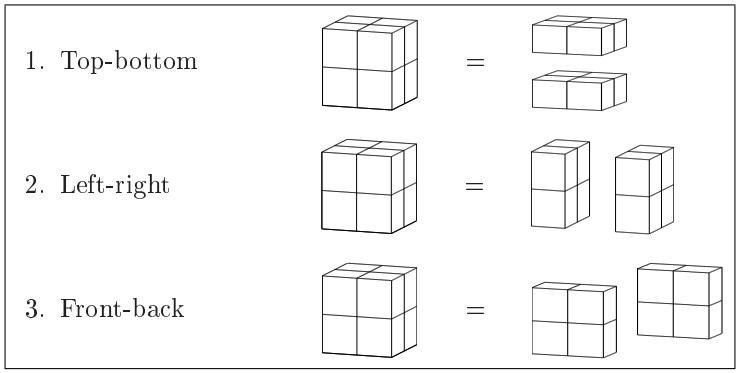

FIG. 2. *Three ways to cut a cube of data for a third-order tensor.*

where $a_3 = \texttt{vec}(\mathcal{A})$. Viewing the different faces of the cube side by side as matrices also corresponds to *unfolding matrices* [10] or *matricizations* [19].

The vec operator is used in the same way on tensors as on matrices. If $\mathcal{A} \in \mathbb{R}^{n\times n\times n}$, then $\texttt{vec}(\mathcal{A}) \in \mathbb{R}^{n^3}$ is the vector formed by stacking the column vectors $\texttt{vec}(A(:,:,i))$ for $i = 1,\ldots,n$.

Hence, given $\mathcal{A} \in \mathbb{R}^{n\times n\times n}$, the basic goal of this work is to find orthogonal matrices $U, V, W \in \mathbb{R}^{n\times n}$ such that

$$a \equiv \texttt{vec}(\mathcal{A}) = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} \sigma_{ijk}(w_k \otimes v_j \otimes u_i), \tag{3.2}$$

and either of the two quantities $\sum_{i=1}^{n} \sigma_{iii}^2$ or $\sum_{i=1}^{n} \sigma_{iii}$ are maximized. In (3.2), u_i, v_j, and w_k are the ith, jth, and kth columns of U, V, and W, respectively.

By (2.2) and (2.3), (3.2) can also be written as the matrix-vector product

$$a = (W \otimes V \otimes U) \cdot \sigma, \tag{3.3}$$

where $\sigma = \texttt{vec}(\Sigma)$ and $\Sigma = (\sigma_{ijk}) \in \mathbb{R}^{n\times n\times n}$. The representation in (3.3) will be used to describe our algorithm in the sections that follow.

One may ask whether it would be better to choose U, V, and W in (3.2) such that the tensor $\Sigma = (\sigma_{ijk})$ is "diagonal" (i.e., $\sigma_{ijk} = 0$ unless $i = j = k$). In this scenario, (3.2) reduces exactly to the matrix SVD in two dimensions. However, diagonality from orthogonal transformations is possible only for tensors of order three or higher

in special cases. In general it is not possible to find orthogonal matrices U, V, W such that the tensor Σ is diagonal [10].

4. `Jacobi-Compress` algorithm for $n \times n \times n$ tensors. We now describe `Jacobi-Compress`, the higher-order generalization of the Jacobi SVD algorithm in [14, p. 457]. The overall idea is to compute tensor decompositions of the form (3.2) of $2 \times 2 \times 2$ subtensors. Here, we explain the general procedure when $\mathcal{A} \in \mathbb{R}^{n \times n \times n}$. The next section details how to solve the $2 \times 2 \times 2$ subproblem.

In the spirit of the Jacobi SVD algorithm, the first step is to choose a (p, q) pair and form the corresponding $2 \times 2 \times 2$ subtensor $\tilde{A}$ given by

$$\tilde{A}(:,:,1) = \begin{bmatrix} a_{ppp} & a_{pqp} \\ a_{qpp} & a_{qqp} \end{bmatrix}, \qquad \tilde{A}(:,:,2) = \begin{bmatrix} a_{ppq} & a_{pqq} \\ a_{qpq} & a_{qqq} \end{bmatrix}.$$

Using the representation in (3.3), suppose we have orthogonal matrices $\tilde{U}, \tilde{V}, \tilde{W} \in \mathbb{R}^{2\times 2}$ such that

$$\begin{bmatrix} \sigma_{ppp} \\ \sigma_{qpp} \\ \sigma_{pqp} \\ \sigma_{qqp} \\ \sigma_{ppq} \\ \sigma_{qpq} \\ \sigma_{pqq} \\ \sigma_{qqq} \end{bmatrix} = (\tilde{W}^T \otimes \tilde{V}^T \otimes \tilde{U}^T) \begin{bmatrix} a_{ppp} \\ a_{qpp} \\ a_{pqp} \\ a_{qqp} \\ a_{ppq} \\ a_{qpq} \\ a_{pqq} \\ a_{qqq} \end{bmatrix}, \tag{4.1}$$

and either $(\sigma_{ppp}^2 + \sigma_{qqq}^2)$ or $(\sigma_{ppp} + \sigma_{qqq})$ is maximized. To update $a = \mathtt{vec}(\mathcal{A}) \in \mathbb{R}^{n^3}$, set U to be the $n \times n$ identity matrix, except that $I(p:q, p:q) = \tilde{U}$ and analogously for V and W. Then perform the update

$$\sigma \leftarrow (W^T \otimes V^T \otimes U^T) \cdot a, \tag{4.2}$$

where $a = [\mathtt{vec}(\tilde{A}(:,:,1)); \mathtt{vec}(\tilde{A}(:,:,2))]$. In practice, significant savings are achieved by observing that the only elements of a that change are those with a p or q in the index; i.e., the pth and qth front-back faces, the pth and qth side faces, and the pth and qth top-bottom faces. This avoids the actual computation of the above Kronecker product.

Rather than just repeating *sweeps* through all the possible (p, q) pairs as in the Jacobi SVD algorithm, we alternate the view or orientation of the cube (Figure 2) at each sweep. Therefore, one *iteration* of `Jacobi-Compress` includes three sweeps: one sweep for each orientation. Changing the orientation simply involves a `reshape` operation defined in (3.1). `Jacobi-Compress` is complete when an iteration does not significantly change (within some specified tolerance) the sums of squares of the diagonals or the trace of the tensor.

5. The $2 \times 2 \times 2$ subproblem. Suppose $\mathcal{A} \in \mathbb{R}^{2\times 2\times 2}$ and $a = \mathtt{vec}(\mathcal{A})$. We now show how to solve (4.1). That is, the goal is to find 2×2 orthogonal matrices $\tilde{U}$, $\tilde{V}$, and $\tilde{W}$ such that

$$\sigma = (\tilde{W}^T \otimes \tilde{V}^T \otimes \tilde{U}^T)a$$

has maximum $(\sigma_{111}^2 + \sigma_{222}^2)$ or $(\sigma_{111} + \sigma_{222})$.

The idea behind solving the subproblem is an iterative approach that differs based on whether we are maximizing the trace or sums of squares. However, both approaches involve taking an SVD of a specific 2×2 matrix.

5.1. Maximizing the trace. Maximizing the trace involves holding one variable constant while varying the others. For example, suppose we have performed the following three steps:

$$\begin{aligned} \sigma_1 &\leftarrow (I \otimes \tilde{V}_1^T \otimes \tilde{U}_1^T)a, \\ \sigma_2 &\leftarrow (\tilde{W}_1^T \otimes I \otimes \tilde{U}_2^T)\sigma_1, \\ \sigma_3 &\leftarrow (\tilde{W}_2^T \otimes \tilde{V}_2^T \otimes I)\sigma_2. \end{aligned} \tag{5.1}$$

Then a tensor decomposition of a has been computed since

$$\begin{aligned} \sigma_3 &= (\tilde{W}_2^T \otimes \tilde{V}_2^T \otimes I)(\tilde{W}_1^T \otimes I \otimes \tilde{U}_2^T)(I \otimes \tilde{V}_1^T \otimes \tilde{U}_1^T)a \\ &= (\tilde{W}_2^T\tilde{W}_1^T \otimes \tilde{V}_2^T\tilde{V}_1^T \otimes \tilde{U}_2^T\tilde{U}_1^T)a \\ &= (\tilde{W}^T \otimes \tilde{V}^T \otimes \tilde{\tilde{U}}^T)a, \end{aligned}$$

where $\tilde{U} = \tilde{U}_1\tilde{U}_2$, $\tilde{V} = \tilde{V}_1\tilde{V}_2$, and $\tilde{W} = \tilde{W}_1\tilde{W}_2$.

The steps in (5.1) illustrate the central idea behind our algorithm. After each step, the trace of σ is maximized. The updates (5.1) are all algorithmically equivalent since

$$\Pi_{4,8}^T(\tilde{W}^T \otimes I \otimes \tilde{U}^T)\Pi_{4,8} = (I \otimes \tilde{U}^T \otimes \tilde{W}^T),$$

$$\Pi_{2,8}^T(\tilde{W}^T \otimes \tilde{V}^T \otimes I)\Pi_{2,8} = (I \otimes \tilde{W}^T \otimes \tilde{V}^T).$$

Therefore it suffices to describe how to find $\tilde{U}, \tilde{V}$ such that

$$\sigma = (I \otimes \tilde{V}^T \otimes \tilde{U}^T)a. \tag{5.2}$$

A remark about (5.2) is necessary. Equation (5.2) can be rewritten in terms of matrices as

$$\begin{aligned} \begin{bmatrix} \sigma_{111} & \sigma_{121} \\ \sigma_{211} & \sigma_{221} \end{bmatrix} &= \tilde{U}^T \begin{bmatrix} a_{111} & a_{121} \\ a_{211} & a_{221} \end{bmatrix} \tilde{V}, \\ \begin{bmatrix} \sigma_{112} & \sigma_{122} \\ \sigma_{212} & \sigma_{222} \end{bmatrix} &= \tilde{U}^T \begin{bmatrix} a_{112} & a_{122} \\ a_{212} & a_{222} \end{bmatrix} \tilde{V}. \end{aligned} \tag{5.3}$$

However, we emphasize that solving (5.3) is not a joint SVD problem. The joint SVD problem [17, 23] uses a least squares approach to find matrices $\tilde{U}, \tilde{V}$ that diagonalize the left two matrices above. A Jacobi-like algorithm was proposed in [23] that reduces the problem to finding joint SVDs of 2×2 matrices by maximizing the sums of squares of the diagonals. In [23] it is shown that the joint SVD problem with 2×2 matrices has an explicit solution, namely, that maximizing the sums of squares is equivalent to maximizing the trace. In our case, we are maximizing the sums of squares or trace of the *tensor* diagonals. The two problems are not equivalent in our case.

The solution depends on whether $\tilde{U}$ and $\tilde{V}$ are both rotation or reflection matrices or whether one is a rotation matrix and one is a reflection matrix.

First, suppose $\tilde{U}_1$ and $\tilde{V}_1$ are the rotation matrices

$$\tilde{U}_1 = \begin{bmatrix} c_1 & s_1 \\ -s_1 & c_1 \end{bmatrix}, \qquad \tilde{V}_1 = \begin{bmatrix} c_2 & s_2 \\ -s_2 & c_2 \end{bmatrix}. \tag{5.4}$$

Then

$$\begin{aligned}\operatorname{tr}(\sigma) &= c_1c_2(a_{111}+a_{222}) + s_1c_2(-a_{211}+a_{122}) \\ &\quad + c_1s_2(-a_{121}+a_{212}) + s_1s_2(a_{221}+a_{112}).\end{aligned} \tag{5.5}$$

If

$$B_1 = \begin{bmatrix} a_{111}+a_{222} & a_{121}-a_{212} \\ a_{211}-a_{122} & a_{221}+a_{112} \end{bmatrix}, \tag{5.6}$$

then the $(1,1)$-entry of $\tilde{U}_1^T B_1 \tilde{V}_1$ is exactly (5.5). Therefore, maximizing (5.5) is equivalent to finding the SVD of B_1, since the largest singular value is the largest $(1,1)$-entry possible. The same result can be derived if $\tilde{U}_1$ and $\tilde{V}_1$ are both reflection matrices. In this scenario, we must ensure that the SVD consists of either both rotation matrices or both reflection matrices. Using an SVD solution to the trace problem is similar to a method used in [23] to compute the joint SVD.

Now, suppose that $\tilde{U}_2$ is a rotation matrix and $\tilde{V}_2$ is a reflection matrix:

$$\tilde{U}_2 = \begin{bmatrix} c_1 & s_1 \\ -s_1 & c_1 \end{bmatrix}, \qquad \tilde{V}_2 = \begin{bmatrix} c_2 & s_2 \\ s_2 & -c_2 \end{bmatrix}. \tag{5.7}$$

Then

$$\begin{aligned}\operatorname{tr}(\sigma) &= c_1c_2(a_{111}-a_{222}) + s_1c_2(a_{211}+a_{122}) \\ &\quad + c_1s_2(-a_{121}-a_{212}) + s_1s_2(a_{112}-a_{221}).\end{aligned} \tag{5.8}$$

If

$$B_2 = \begin{bmatrix} a_{111}-a_{222} & a_{121}+a_{212} \\ a_{211}+a_{122} & a_{221}-a_{112} \end{bmatrix}, \tag{5.9}$$

then the $(1,1)$-entry of $\tilde{U}_2^T B_2^T \tilde{V}_2$ is exactly (5.8), and therefore maximizing (5.8) is equivalent to taking the SVD of B_2. The same result holds if $\tilde{U}_2$ is a reflection matrix and $\tilde{V}_2$ is a rotation matrix.

Comparing the (1,1)-entries of B_1 and B_2 determines how $\tilde{U}$ and $\tilde{V}$ are chosen. For example, if

$$a_{111}+a_{222} > a_{111}-a_{222},$$

then we use B_1 and the SVD should involve either both rotation matrices or both reflection matrices. On the other hand, if

$$a_{111}+a_{222} < a_{111}-a_{222},$$

then we compute the SVD of B_2 and the result should involve one rotation matrix and one reflection matrix.

5.2. Maximizing the sums of squares. Setting up the problem to maximizing the sums of squares in a similar way to section 5.1 involves solving a nonlinear optimization problem that does not have a straightforward explicit solution. Here, we instead hold two variables constant and vary the third, which results in an explicit solution involving the SVD. The basic problem requires solving

$$\begin{aligned}\sigma_1 &\leftarrow (I \otimes I \otimes \tilde{U}^T)a, \\ \sigma_2 &\leftarrow (I \otimes \tilde{V}^T \otimes I)\sigma_1, \\ \sigma_3 &\leftarrow (\tilde{W}^T \otimes I \otimes I)\sigma_2.\end{aligned} \tag{5.10}$$

Similar to section 5.1, each of the steps above are equivalent since they involve permuting the tensor elements at each step. We therefore describe how to find $\tilde{U}$ such that

$$\sigma = (I \otimes I \otimes \tilde{U})\, a = \begin{bmatrix} \tilde{U} & & & \\ & \tilde{U} & & \\ & & \tilde{U} & \\ & & & \tilde{U} \end{bmatrix} \begin{bmatrix} a_{111} \\ a_{211} \\ a_{121} \\ a_{221} \\ a_{112} \\ a_{212} \\ a_{122} \\ a_{222} \end{bmatrix}. \tag{5.11}$$

From (5.11), it suffices to find orthogonal $\tilde{U}$ that maximizes the sums of squares of the diagonals of the matrix $\tilde{U}A$, where

$$A = \begin{bmatrix} a_{111} & a_{122} \\ a_{211} & a_{222} \end{bmatrix}. \tag{5.12}$$

Suppose the SVD of A is given by

$$A = U \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix} \begin{bmatrix} c & s \\ -s & c \end{bmatrix},$$

where (c, s) is a cosine/sine pair. Indeed, if we have a 2×2 orthogonal matrix Z such that the sums of squares of the diagonals of

$$Z \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix} \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \tag{5.13}$$

are maximized, then we set $\tilde{U} = ZU^T$. We now explain how to obtain Z.

Without loss of generality, assume that Z is the rotation matrix

$$Z = \begin{bmatrix} \tilde{c} & \tilde{s} \\ -\tilde{s} & \tilde{c} \end{bmatrix}.$$

From (5.13) we are trying to find the pair $(\tilde{c}, \tilde{s})$ that maximizes

$$(\sigma_1 \tilde{c} c - \sigma_2 \tilde{s} s)^2 + (-\sigma_1 \tilde{s} s + \sigma_2 \tilde{c} c)^2. \tag{5.14}$$

After some simplification, (5.14) is equivalent to

$$(\sigma_1^2 + \sigma_2^2) - \left\| M \begin{bmatrix} \tilde{c} \\ \tilde{s} \end{bmatrix} \right\|_2^2, \tag{5.15}$$

where

$$M = \begin{bmatrix} \sigma_2 s & \sigma_1 c \\ \sigma_1 s & \sigma_2 c \end{bmatrix}. \tag{5.16}$$

Since

$$\sigma_1^2 + \sigma_2^2 = \|M\|_F^2 = \sigma_{max}(M)^2 + \sigma_{min}(M)^2, \tag{5.17}$$

where $\sigma_{max}(M)$ and $\sigma_{min}(M)$ are the largest and smallest singular values of M, respectively, we choose $[\tilde{c}, \tilde{s}]^T$ to be the right singular vector of M associated to the smallest singular value. This also means that the maximum sums of squares of the diagonals of A are equal to $\sigma_{max}(M)^2$. The same result is obtained if Z is chosen to be a reflection matrix.

$[\tilde{U}, \tilde{V}, \tilde{W}, \sigma] =$ **Solve2-by-2-by-2**$(\mathcal{A})$

$a_0 = \texttt{vec}(\mathcal{A})$
for $j = 1, 2, 3$ (each dimension) **do**

% Solves $\sigma_j = (I \otimes Y_j^T \otimes X_j^T)\sigma_{j-1}$
$B \leftarrow \texttt{reshape}(\sigma_{j-1}, 2, 4)$
$\Sigma_1 \leftarrow B(1:2, 1:2)$
$\Sigma_2 \leftarrow B(1:2, 3:4)$

if sums of squares are maximized **then**

$A \leftarrow [\Sigma_1(:,1)\ \ \Sigma_2(:,2)]$
$[U, S, V] =$ **svd**(A)

$M \leftarrow \begin{bmatrix} \sigma_2 s & \sigma_1 c \\ \sigma_1 s & \sigma_2 c \end{bmatrix}$ (See (5.16))
$[\bar{U}, \bar{S}, \bar{V}] =$ **svd**(M)

$Z \leftarrow \begin{bmatrix} \bar{v}_{12} & \bar{v}_{22} \\ -\bar{v}_{22} & \bar{v}_{12} \end{bmatrix}$
$X_j \leftarrow ZU^T; \quad Y_j \leftarrow \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix};$
$\sigma_j \leftarrow \texttt{vec}([X_j\Sigma_1 \mid X_j\Sigma_2])$

else if trace is maximized **then**
if $a_{111} + a_{222} \geq a_{111} - a_{222}$ **then**

$B \leftarrow \begin{bmatrix} a_{111} + a_{222} & a_{121} - a_{212} \\ a_{211} - a_{122} & a_{221} + a_{112} \end{bmatrix}$
else
$B \leftarrow \begin{bmatrix} a_{111} - a_{222} & a_{121} + a_{212} \\ a_{211} + a_{122} & a_{221} - a_{112} \end{bmatrix}$
end if
$[X_j, S, Y_j] =$ **svd**(B)
$\sigma_j \leftarrow \texttt{vec}\left([\ X_j^T\Sigma_1 Y_j \mid X_j\Sigma_2 Y_j\ ]\right)$

end if
end for
$\sigma \leftarrow \sigma_3; \quad \tilde{W} \leftarrow X_2 Y_3; \quad \tilde{V} \leftarrow Y_1 X_3; \quad \tilde{U} \leftarrow X_1 Y_2;$

FIG. 3. *Algorithm to solve the* $2 \times 2 \times 2$ *subproblem.*

6. Algorithm cost. Figures 3 and 4 contain `Jacobi-Compress` and its corresponding $2 \times 2 \times 2$ subproblem in pseudo-MATLAB. Actual MATLAB code can be found at [33].

To assess the amount of work, note that each *iteration* of `Jacobi-Compress` includes three *sweeps* through the cube, and each sweep involves $n(n-1)/2$ (p, q)-

$[U, V, W, \sigma] =$ **Jacobi-Compress**$(\mathcal{A})$

$\sigma \leftarrow \texttt{vec}(\mathcal{A})$
$U \leftarrow I_n; \quad V \leftarrow I_n; \quad W \leftarrow I_n;$
$X \leftarrow I_n; \quad Y \leftarrow I_n; \quad Z \leftarrow I_n;$
repeat
  $\sigma_0 \leftarrow \sigma$
  for $s = 1, 2, 3$ (each view of tensor cube) **do**
    % One sweep of the cube
    for $p = 1 : n-1$ **do**
      for $q = p+1 : n$ **do**

        Set $\Sigma \in \mathbb{R}^{2\times 2\times 2}$ to be the (p, q)-subtensor of σ_{s-1}

        $[\tilde{U}, \tilde{V}, \tilde{W}, \tilde{\sigma}] \leftarrow$ **Solve2-by-2-by-2**(Σ)

        % update other entries
        $W \leftarrow I_n; \quad V \leftarrow I_n; \quad U \leftarrow I_n$

        Update (p, q)-entries of σ_{s-1} with $\tilde{\sigma}$

        $W(p : q, p : q) \leftarrow W$
        $V(p : q, p : q) \leftarrow V$
        $U(p : q, p : q) \leftarrow U$
        $\sigma_s \leftarrow (W^T \otimes V^T \otimes U^T)\, \sigma_{s-1}$

        $X(:, [p \;\; q]) \leftarrow X(:, [p \;\; q]) \cdot U$
        $Y(:, [p \;\; q]) \leftarrow Y(:, [p \;\; q]) \cdot V;$
        $Z(:, [p \;\; q]) \leftarrow Z(:, [p \;\; q]) \cdot W;$
      end for
    end for
    $U_s \leftarrow X; \quad V_s \leftarrow Y; \quad W_s \leftarrow Z;$
  end for
  $\sigma \leftarrow \hat{\sigma}_3$
  $U \leftarrow U \cdot X_1 Z_2 Y_3$
  $V \leftarrow V \cdot Y_1 X_2 Z_3$
  $W \leftarrow W \cdot Z_1 Y_2 X_3$
until convergence

FIG. 4. *Jacobi-Compress for $n \times n \times n$ tensors.*

pairs. Solving the $2\times2\times2$ subproblem is constant work in both the case of maximizing the sums of squares and the trace. It is important to note that the update (4.2) can be computed in linear time by only updating those elements that are affected. Therefore, one iteration of `Jacobi-Compress` is $\mathcal{O}(n^3)$.

7. Block version. `Jacobi-Compress` can be converted to a block algorithm by representing an $n\times n\times n$ tensor as an $N\times N\times N$ block tensor with block size $r\times r\times r$, where $n = Nr$. For example, a $6\times6\times6$ tensor $\mathcal{A}$ can be regarded as a $3\times3\times3$ block

tensor with $2 \times 2 \times 2$ entries. The block version chooses a (p, q) pair, a $2r \times 2r \times 2r$ tensor, to be solved by `Jacobi-Compress`. Therefore, the only difference between the block version and standard `Jacobi-Compress` is the order in which the subproblems are solved.

8. Extension to $\ell \times m \times n$ tensors. Thus far, we have considered only three-way tensors for which each dimension is equal. Many applications involve data with unequal dimensions. By padding the data with zeros, `Jacobi-Compress` can be used for $\ell \times m \times n$ tensors.

Specifically, if $\tilde{p} = \max(\ell, m, n)$, then we pad the tensor with zeros that results in a $\tilde{p} \times \tilde{p} \times \tilde{p}$ tensor. Similar to the Jacobi SVD algorithm, unwanted fill does not occur. Note that computational shortcuts are taken so as to not actually store and perform calculations with zeros. In particular, if, say, $\ell >> m, n$, then many of the subproblems will contain all zeros. Therefore a sweep may contain considerably fewer subproblems than $\tilde{p}(\tilde{p} - 1)/2$.

9. Extension to order-p tensors. `Jacobi-Compress` can be extended to p-way tensors, but the results are not very practicable as p gets large. Maximizing the sums of squares of the diagonals is a direct extension of the $p = 3$ case in section 5.2; i.e., it involves taking SVDs of p 2×2 matrices. Unfortunately, extending the trace solution of section 5.1 does not involve an explicit solution to the subproblem but instead requires the help of a nonlinear solver to solve the optimization problem. More work is needed in this area.

10. Algorithm performance. In this section we describe the performance and numerical convergence typically seen in practice. In randomly generated examples, `Jacobi-Compress` typically converges in three iterations or less, i.e., the sums of squares of the diagonals or trace do not improve (up to a specified tolerance) after three iterations of the algorithm. We also note that in cases where a tensor is orthogonally diagonalizable, our algorithm finds that optimal form. The next example shows the compression of the algorithm.

Example 10.1. Let $\mathcal{A} \in \mathbb{R}^{3\times3\times3}$ be given by (in order of first face, second face, third face)

$$\mathcal{A} = \left[\begin{array}{ccc|ccc|ccc} 8 & 8 & 3 & 10 & 8 & 10 & 9 & 3 & 4 \\ 10 & 5 & 7 & 8 & 3 & 7 & 7 & 7 & 6 \\ 10 & 5 & 4 & 5 & 5 & 3 & 2 & 7 & 5 \end{array}\right].$$

Then `Jacobi-Compress` produces

$$\Sigma = \left[\begin{array}{ccc|ccc|ccc} \mathbf{.15} & -1.1 & 1.4 & 2.7 & .01 & -1.8 & -1.8 & -1.5 & .05 \\ 3.2 & .01 & .63 & 0 & \mathbf{33.4} & .03 & -.66 & .16 & -1.6 \\ -2.2 & 4.8 & .04 & -.17 & .30 & -.83 & -.03 & -.17 & \mathbf{-6.2} \end{array}\right].$$

One way to measure the compression of the algorithm is to look at the the "percent of norm" of the elements in the diagonals. For example, for an $n \times n \times n$ tensor, we can compute

$$\gamma = \frac{\sum_{i=1}^{n} \sigma_{iii}^2}{\sum_{i,j,k=1}^{n} \sigma_{ijk}^2},$$

which measures "how much" of the norm is contained in the diagonals of the tensor. The closer γ is to one, the better the compression. If the tensor is diagonalizable with

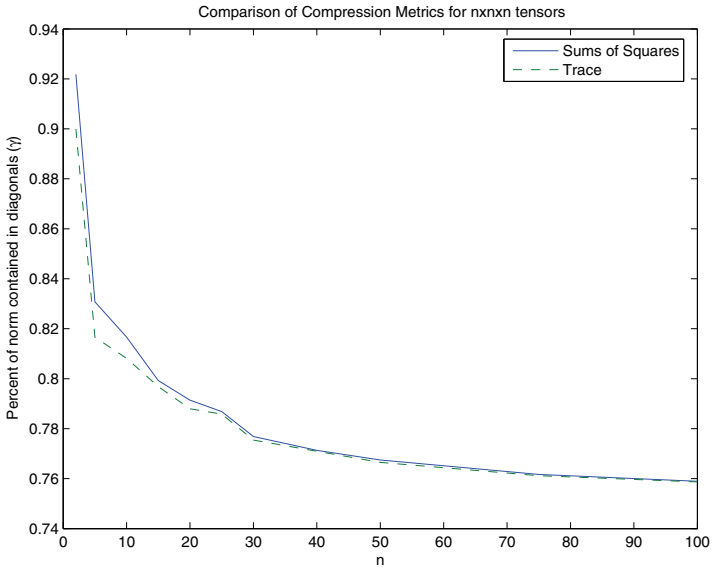

FIG. 5. *Percent of total norm obtained in the diagonal entries for randomly generated starting vectors using different metrics for compression for $n \times n \times n$ tensors.*

orthogonal transformations, γ should be equal to one. Figure 5 shows the average value of γ for different n after running `Jacobi-Compress` for $p = 3$ and the direct extension when $p = 4$. For comparison, γ tends to zero as n increases for random vectors. The clear trend from Figure 5 is that compression decreases as n increases. The inverse relationship between the amount of compression and n is expected since we have relatively fewer elements with which to explain n^3 or n^4 entries of the tensor. More importantly, however, is that the compression is quite high and tends to be around 75 percent compared to before running `Jacobi-Compress`, which is the result we focus attention to in Figure 5.

11. Conclusion. In this paper an algorithm based on the Jacobi SVD algorithm for matrices is given to compute an orthogonal tensor decomposition. The resulting tensor decomposition maximizes the sums of the squares of the "diagonals" or the trace of the tensor. The idea of the algorithm is to compute tensor decompositions of $2 \times 2 \times 2$ subtensors using an alternating least squares approach. In each case, the subproblem has an explicit solution involving the SVD. The general p-way case is still being explored.

REFERENCES

[1] C. A. ANDERSSON AND R. BRO, *Improving the speed of multi-way algorithms: Part* I. *Tucker*3, Chemom. Intell. Lab. Syst., 42 (1998), pp. 93–103.
[2] C. A. ANDERSSON AND R. BRO, *The N-way toolbox for MATLAB*, Chemom. Intell. Lab. Syst., 52 (2000), pp. 1–4.
[3] R. BRO, *PARAFAC. Tutorial and applications*, Chemom. Intell. Lab. Syst., 38 (1997), pp. 149–171.
[4] J. F. CARDOSO AND A. SOULOUMIAC, *Blind beam-forming for non-Gaussian signals*, IEEE Proceedings, Part F, 140 (1993), pp. 362–370.

[5] J. D. Carroll and J. Chang, *Analysis of individual differences in multidimensional scaling via an N-way generalization of "Eckart-Young" decomposition*, Psychometrika, 35 (1970), pp. 283–319.
[6] P. Comon, *Independent component analysis, a new concept?*, Signal Process., 36 (1994), pp. 287–314.
[7] P. Comon, *Tensor diagonalization, a useful tool in signal processing*, in Proceedings of the 10th IFAC Symposium on System Identification, Vol. 1, M. Blanke and T. Soderstrom, eds., IFAC–SYSID, Copenhagen, 1994, pp. 77–82.
[8] P. Comon, *Canonical Tensor Decompositions*, I3S report, RR-2004-17, CNRS-Laboratoire I3S, Université Nice-Sophia Antipolis, France, 2004.
[9] R. Coppi and S. Bolasco, eds., *Multiway Data Analysis*, Elsevier, Amsterdam, 1989.
[10] L. De Lathauwer, B. De Moor, and J. Vandewalle, *A multilinear singular value decomposition*, SIAM J. Matrix Anal. Appl., 21 (2000), pp. 1253–1278.
[11] L. De Lathauwer, B. De Moor, and J. Vandewalle, *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.
[12] L. De Lathauwer, B. De Moor, and J. Vandewalle, *Independent component analysis and (simultaneous) third-order tensor diagonalization*, IEEE Trans. Signal Process, 49 (2001), pp. 2262–2271.
[13] L. De Lathauwer, B. De Moor, and J. Vandewalle, *Computation of the canonical decomposition by means of a simultaneous generalized Schur decomposition*, SIAM J. Matrix Anal. Appl., 26 (2004), pp. 295–327.
[14] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed., Johns Hopkins University Press, Baltimore, MD, 1996.
[15] R. A. Harshman, *Foundations of the PARAFAC procedure: Model and conditions for an "explanatory" multi-mode factor analysis*, UCLA Working Papers in Phonetics, 16 (1970), pp. 1–84.
[16] R. Henrion and C. A. Andersson, *A new criteria for simple-structure transformations of core arrays in N-way principal component analysis*, Chemom. Intell. Lab. Syst., 47 (1999), pp. 190–204.
[17] G. Hori, *A general framework for SVD flows and joint SVD flows*, in Proceedings of the IEEE International Conference of Acoustics, Speech, and Signal Processing, Hong Kong, 2003, pp. 693–696.
[18] H. A. L. Kiers, *Tuckals core rotations and constrained Tuckals modelling*, Statist. Appl., 4 (1992), pp. 661–667.
[19] H. A. L. Kiers, *Towards a standardized notation and terminology in multiway analysis*, J. Chemom., 14 (2000), pp. 105–122.
[20] E. Kofidis and P. A. Regalia, *On the best rank-1 approximation of higher-order supersymmetric tensors*, SIAM J. Matrix Anal. Appl., 23 (2002), pp. 863–884.
[21] T. G. Kolda, *Orthogonal tensor decompositions*, SIAM J. Matrix Anal. Appl., 23 (2001), pp. 243–255.
[22] P. M. Kroonenberg, *Three-mode Principal Component Analysis: Theory and Applications*, DSWO Press, Leiden, The Netherlands, 1983.
[23] B. Pesquet-Popescu, J.-C. Pesquet, and A. P. Petropulu, *Joint singular value decomposition—a new tool for separable representation of images*, in Proceedings of the IEEE International Conference of Image Processing, Thessaloniki, Greece, 2001, pp. 569–572.
[24] D. T. Pham, *Joint approximate diagonalization of positive definite Hermitian matrices*, SIAM J. Matrix Anal. Appl., 22 (2001), pp. 1136–1152.
[25] N. Sidiropoulos, G. Giannakis, and R. Bro, *Blind PARAFAC receivers for DS-CDMA systems*, IEEE Trans. Signal Process., 48 (2000), pp. 810–823.
[26] N. Sidiropoulos, R. Bro, and G. Giannakis, *Parallel factor analysis in sensor array processing*, IEEE Trans. Signal Process., 48 (2000), pp. 2377–2388.
[27] A. Smilde, R. Bro, and P. Geladi, *Multi-way Analysis: Applications in the Chemical Sciences*, John Wiley & Sons, Chichester, England, 2004.
[28] L. R. Tucker, *Some mathematical notes on three-mode factor analysis*, Psychometrika, 31 (1966), pp. 279–311.
[29] M. A. O. Vasilescu and D. Terzopoulos, *Multilinear image analysis for face recognition*, in Proceedings of the International Conference on Pattern Recognition (ICPR), Quebec City, Canada, 2002, pp. 511–514.
[30] M. A. O. Vasilescu, *Human motion signatures: Analysis, synthesis, recognition*, in Proceedings of the International Conference on Pattern Recognition (ICPR), Quebec City, Canada, 2002, pp. 456–460.

[31] C. F. Van Loan, *The ubiquitous Kronecker product*, J. Comput. Appl. Math., 123 (2000), pp. 85–100.
[32] T. Zhang and G. H. Golub, *Rank-one approximation to high order tensors*, SIAM J. Matrix Anal. Appl., 23 (2001), pp. 534–550.
[33] Matlab, http://www.math.jmu.edu/∼carlam.