Original Research Article

An index formalism that generalizes the capabilities of matrix notation and algebra to n-way arrays

Richard A. Harshman
University of Western Ontario

Please address all correspondence to Richard A. Harshman at:
Psychology Dept.
University of Western Ontario
London, Ontario
Canada N6A 5C2

e-mail: harshman@uwo.ca
phone: 519-661-2111 ext 84691 (office)
519-661-3663 (lab)
fax: 519-661-3213

An index formalism that generalizes the capabilities of matrix notation and algebra to n-way arrays

Short title: N-way generalization of matrix notation

Summary

The capabilities of matrix notation and algebra are generalized to n-way arrays. The resulting language seems easy to use; all the capabilities of matrix notation are retained and most carry over naturally to the n-way context. For example, one can multiply a three-way array times a four-way array to obtain a three-way product. Many of the language's key characteristics are based on the rules of tensor notation and algebra. The most important example of this is probably the incorporation of subscript/index related information into both the names of array objects and the rules used to operate on them. Some topics that emerge are relatively unexplored, such as inverses of n-way arrays; these might prove interesting for future theoretical study.

KEYWORDS: Linear and multilinear algebra; tensors; array notation; three-way models; n-way arrays; Tucker; T2; T3; Parafac; Candecomp

Introduction

Shortcomings of current notation for multilinear models

Matrix notation is well suited for two-way data, such as measurements of many objects on multiple variables, and for two-way models, such as the structure given by a product of two or three matrices. However, when we consider higher-way arrays (e.g. objects by variables by conditions) and corresponding higher-way models, certain limitations become apparent. A single two-way array (i.e., a matrix) cannot *directly* represent a three-way 'cubical' array of data, nor can any sum of matrix products *directly* produce a three-way latent structural object.

Various devices have been used to adapt matrix notation to handle three-way or higher-way structure. Many of these are mentioned in References [1-3] , for example. They require either restriction to a single 'representative' slice or unfolding/'matricizing' [2] the n-way array into two-way form by adjoining slices, and then making use of special matrix products such as the Kronecker product and Khatri-Rao (or Khatri-Rao-Bro) product (see Reference [3]).

Very recently, an extensive set of suggestions concerning matrix and array terminology and notation was presented in Reference [2]. There, the objective is primarily to compile and organize an account of the best current practice, but also extensions are offered to facilitate work with n-way arrays, some of which resemble features of the (independently developed) proposals to be made below. (For example, the suggested use of ':' in subscripts (Reference [2], p. 109) resembles our use of upper case subscripts, as will become clear.) However, an adequate description or comparison is beyond the scope of this article.

A different approach is the graphical notation proposed by Alsberg in [4], and further studied, extended, and recommended for multilinear work by D. S. Burdick (paper presented at TRICAP 2000, the Third Annual Meeting on Three-Way Methods in Chemistry and Psychology, Faaborg, July 2000). Unlike previous proposals, it includes rules that allow it to easily express many non-matrix operations and transformations necessary for n-way multilinear algebra.

Proposal for new notation

The following pages describe an alternative to matrix notation and the associated rules of matrix manipulation. As far as we have been able to tell, it provides all the capabilities of matrix notation. Once one is familiar with it, the language may be a bit easier to use than standard matrix notation because it relaxes some restrictions such as noncommutativity of multiplication. In contrast to the Alsberg (and Burdick) diagrams, the approach to be described below is completely algebraic and non-graphical. Nonetheless,

both languages share the same underlying logical structure and unified approach to multilinear algebra and, in fact, seem complementary*.

Perhaps the most important feature of the proposed notation is that it treats three- and higher-way arrays in precisely the same way as vectors and matrices. It can represent and operate with arrays of any order with equal ease. Thus it is proposed here as a possible way to reduce or eliminate the difficulties arising from use of matrix notation in a multilinear context.

The language, which we call 'array index notation' (AIN) is inspired by the notation and rules of algebra used with *tensors*, which are multilinear objects that can have an arbitrary number of ways or modes. Since tensor notation is designed to be equally suitable for one-, two- or n-way tensors, it provides an appropriately flexible starting point for our array language. However, since AIN does not completely adopt the rules governing tensors, and can be used to work with arrays that do not have tensor properties†, it may also be described as a 'quasi-tensor' notation. (For an accessible introduction to tensor products with applications to familiar multilinear models, see D. S. Burdick [5], and for an introduction to tensors in the context of linear algebra, see Reference [6].)

General features of the new notation

AIN has three basic characteristics that are adapted from tensor notation:

(i) array names that are suitable for arrays of any order and that display important information needed for the algebra;

(ii) algebra rules that define meaningful sums and products for arrays of any order; and

(iii) an index summation convention that simplifies specification of linear operations on arrays.

Features of tensors and tensor notation *not* incorporated here include strict multilinearity, covariance and contravariance, and, more generally, any restriction on how the arrays must transform under a change of basis. (In fact, the idea of a 'change of basis' may not even be appropriate or defined for some applications and/or some models.)

Two new (non-tensor) principles are introduced:

(i) a subscript convention that allows sets of index values rather than single index values to be specified at a particular subscript position (this facilitates representation of subarrays and arrays); and

(ii) a different treatment and interpretation of subscript order.

These ideas are explained in Rules 2 and 7 below.

Other new notational devices are described which are helpful but do not fundamentally extend the notation. They are concerned with 'composite subscripts', which

---

*In fact, Alsberg and Burdick have both suggested that the diagram notation could serve as a "front end" to AIN.

† Some recent articles in the multi-way modeling literature use the word 'tensor' to mean 'three-way or higher-way array' and others treat it even more generally, as if it is interchangeable with the word 'array'. Either use risks interdisciplinary confusion, since it conflicts with the existing, more restricted use of 'tensor' by mathematicians, physicists, engineers, etc. For example, [6] gives the narrower definition and provides a way to determine whether an array has 'tensor character'.

help bridge AIN and conventional matrix methods, and 'composite array elements' (see Rule 3 below), which allow compact expression of partitioned arrays, for example.

Illustrative examples of the new notation

The features of the language are illustrated in Tables I-V. These tables provide a set of examples that starts with very basic characteristics of object names and gradually introduces each new feature. Some readers (e.g., those who prefer not to read instruction manuals before trying out a new device) may prefer to start by reading these tables to get a concise overview of the language, and then proceed to the formal statements of rules below in order to clarify aspects that the tables do not make clear.

Insert Tables I-V about here

Rules for array index notation

Seven basic rules for AIN are presented below. Where applicable, the reader will also be referred to particular parts of Tables I-V for illustrative examples of how a given rule is applied.

Rule 1. Array names

(a) *Definition of terms.* The term 'array' is used to represent any ordered set of numbers —or subset thereof (see Rule 2). This includes a vector, matrix, and three-way array and/or higher-way array. The number of ways or modes of an array corresponds to the number of array subscripts that are 'active' (i.e., that take on more than one value within the array; see Rule 2). Inactive subscripts, if attached, designate modes of the *parent* array of which the named object is a sub-array and are not counted as modes of the sub-array.

(b) *Name format.* The symbolic name of an array usually consists of a letter* to which subscripts have been attached on the right. For an n-way array, exactly *n* of these subscripts will be upper case. For example, x_{JK} represents a two-way array indexed by J and K. See Table I for further examples (other aspects of these examples will be explained in the rules below).

(c) *Multi-letter names.* Multiple letters can be used in an array name if their meaning is obvious, as in "$cov_{JJ'}$" (if a covariance matrix is under discussion) or when they are

* In this paper, lower case italics are used in the array name but others might prefer a different type-face and/or font. See the second ALS example, below.

appropriately linked (e.g., by an underline), as in $\underline{ssq}_{JK}$. Examples are given in line 5 of Table I and in Tables III-V.

(d) *Expressions in place of names.* A parenthesized AIN expression can be used in place of a letter or set of letters. For example, $(b_j c_k)_{JK}$ also represents a two-way array indexed by J and K, but one in which the values of the elements are computed in a specific way (see Rule 3 below). This method describes rather than names the array.

Rule 2. Simple subscripts

(a) *Upper- vs. lower-case subscripts.* An upper-case subscript indicates that all values in its range occur in the object represented (i.e., the subscript is 'active'); lower case indicates that only a single value occurs (i.e., the subscript is 'inactive'). For example, h_{Ij} is a subarray of h_{IJ} (i.e., it is a vector of elements taken across every level of the first mode at a single level of the second mode). More generally, the upper case letter is the name of an *index set* and the subscript takes on all values that are elements in that set*. Lines 7-19 in Table I provide examples of subarray notation.

In its normal role, where it identifies the index name associated with a given position, a subscript is not italicized. In h_{Ij}, for example, 'j' represents an *arbitrary* level of the index (i.e. an arbitrary element from the index set J). However, if a particular non-italic lower case appears more than once in an expression, it represents another occurrence of the *same* arbitrary level.

A subscript is italicized only if it denotes a *particular* value, one that has been defined outside the expression and is now being assigned to that index, as in h_{Iu}, which indicates that j = *u*. (This second expression is analogous to h_{I2} except that the external variable *u* is inserted in the second position instead of the constant 2. The index set for the second position is still J, but *u* picks out a particular element of that set.) Line 19 in Table I illustrates the difference, where the subscript symbol 'i' indicates that the first subscript position contains an arbitrary single value taken from an index set called 'I', while the subscript symbol '*u*' indicates that the fourth index position has the value of the variable *u*.

(b) *Expressions inside subscripts.* Instead of a letter or name, a subscript position may contain a valid AIN expression that can be evaluated to obtain the desired index value or

* The idea of an 'index set' containing assignable values for a given subscript position is taken from Reference [7].

set of index values, as in $h_{\mathrm{I},[k_1+2]}$ or $h_{\mathrm{I},[k_2\ k_4 \ldots\ k_{2r}]}$*. In such cases, subscripts are separated by commas (and spaces).

Rule 3. Composite subscripts and composite elements

(a) Composite subscripts. A composite subscript is written by enclosing two or more index-set symbols in parentheses. For example, $a_{\mathrm{I(JK)}}$ designates a two-way array that is a strung-out version of the three-way array a_{IJK}. The '(JK)' represents a single subscript that takes on a distinct value for each combination of an element from the index set J with one from the index set K; in other words, the composite (JK) subscript takes on the values of the Kronecker product J⊗K. (By convention, the left indicator changes fastest; see Rule 7 for more details about subscript order in the AIN context.)

(b) Composite elements. A composite element is written by enclosing an AIN expression or computation formula in parentheses and then assigning it subscripts. For example, $(b_{\mathrm{j}}c_{\mathrm{k}})_{\mathrm{jk}}$ refers to an element in the array given above in Rule 1(d). The expression inside the parentheses describes the contents of the array at the location designated by the subscripts outside the parentheses.

One or more upper-case subscripts inside parentheses indicates that the expression represents a sub-array instead of a single element. Any inner subscript that appears in upper case does not appear outside the parentheses. For example, the subscript 'J' in $(p_{\mathrm{i}}q_{\mathrm{J}}s_{\mathrm{k}})_{\mathrm{ik}}$ designates a vector at location (i,k) of a three-way array, while the entire array may be written as $(p_{\mathrm{i}}q_{\mathrm{J}}s_{\mathrm{k}})_{\mathrm{IK}} = t_{\mathrm{IJK}}$ or as $(p_{\mathrm{i}}q_{\mathrm{j}}s_{\mathrm{k}})_{\mathrm{IJK}} = t_{\mathrm{IJK}}$. The first representation of t_{IJK} emphasizes a particular sub-array structure while the second does not.

(c) *Array shape and equality*. Equality of two arrays, designated by the equal sign '=', implies equality of both the array contents and array shape (also sameness of orientation, see Rule 7, below). The symbol '$\cong$' is used to express a weaker equality, that of contents but not of shape, and is read 'equals ignoring shape'. It will usually occur when composite subscripts are employed for what was originally a higher-way array. Thus $a_{\mathrm{IJK}} \cong a_{\mathrm{I(JK)}}$ and $a_{\mathrm{IJK}} \cong a_{\mathrm{(IJ)K}}$, for example, which implies $a_{\mathrm{(IJ)K}} \cong a_{\mathrm{I(JK)}}$.

* An anonymous reviewer suggested adopting a MATLAB-like convention that uses the colon ':' to indicate sequences of subarray index values, as in $h_{\mathrm{I},\ k_2:2:k_{2x}}$ or $a_{\mathrm{I},\ j_1:j_2,\mathrm{K}}$.

Rule 4. Array algebra

Array operations follow the standard rules used for tensors:

(a) *Addition.* Arrays are summed by adding corresponding elements (i.e., elements with the same values in the same subscript positions).

(b) *Scalar multiplication.* An array multiplied by a scalar has all its elements multiplied by that scalar.

(c) *Array multiplication.* The product of two arrays is an array containing (before any contraction) each possible combination of an element from one array times an element from the other, indexed by the subscripts from both arrays. For example, the array product $a_{\mathrm{IJ}}b_{\mathrm{KLM}} = a_{\mathrm{IJ}} \otimes b_{\mathrm{KLM}} = c_{\mathrm{IJKLM}}$ has elements $c_{\mathrm{ijklm}} = a_{\mathrm{ij}}b_{\mathrm{klm}}$. Typically, array products will be used in conjunction with contraction (explained below). See Table II for examples of array multiplication (repeated subscripts and subscript embellishments such as I′ are discussed in Rule 6).

Rule 5. Contraction (array trace)

(a) *Contraction of an array.* Array contraction is a generalization of matrix trace. An n-way array can be *contracted* with respect to a given pair of indices by summing all elements where the index value on these indices is the same and placing the sum in an $(n-2)$-way array at the location determined by the $n-2$ indices not involved in the summation. Both indices involved in the contraction must have the same range of values (more generally, both index sets must have an equivalent set of elements).

For example, let b_{JL} be the result of contracting a_{IJKL} with respect to indices I and K. The elements of b_{JL} are given by $b_{\mathrm{jl}} = a_{1\,\mathrm{j}1\,\mathrm{l}} + a_{2\,\mathrm{j}2\,\mathrm{l}} + \cdots$ or in summation notation, by $b_{\mathrm{jl}} = \sum_i \sum_k a_{\mathrm{ijkl}}\delta_{ik}$, where δ_{ik} is the Kronecker delta (i.e., $\delta_{ik} = 1$ if i=k and 0 otherwise).

The contraction operation is represented by enclosing inside a set of parentheses both the array to be contracted and (separated by two vertical lines) the index pair on which the contraction is to be done.* For example, an expression for the array b_{JL} defined above is

$b_{\mathrm{JL}} = (a_{\mathrm{IJKL}} \,\|\, \mathrm{I{=}K})$.

* This contraction notation was originally proposed by D. S. Burdick for use with MATLAB (personal communication, circa 1990). He also showed how the Parafac and Tucker models can be written using this notation.

The two-way case is, of course, $(x_{IJ} \,||\, \text{I=J}) \;=\; x_{11} + x_{22} \ldots \;=\; trace(\mathbf{X})$.

(b) *Contraction on multiple index pairs in one array.* An array can be contracted on more than one pair of indices at a time, so long as the indices in each pair have the same range of values (same index set elements). For example, the array $c_{IK} = (a_{IJKLMN} \,||\, \text{J=L, M=N})$ has elements $c_{ik} = \sum_j \sum_l \sum_m \sum_n a_{i\,j\,k\,l\,m\,n} \delta_{jl} \delta_{mn}$, where δ_{jl} and δ_{mn} are Kronecker deltas. The entire array can also be expressed as

$$c_{IK} = \sum_j \sum_l \sum_m \sum_n a_{I\,j\,K\,l\,m\,n} \delta_{jl} \delta_{mn} \quad .$$

(c) *Generalization to k indices in one array.* A *generalized* or *k-index contraction* can be performed with respect to a specific set of *k* indices by summing all the elements having the same value on these *k* indices and placing the sum in an $(n-k)$-way array at the location determined by the $n-k$ indices not involved in the summation. For example, if d_{JMN} is the result of contracting a_{IJKLMN} on the indices I, K, and L, then we can write $d_{JMN} = (a_{IJKLMN} \,||\, \text{I=K=L})$. The elements are $d_{jmn} = \sum_i \sum_k \sum_l a_{ijklmn} \delta_{ik} \delta_{kl}$ or, equivalently, $\sum_i \sum_k \sum_l a_{ijklmn} \delta_{ikl}$, where δ_{ikl} is the (three-way) generalized Kronecker delta ($\delta_{ikl} = 1$ if $i = k = l$ and 0 otherwise). As always, all indices involved in the contraction (in this example, I, K and L) must have the same range of values.

(d) *Contraction of a pair of arrays (array multiplication with contraction).* Two arrays can be *contracted* with respect to a given pair of indices (one index from each array) by taking pairs of elements (one from each array) that have the same value on the two given indices, computing their product, summing these products, and placing the sum in a location in the product array determined by the remaining (non-matching) subscripts. For example, the standard matrix product **XY**=**Z** is produced by first computing the four-way product $x_{IJ} y_{KL} = z_{IJKL}$ and then contracting on J and K. (Another way of representing this is $x_{IJ} y_{JL} = z_{IJJL}$, with contraction on the repeated subscript; repeated subscripts are dealt with in Rule 6.)

The form of the final product array depends not only on the size and shape of the arrays being multiplied together but also on the user's purposes (i.e., which contractions are specified). Thus, for example, different products resulting from the same arrays might be represented as $(a_{IJK} b_{LM} \,||\, \text{J=L}) \;=\; (g_{IJKLM} \,||\, \text{J=L}) \;=\; h_{IKM}$ or as $(a_{IJK} b_{LM} \,||\, \text{J=L, K=M}) \;=\; (g_{IJKLM} \,||\, \text{J=L, K=M}) \;=\; h_I$, depending on the circumstances.

(e) *Generalization to k indices in several arrays.* A generalized or k-index contraction can be performed with respect to a specific set of *k* indices distributed among several arrays by forming products of all the elements having the same value on these *k* indices, summing these products and placing the sum in an $(n-k)$-way array at the location determined by the $n-k$ indices not involved in the summation.

For example, suppose d_{JMN} is the result of contracting an array product on the triple of indices I, K, and L such that $d_{\mathrm{JMN}} = (a_{\mathrm{IJ}} b_{\mathrm{KLM}} c_{\mathrm{N}} \parallel \mathrm{I{=}K{=}L}) = (u_{\mathrm{IJKLMN}} \parallel \mathrm{I{=}K{=}L})$. The corresponding summation notation is either $d_{\mathrm{jmn}} = \sum_i \sum_k \sum_l a_{ij} b_{klm} c_{\mathrm{n}} \delta_{ikl}$ or $d_{\mathrm{jmn}} = \sum_i \sum_k \sum_l u_{ijklmn} \delta_{ikl}$, where δ_{ikl} is the three-way generalized Kronecker delta as in (c) above. The notation using *u* represents the process as two-stage: first the direct product of all the arrays is formed, then a *k*-index generalized contraction is performed on the result. Note that the contraction need not involve every array in the product; here, for example, the array c_{N} is involved in the product, but not in the contraction.

Rule 6. Summation convention for array products

(a) *Summation convention for matrices.* For array products, there is sometimes an alternative to the contraction notation presented above. Tensor notation uses repeated subscripts to indicate how the product array is to be contracted*. For example, the matrix product **XY**=**Z** is written as $x_{\mathrm{IJ}} y_{\mathrm{JL}} = z_{\mathrm{IL}}$, with the contraction performed on the repeated subscript J. Another example is the product of the matrix a_{IJ} and a two-way subarray of the three-way array b_{JKQ} which may be written as $a_{\mathrm{IJ}} b_{\mathrm{JKq}} = c_{\mathrm{IKq}}$, where $c_{\mathrm{ikq}} = \sum_{j=1}^{J} a_{\mathrm{i}j} b_{j\mathrm{kq}}$. Under this convention, the explicit contraction notation is not needed but if used, c_{IKq} would be denoted as $a_{\mathrm{IJ}}\, b_{\mathrm{JKq}} = (a_{\mathrm{IJ}}\, b_{\mathrm{JKq}} \parallel \mathrm{J{=}J}) = (u_{\mathrm{IJJKq}} \parallel \mathrm{J{=}J}) = c_{\mathrm{IKq}}$. The examples in Table II also use repeated subscripts.

A second shared index represents an independent summation. For example, the multiplication of one three-way array and two matrices to obtain a three-way array product is written $a_{\mathrm{IJ}} b_{\mathrm{JK}} u_{\mathrm{HKL}} = \sum_{k=1}^{K} \sum_{j=1}^{J} a_{\mathrm{I}j} b_{jk} u_{\mathrm{H}k\mathrm{L}} = t_{\mathrm{HIL}}$.

* This is often called the 'Einstein summation convention'.

This summation convention is subject to the restriction that any subscript is repeated only twice. A generalization of the rule to more repetitions is given in (c) below.

(b) *Avoiding unwanted summation.* Sometimes it is natural to use a subscript index letter twice but *no* summation over this index is desired. In such cases, some embellishment is used to make the subscripts slightly different. Here we use a single quotation or 'prime' symbol, as is frequently done in standard tensor notation (see e.g. Reference [6]). There should be no ambiguity in the use of this symbol since the transpose (or generalized transpose) is not indicated by a prime in AIN (see Rule 7). To distinguish three or more versions of the same subscript, one can use multiple primes or introduce other embellishments such as the asterisk. If this becomes awkward because too many alternative versions of a given subscript are needed, one can employ subscripted subscripts.

For example, one could write $a_{\mathrm{I}}a_{\mathrm{I}'} = h_{\mathrm{II}'}$ to represent the outer product of a vector a_{I} with itself, and $b_{\mathrm{IJ}}b_{\mathrm{I}'\mathrm{J}} = \sum_{j=1}^{J} b_{\mathrm{I}j}b_{\mathrm{I}'j} = c_{\mathrm{II}'}$ for the cross-product matrix obtained by multiplying a matrix b_{IJ} by itself. To avoid having to use embellishments to denote no summation, of course, one can always use distinct subscript names and state explicitly which subscripts are equivalent. This might be preferred in some multi-way cases.

Having said this, however, embellished subscripts may in fact be contracted if they are repeated, using the convention described in 6(a) above. For example, $x_{\mathrm{IR}}y_{\mathrm{JR}'}z_{\mathrm{KR}'}b_{\mathrm{JR}''}c_{\mathrm{KR}''} = d_{\mathrm{IR}}$, where the double occurrence of R′, R″, J and K implicitly indicates contraction on these subscripts. The generalization to more repetitions in (c) below also applies to embellished subscripts. Sometimes this usage may be confusing, especially if the expression contains embellishments of other occurrences of the same index name to *prevent* summation, and in such cases should be avoided.

Some of the items in Tables II-V demonstrate both uses of embellishments. For example, line 9b in Table II shows I′ used once to indicate no summation over I, and J′ used twice to indicate contraction over J′.

(c) *Multilinear generalization of the summation rule.* When *n* arrays are being multiplied together and the same symbol occurs as a subscript in *k* of these, the product array is given a *k*-index generalized contraction on the matching subscripts (see also Rule 5(e)). In other words, when the same index occurs in three or more arrays in a multi-array product, elements having the same value of the repeated index are multiplied together and then these products are summed across the range of the index.

For example, the Parafac/Candecomp model [8,9] can be written for a single element as $x_{ijk} = \sum_r a_{ir} b_{jr} c_{kr}$ if we combine AIN with standard summation notation, and for the entire array as $x_{IJK} = \sum_{r=1}^{R} a_{Ir} \otimes b_{Jr} \otimes c_{Kr}$ in AIN and Kronecker product notation. AIN and composite subscripts allow us to write the model for the unfolded or matricized [2] version of the array as $x_{I(JK)} = a_{IR} (b_{jr} c_{kr})_{(JK)R}$. However, by using AIN and the generalized summation rule, the entire three-way array may be represented simply as $x_{IJK} = a_{IR} b_{JR} c_{KR}$.

It is always possible to replace an expression involving the generalized summation rule with one that does not, by using the generalized Kronecker delta. For example, the Parafac1 model can also be represented as $x_{IJK} = a_{IR} b_{JR'} c_{KR''} \delta_{RR'R''}$ (which highlights its structure as a special case of the Tucker3 model; see, e.g. References [10,11]). In fact, one *must* use the less compact form involving the generalized Kronecker delta whenever the generalized summation rule would lead to ambiguity.

Rule 7. Subscript order (including array transpose)

(a) *Array orientation and generalized transpose.* The order of the subscripts attached to an array name determines the array 'orientation'. Changing this order is a generalization of taking the transpose of a matrix. Consequently, if $x_{IJ} = \mathbf{X}$, then $x_{JI} = \mathbf{X}'$. Likewise, array a_{RJKS} is one possible transpose of the array a_{JKRS}. There are 24 possible orientations of this array, and so any particular version has 23 possible transposes.

(b) *Array orientation and equality.* The standard equal sign ('=') between two arrays implies equality of both the arrays and their orientation. Thus, in general, $a_{IJK} \neq a_{IKJ}$.

However, transposes have a weaker kind of equality, namely, equality of array structure and contents. The symbol '$\cong$' is used to express this weaker equality (but it does not preclude strict equality); it is read as 'equals as object' or 'equals ignoring orientation' or 'equal up to a permutation of subscripts'. Hence $a_{IJK} \cong a_{IKJ}$. Likewise, $y_{KL} \cong x_{IJ}$ implies either $y_{KL} = x_{JI}$ or $y_{LK} = x_{JI}$. Naturally, if $y_{KL} = x_{JI}$ then $y_{LK} = x_{IJ}$ and similarly, $y_{LK} = x_{JI}$ implies $y_{KL} = x_{IJ}$.

(c) *Symmetry.* If an array is unchanged when two particular indices are permuted (e.g., $a_{IJK} = a_{IKJ}$) then the array is symmetric with respect to those two indices. If the array reverses sign when two particular indices are permuted (e.g., $a_{IJK} = -a_{IKJ}$) the array is anti-symmetric or skew-symmetric with respect to the two indices. An array that remains

constant under any permutation of indices is symmetric (without qualification), and one that changes sign under permutation of any two indices is anti-symmetric (without qualification). With higher-way arrays, there are additional aspects of symmetry that can be considered (e.g., involving relations among effects of different possible permutations), but these are beyond the scope of this article.

(d) *Subscript order for the product of arrays.* An expression describing the product of two or more arrays does not, in itself, establish the subscript order of the result. If the preceding text has stated that 'subscript order is left unspecified' then $a_{\mathrm{IJ}}b_{\mathrm{MJK}} \cong c_{\mathrm{IMK}} \cong c_{\mathrm{MIK}} \cong \cdots$. Otherwise, the orientation of the product array is determined in one of the following ways (listed in order of priority):

(i) by explicitly specifying it in the expression, using '=' instead of '$\cong$';
(ii) by previously having specified an ordering convention (e.g., 'throughout this section, the default subscript symbol order will be M,V,I,U,W,G'); or
(iii) by using the alphabetical order of the subscripts.

(e) *Subscript order and subscript correspondence.* In an array product, corresponding subscripts (for purposes of contraction) are those with matching *symbols*. In an array sum, corresponding subscripts are those with matching *positions* in the subscript lists of the arrays being added. Thus, subscript order does not affect multiplication but it does affect addition. Subscript order also affects the sequence of elements resulting from a vec( ) operation.

(f) *Indicating hidden order via annotated subscripts.* In cases where ambiguities concerning subscript names, subscript correspondence, etc. might arise, subscripts can be 'annotated' by placing a horizontal line underneath the subscript set, and then any information needed below that. For example, the following expression represents a two-way subarray of a_{IJR} at level g+3 of I, the first subscript; it specifies that the second subscript, J, ranges over its full set of index elements, and that the third subscript, R, takes on only the three values 1, 3 and 8 rather than the original full index set for R:

$$a_{\substack{g+3,\ \mathrm{J},\ [1,3,8] \\ \mathrm{i}\quad \mathrm{J}\quad \mathrm{R}}} \quad .$$

Using the array operations

Array addition

Addition and subtraction of n-way arrays is used the same way and in the same contexts as vector and matrix addition. For example, a fallible version of the Tucker3 model can be represented as $x_{\mathrm{IJK}} = a_{\mathrm{IR}}b_{\mathrm{JS}}c_{\mathrm{KT}}g_{\mathrm{RST}} + e_{\mathrm{IJK}}$.

Array multiplication

Array multiplication under AIN is generally independent of array order. For example, by using our freedom to chose the orientation of the array product, we can write $a_{IJK}b_{JR} = c_{IKR}$ and $b_{JR}a_{IJK} = c_{IKR}$. (If the subscript order has already been fixed for some reason, however, we can only write $a_{IJK}b_{JR} \cong b_{JR}a_{IJK}$.)

It is not the underlying operations (tensor product and contraction) that make standard matrix multiplication non-commutative, but rather, it is the convention that the contraction be applied to subscripts that are 'adjacent' (i.e., the column index of pre-multiplying matrix and row index of the post-multiplying matrix). Hence the product depends on the order of the matrices. In contrast, AIN contracts on subscripts that have matching index-set names ('I', 'J', etc.), and this is independent of order and orientation of the arrays being multiplied.

Another difference is that matrix notation determines the order of the subscripts of the product matrix by the order of the matrices being multiplied. The subscript order in AIN is more flexible, however, since it can be chosen as desired or as determined by a convention.

Appendix I provides further details on the application and interpretation of array multiplication, including demonstrations of commutativity and associativity.

Array 'division' or inversion

What kind of array inverses exist?

A matrix inverse is that matrix which, when multiplied by the original one, yields the identity matrix. When this is generalized to rectangular matrices, the identity is only expected if the matrix is multiplied by its inverse in the appropriate order, and the resulting identity is not expected to be the same size as the original matrix. For the rank deficient case, and for other kinds of generalized inverse that are 'weaker' than Moore-Penrose, the expectations are even more modest. It is not surprising, then, that we should be prepared to accept some restriction on the conditions of multiplication and some flexibility concerning the expected results, when the notion of inverse is generalized to higher-way arrays.

What should result when an n-way array is multiplied by its inverse? Sometimes one might want a superdiagonal array (i.e., a generalized Kronecker delta) but other times, an array that is 'slicewise' identity (i.e., consists of matrices, each of which is the identity). There are almost certainly additional interesting and potentially useful kinds of inverses of higher-way arrays. Inverses of a particular kind might not exist for a particular class of arrays, or might only exist if certain rank conditions are fulfilled, or perhaps only for certain very special cases.

It would be interesting to know what regularities can be discovered about which kinds of inverses exist and when. We have not studied the general question of higher-way inverses and know of no significant work in this area, although those who have studied array rank and/or array diagonalization, such as J. DeLeeuw, H. A. L. Kiers, J. B. Kruskal,

and J. TenBerge, have thereby implicitly worked on array inverses. If progress has been made, some reader(s) might bring this to our attention; if not, I would encourage mathematically talented scholars to look at this question.

As an example of how AIN might be used in the study of array inverses, consider the simplest Parafac model $x_{IJK} = a_{IR} b_{JR'} c_{KR''} \delta_{RR'R''}$, mentioned above in Rule 6. Suppose a_{IR}, $b_{JR'}$ and $c_{KR''}$ are full column rank. What happens if we define a six-way array y as $(a_{IR} b_{JR'} c_{KR''})^{+} = a^{+}_{I\underline{R}} b^{+}_{J\underline{R}'} c^{+}_{K\underline{R}''} = y_{IJK\underline{R}\underline{R}'\underline{R}''}$? Then we have

$$x_{IJK} y_{IJK\underline{R}\underline{R}'\underline{R}''} = (a_{IR} b_{JR'} c_{KR''} \delta_{RR'R''})(a^{+}_{I\underline{R}} b^{+}_{J\underline{R}'} c^{+}_{K\underline{R}''}) \quad .$$

This may be rewritten as

$$x_{IJK} y_{IJK\underline{R}\underline{R}'\underline{R}''} = (a_{IR} a^{+}_{I\underline{R}})(b_{JR'} b^{+}_{J\underline{R}'})(c_{KR''} c^{+}_{K\underline{R}''}) \delta_{RR'R''}$$

or as

$$x_{IJK} y_{IJK\underline{R}\underline{R}'\underline{R}''} = \delta_{R\underline{R}} \delta_{R'\underline{R}'} \delta_{R''\underline{R}''} \delta_{RR'R''} = \delta_{\underline{R}\underline{R}'\underline{R}''} \quad .$$

Thus, by one definition of a three-way inverse, *y* would seem to be the inverse of *x*. In the two-way case, this type of inverse is closely related to the Moore-Penrose inverse, but in general, this method would seem to produce an inverse with twice as many ways as the array for which it is the inverse. There are probably inverses that are more compact, and they could be more desirable.

Note, however, that caution is required when working with array inverses. In expressions such as those above, a very specific interaction between different quantities is represented but this is not reflected by specific notation. AIN does not currently have distinct symbols or conventions to differentiate between different kinds of array inverses, and so the intended meaning of expressions like "x^{+}_{IJK}" must be defined each time or be quite clear from context. This is an area for further development by those studying array inverses (and/or solutions to least squares problems, see below).

Inverses for solving least squares problems

Clearly, one important kind of inverse is necessary if AIN is to be a suitable substitute for, and generalization of, matrix notation. This is the class of 'generalized inverses' which provides a solution to certain least-squares problems posed in terms of arrays, and is important for estimation of models by methods like Alternating Least Squares (ALS). Model estimation can also be done by methods such as Paatero's Multilinear Engine [7], which is sufficiently flexible and powerful to allow the estimation of the parameters of higher-way models without direct closed-form computation of array inverses. Nonetheless, the widespread use and good features of ALS prompts us to provide this option in the AIN context, and it is demonstrated later in the paper.

Uses of composite subscripts

Composite subscripts can be used both to regroup subscripts or 'unfold' a higher-way array into a lower-way one and to 'vectorize' it. For example, a_{IJK} can be unfolded into a vertical matrix $a_{(IJ)K}$, unfolded horizontally as $a_{I(JK)}$ or vectorized as $a_{(IJK)}$. A higher-way array such as b_{IJKLMN} can be unfolded in various ways, for example into a four-way array $b_{(IJ)(KL)MN}$ or into a matrix $b_{(IJK)(LMN)}$, and it can also be vectorized as $b_{(IJKLMN)}$. The ordering of the elements in the unfolded array or vector should be explicit –a suggested convention is that the fastest changing (or most deeply nested) subscript is on the left, and those changing successively more slowly occur in sequence left to right. Unfolding arrays into matrices is needed in problems involving generalized inverses, as will be shown in the ALS examples below.

Another important use of composite subscripts is for designating subparts of higher-way arrays. This use is demonstrated in Appendix II with partitioned matrices and higher-way arrays.

AIN representation of some important models

Tables III and IV show matrix notation contrasted with AIN for some two-way factor models and several ways of writing Parafac models, respectively. Only AIN is presented in Table V for the various Tucker models. The two-way factor models are self-explanatory; they are easily represented in matrix notation and what AIN adds is the size of each matrix. The same might be said for the other models when they are presented in two-way form, but AIN is the only alternative for higher-way representations.

The Parafac1 model [8] for a full array cannot be represented in matrix notation, and so it is also presented for any slice and for the unfolded array. The full array AIN has three variations here, the first of which demonstrates the generalized summation rule over R, the second which represents the model as a special case of the Tucker3 model [10,11] and the third which views R (i.e., factors) as a fourth mode and incorporates a vector of R ones (i.e., $\mathbf{1}_R$) to sum over it (i.e., sum the factor contributions).

The arbitrary-slice Parafac1 AIN is also given in three forms. The first is a simple multiplication of three matrices. The second uses composite elements and the generalized summation rule to accomplish the same thing. The third multiplies two matrices and a vector from **C** without an intermediate diagonal matrix. It also employs the generalized summation rule. The third version is presented in two ways to demonstrate the uses of commutativity: the first more closely mirrors the corresponding matrix formula, the second more closely represents the underlying logic and is more simply related to the prior full-array representation.

The two unfolded arrays are written in matrix form using Bro's notation [1] and are represented in AIN by using regrouped subscripts and composite elements. None of the alternative representations for Parafac1 match the simplicity of the first alternative given for the full array, however. Their value lies in their different perspective, which may be

useful in specific situations (such as a discussion bridging the gap between matrix and array models).

The Parafac2 model [1,12]* for the full array, which once again matrix notation cannot represent, requires the use of composite elements in AIN. The AIN representation of an arbitrary slice is another demonstration of the generalized summation rule, and of the order independence of matrix multiplication.

The various Tucker models (see [11] for Tucker2, -3 and -*n*) in Table V, are all easily represented in AIN in full array modeling form. ('Tucker0' is not a real model, of course, and is included only for completeness.) Just as with the Parafac models in Table IV, these models can be represented in matrix notation as well as AIN if the array is unfolded, but were not included in the table to save space. The last two lines of Table V give special versions of T3 and T2 for covariance analysis. The last model is the basis for Carroll's independently developed MDS model called IDIOSCAL (J. D. Carroll and J. J. Chang, paper presented at the Meeting of the Psychometric Society, Princeton, NJ, March 1972) [9,13] which allows IDIOsyncratic SCALing of both dimension weights and angles.

Least squares estimation of model parameters

We now have the machinery needed to use AIN for solving multilinear problems. We can demonstrate this by considering the estimation of parameter sets for multilinear models.

Example 1. ALS estimation of the Parafac1 model

As noted in Rule 7(c) and in Table IV, one way to write the Parafac/Candecomp model is $x_{\mathrm{IJK}} = a_{\mathrm{IR}} b_{\mathrm{JR}} c_{\mathrm{KR}}$. One can easily express the ALS procedure to estimate any one of the sets of parameters *a*, *b* and *c* (i.e., obtain a least squares solution for one set while taking the others to be fixed) by using composite elements and composite subscripts. For example, we can write the Mode A estimation procedure as

$$a_{\mathrm{IR}} = x_{\mathrm{IJK}} (y_{\mathrm{JKR}})^{+}$$

where

$$y_{\mathrm{JKR}} = (b_{\mathrm{jr}} c_{\mathrm{kr}})_{\mathrm{JKR}} \quad .$$

To obtain $(y_{\mathrm{JKR}})^{+}$, we first regroup the elements of *y* as $y_{(\mathrm{JK})\mathrm{R}}$ so that it becomes a two-way array, then compute its Moore-Penrose inverse $\left(y_{(\mathrm{JK})\mathrm{R}}\right)^{+}$ and finally, 'ungroup' the elements of the inverse.

* See also Kiers HAL, Ten Berge JMF, Bro R. PARAFAC2—Part I. A direct fitting algorithm for the PARAFAC2 model. *J. Chemometrics* 1999; **13**: 275-294.

In fact, we could have used composite subscripts to impose a corresponding two-way regrouping on x_{IJK} at the beginning. Then the estimation formula could have been viewed as an ordinary matrix equation

$$a_{\mathrm{IR}} = x_{\mathrm{I(JK)}} \, (y_{\mathrm{(JK)R}})^{+} \quad .$$

Some might prefer to do this in practice, since, based on what we know now, the array inversion must be reduced to a matrix problem. In theory, however, this added regrouping is unnecessary. With further developments in the area of array inverses, it is hoped that the need for regrouping will be reduced or eliminated altogether.

Contrast the regrouping of x_{IJK} into $x_{\mathrm{I(JK)}}$ with the more cumbersome and less transparent matrix notation, which would represent the model as

$$\left[\mathbf{X}_1 \mid \mathbf{X}_2 \mid \ldots \mid \mathbf{X}_K\right] = \left[\mathbf{A}\mathbf{D}_1\mathbf{B}' \mid \mathbf{A}\mathbf{D}_2\mathbf{B}' \mid \cdots \mid \mathbf{A}\mathbf{D}_k\mathbf{B}' \mid \cdots \mid \mathbf{A}\mathbf{D}_K\mathbf{B}'\right],$$

where the $\mathbf{X}_k$ are *I* by *J* slices of the data array; the $\mathbf{D}_i$ diagonals comprise rows of **C**, a *K* by *R* set of parameters; and **A** and **B** are *I* by *R* and *J* by *R* parameter sets, respectively. The estimation of **A** would then be represented as

$$\mathbf{A} = \left[\mathbf{X}_1 \mid \mathbf{X}_2 \mid \ldots \mid \mathbf{X}_K\right]\left[\mathbf{D}_1\mathbf{B}' \mid \mathbf{D}_2\mathbf{B}' \mid \cdots \mid \mathbf{D}_k\mathbf{B}' \mid \cdots \mid \mathbf{D}_K\mathbf{B}'\right]^{+}$$

(see e.g. Reference [14], p. 50). The economy and transparency of AIN is even more apparent in four-way Parafac, where the estimation of the Mode A matrix would be represented as

$$a_{\mathrm{IR}} = x_{\mathrm{I(JKL)}}((b_{\mathrm{jr}}c_{\mathrm{kr}}d_{\mathrm{lr}})_{\mathrm{(JKL)R}})^{+} .$$

Sometimes AIN will suggest a computation that might not have been tried otherwise, for example, an alternative algorithm for Parafac that estimates the parameters one row at a time. The estimation sub-step to update one row of *a* can be written (with the '+' designating the inverse placed above the subscript for compactness) as

$$a_{\mathrm{iR}} = x_{\mathrm{iJK}} \, b^{+}_{\mathrm{JR'}} \, c^{+}_{\mathrm{KR"}} \quad .$$

This is not a least squares procedure, but it avoids the computation and inversion of the larger *y* matrix that is necessary in the least squares version. Preliminary tests suggest that an iteration may take only 4% of the execution time required by the true least squares version. On the other hand, it may have undesirable properties such as instability in some cases–this remains to be studied[*].

[*] This approach has recently been proposed by Jiang J, Wu H, Li Y, Yu R. Three-way data resolution by alternating slice-wise diagonalization (ASD) method. *J. Chemometrics* 2000; **14**: 15-36.

Alternative conventions for array symbol fonts

Until now, we have used lower-case italics for the non-subscript portion of an array name. This convention was chosen to express the underlying unity of the nature of all array objects, be they scalars, vectors, n-way arrays, elements from arrays, subarrays, or whatever. In addition, this allows the text part of the name of a given array to be the same as that of any subarray of that array, including a single element.

This typographical convention is not an essential part of AIN, however. Other more distinctive fonts could be used for the array names, or even different display conventions for the text part of the name to distinguish arrays of different orders (as below). These would not change the properties of the objects or the rules for working with them.

Besides demonstrating general properties of AIN, the second ALS example below illustrates the use of an alternative typeface, one suggested by Henk Kiers (personal communication, July 2000). The usual matrix notation conventions for vectors, matrices, and arrays are retained, and so the resulting equations look more familiar and may be easier for some users to read. This typography might thus be used to facilitate the transition from matrix notation to full AIN.*

Example 2. ALS estimation of the parameters in the Tucker3 model

The AIN representation of the Tucker3 model is $\underline{\mathbf{X}}_{IJK} = \mathbf{A}_{IR}\mathbf{B}_{JS}\mathbf{C}_{KT}\underline{\mathbf{G}}_{RST}$.

If the Tucker model being estimated has the same number of factors for each mode, estimation of the core array is simply and transparently written (with the compact inverse notation used above) as

$$\underline{\mathbf{G}}_{RST} = \underline{\mathbf{X}}_{IJK}\ \mathbf{A}^{+}_{IR}\mathbf{B}^{+}_{JS}\mathbf{C}^{+}_{KT} \quad ,$$

which is a true least squares procedure based on a two-way case proven in Reference [15], pp. 60-61.

(Note that even in the case where $\mathbf{A}^{+}_{IR}\mathbf{B}^{+}_{JS}\mathbf{C}^{+}_{KT}$ do not all have the same rank, the solution above will work. This is because no contraction has been done, and so the inverse is a six-way array which retains the full information.)

Of course, it would also be possible to estimate the core by regrouping elements so that the approach more closely parallels the standard matrix one. This would be

---

* (An even more minimalist approach --not illustrated in this article-- is to use conventional matrix notation *and conventional rules of algebra* except where it is helpful to invoke the extended capabilities of AIN. The AIN terms are distinguished from standard matrix terms by their appended upper-case subscripts, and this implies that AIN rules of algebra must be used when manipulating them. However, this hybrid approach might seem awkward to many users and would probably be replaced by a more uniform AIN notation as familiarity with AIN increased.)

$$\mathbf{g}_{(\mathrm{RST})} = \mathbf{x}_{(\mathrm{IJK})}(a_{\mathrm{ir}}b_{\mathrm{js}}c_{\mathrm{kt}})^{+}_{(\mathrm{IJK})(\mathrm{RST})} \quad .$$

If we wanted to avoid the use of a composite element, we could let $\underline{\mathbf{Z}}_{\mathrm{IJKRST}} = \mathbf{A}_{\mathrm{IR}}\mathbf{B}_{\mathrm{JS}}\mathbf{C}_{\mathrm{KT}}$ and, since $\mathbf{x}_{(\mathrm{IJK})} = \mathbf{g}_{(\mathrm{RST})}\mathbf{Z}_{(\mathrm{IJK})(\mathrm{RST})}$, we could write the estimation as $\mathbf{g}_{(\mathrm{RST})} = \mathbf{x}_{(\mathrm{IJK})}(\mathbf{Z}_{(\mathrm{IJK})(\mathrm{RST})})^{+}$.

The estimation of the Mode A, B, and C loading matrices is similar to that used in Parafac, except with the addition of the core array. For example, the Mode A estimation is

$$\mathbf{A}_{\mathrm{IR}} = \underline{\mathbf{X}}_{\mathrm{IJK}}\underline{\mathbf{Y}}^{+}_{\mathrm{JKR}} \quad ,$$

where

$$\underline{\mathbf{Y}}_{\mathrm{JKR}} = (b_{\mathrm{js}}c_{\mathrm{kt}})_{\mathrm{JKST}}\,\underline{\mathbf{G}}_{\mathrm{RST}} \quad .$$

What was said in Example 1 above about the computation of $\underline{\mathbf{Y}}^{+}_{\mathrm{JKR}}$ applies here as well, as does the comment about regrouping $\underline{\mathbf{X}}_{\mathrm{IJK}}$ into $\mathbf{X}_{\mathrm{I(JK)}}$. In this case, the two-way regrouping of $\underline{\mathbf{Y}}_{\mathrm{JKR}}$ can be viewed in different ways, however. One is

$$\mathbf{Y}_{(\mathrm{JK})\mathrm{R}} = \left((b_{\mathrm{jS}}c_{\mathrm{kT}}g_{\mathrm{rST}})_{(\mathrm{JK})\mathrm{R}}\right) \quad ,$$

and another is

$$\mathbf{Y}_{(\mathrm{JK})\mathrm{R}} = \left((b_{\mathrm{js}}c_{\mathrm{kt}})_{(\mathrm{JK})(\mathrm{ST})}\mathbf{G}_{\mathrm{R(ST)}}\right) \quad .$$

The second regrouping allows us to obtain $\mathbf{Y}^{+}_{(\mathrm{JK})\mathrm{R}}$ from the product of two other matrix inverses, i.e.

$$\mathbf{Y}^{+}_{(\mathrm{JK})\mathrm{R}} = \left((b_{\mathrm{js}}c_{\mathrm{kt}})_{(\mathrm{JK})(\mathrm{ST})}\mathbf{G}_{\mathrm{R(ST)}}\right)^{+} = (b_{\mathrm{js}}c_{\mathrm{kt}})^{+}_{(\mathrm{JK})(\mathrm{ST})}\mathbf{G}^{+}_{\mathrm{R(ST)}}$$

(provided that $(b_{\mathrm{js}}c_{\mathrm{kt}})^{+}_{(\mathrm{JK})(\mathrm{ST})}$ and $\mathbf{G}^{+}_{\mathrm{R(ST)}}$ have sufficient rank). This has the advantage that the larger matrix $\mathbf{Y}_{(\mathrm{JK})\mathrm{R}}$ does not have to be inverted. The 'ungrouped' $\mathbf{Y}^{+}_{(\mathrm{JK})\mathrm{R}}$ may now be represented as

$$\underline{\mathbf{Y}}^{+}_{\mathrm{JKR}} = (b_{\mathrm{js}}c_{\mathrm{kt}})^{+}_{\mathrm{JKST}}\,\underline{\mathbf{G}}^{+}_{\mathrm{RST}} \quad .$$

The flexibility in grouping demonstrated by the above examples is the result of the relatively arbitrary nature of the arrangement of elements in these arrays; two different groupings can be functionally equivalent so long as the subscripts unambiguously define the particular multiplication and summation of elements necessary to obtain the product independent of the grouping. Thus, by using composite subscripts, higher-way arrays can be reduced to matrices where conventional methods and ideas of inverses may be applied.

Future directions

It is easier to demonstrate how AIN can simplify the expression of known multilinear models and estimation methods than it is to demonstrate how it might facilitate fundamentally new kinds of models or methods. There are, however, a few observations that suggest possible new directions.

Because of its greater flexibility, expressions can be written in AIN that specify relationships among arrays that seem impossible to express in matrix notation*. Consider the equation

$$x_{IJKL} = a_{IRST} b_{JRUV} c_{KSUW} d_{LTVW} \quad .$$

Here, every array is linked by one contraction to each of the other three. How could this possibly be represented with matrix notation, even allowing unfolding of *x*? Or consider the simpler three-way version given in line 12 of Table II. Here, there are three 'sources' or 'types' of factors: type-r, type-s, and type-t. These factors do not act through individual modes but instead through *pairs* of modes. Consequently, each of the three modes has a factor loading 'matrix' that is actually a three-way 'factor loading array', containing an entry at each external level (i or j or k) for each *combination* of two internal levels (r and s, r and t, or s and t). This seems to represent some kind of factor interaction model, where individual factors do not have a simple multiplicative effect in a given mode. Might this be a more appropriate approach for modeling an ecological network? Further generalizations may also be useful. For example, could there be a meaningful model in which the loadings array for some mode(s) have a different number of ways from others? Could it even make sense to combine this with one or more arrays that are purely internal, similar to Tucker's 'core array'?

The flexibility of AIN for expression of relationships might also make it possible to use it to represent structural equation models. Since little work has been done on three-way or higher-way structural equation models, perhaps this could be a useful way to approach this research area.

* This raises a fundamental question: can a proof of the nonequivalence (or equivalence) of the two languages be constructed? That is, for any expression that can be written in AIN is it possible to find an equivalent matrix expression (involving unfolded arrays)? Is there always (AIN product) $\simeq$ (matrix product) or even (AIN expression) $\simeq$ (matrix expression)?

Acknowledgements

I would like to thank Marg Lundy for many important contributions, and also Donald Burdick for inspiration and Henk Kiers for useful discussion. This research was supported by Natural Sciences and Engineering Research Council of Canada research grant OGP-000-7896.

References

1. Bro R. *Multi-way Analysis in the Food Industry: Models, Algorithms and Applications.* University of Amsterdam: Amsterdam, 1998; http://www.mli.kvl.dk/staff/foodtech/brothesis.pdf [1 March 2001].

2. Kiers HAL. Towards a standardized notation and terminology in multiway analysis. *J. Chemometrics* 2000; **14**: 105-122.

3. Harshman RA. 'Stretch' vs.'slice' methods for representing three-way structure via matrix notation. *Department of Psychology Research Bulletin #761*; University of Western Ontario: London, ON, 2001.

4. Alsberg BK. A diagram notation for N-mode array equations. *J. Chemometrics* 1997; **11**: 251-266.

5. Burdick DS. An introduction to tensor products with applications to multiway data analysis. *Chemom. Intell. Lab. Syst*. 1995; **28**: 229-237.

6. Akivis, MA, Goldberg VV. *An Introduction to Linear Algebra & Tensors* (rev. English edn), Silverman RA (trans. and ed.). Prentice-Hall: Englewood Cliffs, NJ, 1972.

7. Paatero P. The multilinear engine–a table-driven, least squares program for solving multilinear problems, including the n-way parallel factor analysis model. *Comput. Graph. Statist*. 1999; **8**: 854-888.

8. Harshman RA. Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-modal factor analysis. *UCLA Working Papers Phonet.* 1970*;* **16**: 1-84.

9. Carroll JD, Chang JJ. Analysis of individual differences in multidimensional scaling via an N-way 'Eckart-Young decomposition. *Psychometika* 1970; **35**: 283-319.

10. Tucker LR. Some mathematical notes on three-mode factor analysis. *Psychometrika* 1966; **31**: 279-311.

11. Kroonenberg PM. *Three-mode Principal Component Analysis*. DSWO Press: Leiden, The Netherlands, 1983.

12. Harshman RA. PARAFAC2: Mathematical and technical notes. *UCLA Working Papers Phonet.* 1972*;* **22**: 30-44.

13. Carroll JD, Wish M. Models and methods for three-way multidimensional scaling. In *Contemporary Developments in Mathematical Psychology, Vol. 2*, Krantz DH, Atkinson RC, Luce RD, Suppes P (eds). Freeman: San Francisco, 1974; 57-105.

14. Harshman RA, Lundy ME. PARAFAC: Parallel Factor Analysis. *Comput. Statist. Data Anal.* 1994; **18:** 39-72.

15. Rao CR, Mitra SK. *Generalized Inverse of Matrices and its Applications*. Wiley & Sons: New York, 1971.

16. Harshman RA, Lundy ME. Data preprocessing and the extended PARAFAC model. In *Research Methods for Multimode Data Analysis*, Law HG, Snyder CW, Jr, Hattie JA, McDonald RP (eds). Praeger: New York, 1984; 216-284.

17. Kruskal JB, Harshman RA, Lundy ME. How 3-MFA data can cause degenerate PARAFAC solutions, among other relationships. In *Multiway Data Analysis*, Coppi R, Bolasco S (eds). North-Holland: Amsterdam, 1989; 115-121.

Appendix I. Examples of the substantive interpretation of array multiplication

As a concrete application of how higher-way array products can be interpreted, consider the problem of computing the vitamin content of foods. We start simply, by first assuming that we have two matrices, one *V* by *I* and the other *I* by *D*, that we call a_{VI} and b_{ID}, respectively. The element a_{vi} gives the amount of vitamin *v* (e.g., ascorbic acid, riboflavin, etc.) in cooking ingredient *i* (peas, beef, butter, etc.), expressed, say, in milligrams of the vitamin per gram of the ingredient. The entry b_{id} gives the amount of ingredient *i* in dish *d* (chicken soup, beef stew, chocolate cake, etc.) expressed, say, in grams of ingredient per kilogram of the dish.

The product of these two matrices, c_{VD}, specifies the amount of each vitamin in each dish (in milligrams per kilogram), and the relation can be represented in AIN as $a_{\mathrm{VI}} b_{\mathrm{ID}} = c_{\mathrm{VD}}$.

Array-matrix multiplication

Now suppose that we start with more information. Since year-to-year weather variations and/or farming method changes and/or variation in genetics of the seed planted, etc., produce changes in vitamin content, we consider the year that the ingredient was grown as another factor influencing vitamin content. We incorporate this additional information by converting a_{VI} to a three-way array a_{VIY} ; this array gives the typical amount of each vitamin (*v*) in each ingredient (*i*) as measured each year (*y*). Now, the product of array a_{VIY} and b_{ID} , which may be easily represented as

$$a_{\mathrm{VIY}} b_{\mathrm{ID}} = c_{\mathrm{VDY}} \quad ,$$

gives the vitamin content of each dish for the year in which it was prepared. Figure 1 shows diagrammatically how to obtain c_{VDY} . Implicitly, the tensor product of a_{VIY} and b_{ID} is computed to produce a 5-way result, which is then contracted on 'I' (Ingredients) to produce a three-way product.

Array-array multiplication

In our multilinear modeling literature, array-matrix multiplication has already been encountered (see Kruskal as cited in [16], p. 256, Reference [17], pp. 115-116, and Reference [2]), but array-array multiplication is unfamiliar. However, AIN defines such products and makes determining the result quite straightforward. We again use our food example to demonstrate a meaningful interpretation.

Suppose our matrix b_{ID} is also modified to include information about the recipe used for each dish, since the amounts of each ingredient may differ from one cooking authority (e.g., cookbook) to another. We replace b_{ID} with b_{IDA} , which gives the amount

of ingredient *i* in dish *d* if prepared according to the recipe of authority *a*. Our four-way product array,

$$a_{\mathrm{VIY}} b_{\mathrm{IDA}} = c_{\mathrm{VDYA}} \quad ,$$

(see Figure 1, bottom) gives the amount of each vitamin in any given dish during any particular year, if the dish were prepared according to a particular recipe. Formally, this array multiplication can be decomposed into computation of the 6-way tensor product followed by contraction on 'I', which yields a four-way array.

A triple product

Now suppose that we introduce a third array, d_{DPY}, which tells us how many times each dish was consumed by each of several persons in each of the years being considered. We can then form the triple array product

$$a_{\mathrm{VIY}} b_{\mathrm{IDA}} d_{\mathrm{DPY}} = c_{\mathrm{VPA}} \quad .$$

The resulting three-way array gives the total intake of each vitamin, totaled across all dishes and all years, for each person using the recipes of a given authority.

The formation of this product is demonstrated in Figure 2, and can be viewed in one of two ways. It can be seen either as a single action which creates a nine-way tensor product that is then contracted on the shared indices (I, D, and Y), or as two successive steps, each involving the product of two arrays. In the two-step process, the flexibility of AIN allows *any* two of the three matrices to be multiplied together, in any order, and then the product multiplied by the third matrix (i.e., $(a_{\mathrm{VIY}} b_{\mathrm{IDA}}) d_{\mathrm{DPY}} = c_{\mathrm{VPA}}$ or $a_{\mathrm{VIY}} (b_{\mathrm{IDA}} d_{\mathrm{DPY}}) = c_{\mathrm{VPA}}$ or $b_{\mathrm{IDA}} (a_{\mathrm{VIY}} d_{\mathrm{DPY}}) = c_{\mathrm{VPA}}$; the subscript order is arbitrarily determined by the user).

An important observation to make regarding the two-step approach is that no matter how the procedure is done, the intermediate step will make substantive sense. Let us look at the intermediate products in the above example, the first of which is $(a_{\mathrm{VIY}} b_{\mathrm{IDA}}) = p_{\mathrm{VDYA}}$, which gives the vitamin content of each dish for each year, based on the recipes of each authority. The second intermediate product is $(b_{\mathrm{IDA}} d_{\mathrm{DPY}}) = q_{\mathrm{IPYA}}$, which gives the amount of each ingredient that would have been consumed by each person in any given year based on the recipes of each authority. Similarly meaningful, the third product, $(a_{\mathrm{VIY}} d_{\mathrm{DPY}}) = r_{\mathrm{VPDI}}$, gives the total amount (cumulated over the years covered) of each vitamin consumed by each person, broken down by each type of dish and by ingredients in that dish. This last product might be useful to a mother, for example, who is trying to decide whether to switch to a non-dairy ice-cream substitute when making certain desserts for her family. She could find the answer to questions like: "how much calcium has my daughter obtained from the milk content of the butterscotch sundaes that she is so fond of?".

(Note also that if totals are desired, any detailed breakdown can be summed across one or more modes by simply multiplying the array by the appropriate unit vector(s). For example, the breakdown by dish but not by ingredient would be given by the array product $r_{\mathrm{VPDI}}\mathbf{1}_{\mathrm{I}}$ (where $\mathbf{1}_{\mathrm{I}}$ is a vector of all ones whose length is equal to the size of the index set I), and the total vitamin intake of each person would be given by the two-way table $r_{\mathrm{VPDI}}\mathbf{1}_{\mathrm{I}}\mathbf{1}_{\mathrm{D}} = t_{\mathrm{VP}}$.)

'Folding in' an array to add information

There is another sort of array-combining operation that should be described; for lack of a better name, we call it 'folding in' an array to a product. It is best explained by an example.

For simplicity, let us go back to

$$a_{\mathrm{VIY}}b_{\mathrm{IDA}} = c_{\mathrm{VDYA}} \quad .$$

Suppose we are concerned that the vitamin content of an ingredient is reduced when it is cooked, and we know that the higher the cooking temperature the more is lost, with some vitamins being more heat sensitive than others. We obtain a table giving the loss of each vitamin as a function of cooking temperature (when other things like cooking time are held constant). In this matrix, an element g_{vt} is a value between 0.0 and 1.0 that represents the proportion of a dish's typical content of vitamin *v* that remains if the dish was cooked (for the recommended time) at a temperature that is raised *t* degrees above its standard cooking temperature*.

We want to incorporate this information as one additional mode of the array a_{VIY}. However, simple array-matrix multiplication gives us either

$$a_{\mathrm{VIY}}g_{\mathrm{VT}} = a_{\mathrm{IYT}}$$

or

$$a_{\mathrm{VIY}}g_{\mathrm{V'T}} = a_{\mathrm{VV'IYT}} \quad ,$$

neither of which accomplishes our goal. What we want is an array a_{VIYT}, and we can obtain it by using composite elements to do what is essentially matrix-vector multiplication, expressed as

$$a_{\mathrm{VIYT}} = (a_{\mathrm{vIY}}\ g_{\mathrm{vT}})_{\mathrm{V}} \quad .$$

To some readers, this might seem too far removed from our tensor-like foundations. However, it can be rewritten using a more complex expression that involves only standard tensor products and contractions as

$$a_{\mathrm{VIYT}} = a_{\mathrm{VIY}}\ g_{\mathrm{V'T}}\ \delta_{\mathrm{VV'}} \quad ,$$

* In this example, only temperatures above a dish's standard temperature are considered. If lower-than-standard temperatures were also included, values of g_{vt} greater than one would occur as well.

where δ is the (three-way) generalized Kronecker delta (i.e., the elements $\delta_{vv'}$ are one where v=v′ and zero otherwise).

Appendix II. Partitioned Matrices and Arrays

Partitioned Matrices in matrix notation and AIN

Partitioned matrices can be represented clearly, if somewhat awkwardly, in matrix notation. Parallel capabilities were not deliberately built into AIN, but as it turns out, composite elements (Rule 3) can be used as a compact way of denoting such matrices.

Partitioned matrices as implicit higher-way arrays

We define a 'partitioned matrix' in the usual way: a matrix in which subsets of the rows and columns are blocked off from others by boundaries or 'partitions', as in

$$\mathbf{A} = \left[\begin{array}{cc|cc|cc} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} \\ \hline a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} \end{array}\right] .$$

In contrast, **A** can be represented in AIN by using composite elements as

$(a_{\mathrm{IJmn}})_{\mathrm{MN}}$,

where I=J=2, M=2 and N=3 (or as $a_{(\mathrm{IM})(\mathrm{JN})}$ if the partitioning is ignored). This also shows how **A** may be viewed as a 'supermatrix', a matrix whose elements are themselves matrices, as in

$$\mathbf{A} = \left[\begin{array}{c|c|c} \mathbf{A}_{11} & \mathbf{A}_{12} & \mathbf{A}_{13} \\ \hline \mathbf{A}_{21} & \mathbf{A}_{22} & \mathbf{A}_{23} \end{array}\right]$$

where, for example, one element (or 'submatrix') is

$$\mathbf{A}_{21} = \begin{bmatrix} a_{31} & a_{32} \\ a_{41} & a_{42} \end{bmatrix} .$$

Partitioned matrices arise in various contexts. Often they have served as another device for incorporating higher-way classifications into matrix form. For example, $(a_{\mathrm{IJmn}})_{\mathrm{MN}}$ can be written as a four-way array a_{IJMN} if the composite elements are eliminated by replacing each lower case subscript inside the parentheses with the corresponding index set name from outside. This procedure is the converse of some earlier examples, where we started with a higher-way array and used composite subscripts to represent subarrays thereof.

Multiplication of partitioned matrices

It is well known that the rules of matrix multiplication used for scalar elements can be applied in the same way to the matrix elements of supermatrices. For example, matrix notation shows the multiplication of two supermatrices **A** and **B** as

$$\left[\begin{array}{c|c|c} \mathbf{A}_{11} & \mathbf{A}_{12} & \mathbf{A}_{13} \\ \hline \mathbf{A}_{21} & \mathbf{A}_{22} & \mathbf{A}_{23} \end{array}\right]\left[\begin{array}{c|c} \mathbf{B}_{11} & \mathbf{B}_{12} \\ \hline \mathbf{B}_{21} & \mathbf{B}_{22} \\ \hline \mathbf{B}_{31} & \mathbf{B}_{32} \end{array}\right]=\left[\begin{array}{c|c} \mathbf{C}_{11} & \mathbf{C}_{12} \\ \hline \mathbf{C}_{21} & \mathbf{C}_{22} \end{array}\right],$$

where

$$\left[\begin{array}{c|c} \mathbf{C}_{11} & \mathbf{C}_{12} \\ \hline \mathbf{C}_{21} & \mathbf{C}_{22} \end{array}\right]=\left[\begin{array}{c|c} \mathbf{A}_{11}\mathbf{B}_{11}+\mathbf{A}_{12}\mathbf{B}_{21}+\mathbf{A}_{13}\mathbf{B}_{31} & \mathbf{A}_{11}\mathbf{B}_{12}+\mathbf{A}_{12}\mathbf{B}_{22}+\mathbf{A}_{13}\mathbf{B}_{32} \\ \hline \mathbf{A}_{21}\mathbf{B}_{11}+\mathbf{A}_{22}\mathbf{B}_{21}+\mathbf{A}_{23}\mathbf{B}_{31} & \mathbf{A}_{21}\mathbf{B}_{12}+\mathbf{A}_{22}\mathbf{B}_{22}+\mathbf{A}_{23}\mathbf{B}_{32} \end{array}\right].$$

This equivalence is also reflected in AIN, which would represent the above multiplication as $a_{\mathrm{MN}}b_{\mathrm{NP}}=c_{\mathrm{MP}}$ (M=P=2, N=3). Viewing the problem at the level of the scalar elements of the submatrices, the same multiplication may be written as

$$a_{(\mathrm{IJmn})\mathrm{MN}}b_{(\mathrm{JKnp})\mathrm{NP}}=c_{(\mathrm{IKmp})\mathrm{MP}}$$

(assuming each submatrix in **A** is I by J and each submatrix in **B** is J by K).

Still at the level of the scalar elements of the submatrices, we could also consider the above multiplication to be a four-way product of two four-way arrays, that is, $a_{\mathrm{IJMN}}b_{\mathrm{JKNP}}=c_{\mathrm{IKMP}}$. Or, ignoring all the partitioning, we would have $a_{(\mathrm{IM})(\mathrm{JN})}b_{(\mathrm{JN})(\mathrm{KP})}=c_{(\mathrm{IM})(\mathrm{KP})}$. In practice, the choice of representation depends on which best reflects the theoretical structure of the problem at hand.

Partitioned higher-way arrays

It is straightforward to generalize from matrix to array partitioning. Array partitioning may be interpreted as simply dividing an array into regions, but an equivalent perspective is that *arrays of arrays* are constructed. That is, *n*-way arrays can be built using *m*-way (sub)arrays as elements, with the size of *m* unrestricted relative to *n*.

Partitioning as a device to deal with higher-way arrays within a two-way context is unnecessary in AIN, as we have seen, but sometimes it may provide useful theoretical or conceptual insight into how operations at a higher scale are related to those at a smaller more familiar scale. Therefore, we examine the generalization of partitioning to *n*-way arrays (keeping in mind Reference [5]'s point that most often how we visualize the

arrangement of the elements is irrelevant; this will become increasingly apparent as we consider equivalent representations below.)

A partitioned four-way array

Consider a four-way array that is partitioned into four-way subarrays, as illustrated in Figure 3 on the left. Here we have $a_{(\mathrm{IL})(\mathrm{JM})\mathrm{KN}}$, which after partitioning, has the structure $(a_{\mathrm{IJKNlm}})_{\mathrm{LM}}$. Given that *I*=6, *J*=2, *K*=4, *L*=2, *M*=3, and *N*=6, we see that originally it is 12x6x4x6, and after the partitioning is a 2x3 superarray composed of 6 elements that are 6x2x4x6 subarrays.

Note that, if the 'nesting' of subarrays within other arrays that is indicated by the partitioning notation is not meaningful for the current application, we might instead replace the partitioned array with the unpartitioned six-way array a_{IJKLMN} . For many purposes, this might be equally effective, but this does not reveal the structural relationships inherent in the partitioning. Still other groupings of the subscripts are possible, corresponding to other ways of conceptualizing the array object. For example, one variation is hierarchical partitioning, where we could represent the partitioned array as $((a_{\mathrm{IJKlm}})_{\mathrm{LMn}})_{\mathrm{N}}$. It is not particularly meaningful in this context, and we present it only to give the reader an idea of what other possibilities there are.

Multiplication of n-way partitioned arrays

Figure 3 illustrates the result of multiplying two partitioned arrays together while maintaining the partitioned structure. It is written as

$$a_{(\mathrm{IJKNlm})\mathrm{LM}} b_{(\mathrm{JPKmq})\mathrm{MQ}} = c_{(\mathrm{IPNlq})\mathrm{LQ}} \quad .$$

This views the problem as an *L*x*M* supermatrix (composed of subarrays that are *I*x*J*x*K*x*N*) multiplied by an *M*x*Q* supermatrix (composed of subarrays that are *J*x*P*x*K*) to produce an *L*x*Q* product (composed of arrays that are *I*x*P*x*N*).

As a check, let us represent the multiplication ignoring the partitioning in the arrays. This perspective is equivalent to multiplying the scalar elements of the arrays together. Now we have

$$a_{(\mathrm{IL})(\mathrm{JM})\mathrm{KN}} b_{(\mathrm{JM})(\mathrm{PQ})\mathrm{K}} = c_{(\mathrm{IL})(\mathrm{PQ})\mathrm{N}} \quad .$$

Here we see the product as one big three-way array, rather than a set of smaller three-way arrays.

Unless there is some reason to retain the partitioning as shown, it is of course simpler to represent the partitions as more 'ways' in the arrays. Then the situation involves a six-way array times a five-way array, which results in a five-way product, and is given by

$$a_{\mathrm{IJKNLM}} b_{\mathrm{JPKMQ}} = c_{\mathrm{ILPQN}} \quad .$$

Table I. Objects: Arrays and subarrays

	Object	**Standard matrix notation**	**Array index notation**
1	Scalar	a	a
2	Vector	$\mathbf{a}$	a_{I}
3	Matrix	$\mathbf{A}$	a_{IJ}
4	N-way array	$\underline{\mathbf{A}}$	$a_{\mathrm{IJK...}}$
5	Array with multi-character name	___	$\underline{std}_{\mathrm{II'}}$
6	Matrix element	a_{ij}	a_{ij}
7	Column vector in matrix	$\mathbf{a}_j$	a_{Ij}
8	Row vector in matrix	$\mathbf{a}_i$	a_{iJ}
9	Row 2 in matrix (as column vector)	$\mathbf{a}_2$	a_{2J}
10	Column 2 in matrix (as column vector)	$\mathbf{a}_2$	a_{I2}
11	Column 2 in matrix (as row vector)	$\mathbf{a}_2'$	a_{I2}
12	3-way array	$\underline{\mathbf{X}}$	x_{IJK}
13	kth (frontal) slab in array	$\mathbf{X}_k$	x_{IJk}
14	4th (horizontal) slab in array	$\mathbf{X}_4$	x_{4JK}
15	4th (lateral) slab in array	$\mathbf{X}_4$	x_{I4K}
16	Array fiber (column vector)	$\mathbf{x}_{ik}$	x_{iJk}
17	4-way array	$\underline{\mathbf{Y}}$	y_{IJRS}
18	3-way subarray of $\underline{\mathbf{Y}}$ at level 'r' of third mode (Mode C)	$\underline{\mathbf{Y}}_r$	z_{IKrT}

19	2-way subarray of $\underline{\mathbf{Y}}$ at level 'i' of Mode A and for a level of Mode D specified by the value in variable *u* (i.e., t=*u*)	$\mathbf{Y}_{iu}$	$y_{\mathrm{iKR}u}$

Table II. Matrix products

	Product	**Matrix notation**	**Array index notation**
1	Two scalars	wa	wa
2	Scalar and vector	$w\mathbf{a}$ or $\mathbf{a}w$	wa_{I} or $a_{\mathrm{I}}w$
3	Inner product of two vectors	$\mathbf{c}'\mathbf{d}=e$	$c_{\mathrm{I}}d_{\mathrm{I}}=e$
4	Outer product of two vectors	$\mathbf{ab}'=\mathbf{C}$	$a_{\mathrm{I}}b_{\mathrm{J}}=c_{\mathrm{IJ}}$
5	Outer product of vector with itself	$\mathbf{aa}'=\mathbf{G}$	$a_{\mathrm{I}}a_{\mathrm{I}'}=g_{\mathrm{II}'}$
6	Two matrices	$\mathbf{AB}=\mathbf{C}$	$a_{\mathrm{IJ}}b_{\mathrm{KJ}}=c_{\mathrm{IK}}$
7	Two matrices (second is square)	$\mathbf{AQ}=\mathbf{C}$	$a_{\mathrm{IJ}}q_{\mathrm{JJ}'}=c_{\mathrm{IJ}'}$
8	Two matrices (second is square and multiplied rowwise)	$\mathbf{AQ}'=\mathbf{C}$	$a_{\mathrm{IJ}'}q_{\mathrm{JJ}'}=c_{\mathrm{IJ}}$
9a	Three matrices	$\mathbf{ABC}=\mathbf{H}$	$a_{\mathrm{IJ}}b_{\mathrm{JK}}c_{\mathrm{KQ}}=h_{\mathrm{IQ}}$, $a_{\mathrm{IJ}}b_{\mathrm{KJ}}c_{\mathrm{KQ}}=h_{\mathrm{IQ}}$, etc.
9b	Three matrices (one repeated)	$\mathbf{AGA}'=\mathbf{C}$	$a_{\mathrm{IJ}}\ g_{\mathrm{JJ}'}\ a_{\mathrm{I}'\mathrm{J}'}\ =\ c_{\mathrm{II}'}$
10	Higher-way arrays	_____	$a_{\mathrm{IJK}}b_{\mathrm{JKL}}=c_{\mathrm{IL}}$
11	Array - subarray	_____	$a_{\mathrm{IjK}}b_{\mathrm{jKL}}=c_{\mathrm{IjL}}=d_{\mathrm{IL}}$

12	More complex linkages (e.g., for new kinds of models)	_____	$a_{\mathrm{IRS}} b_{\mathrm{JST}} c_{\mathrm{KRT}} = x_{\mathrm{IJK}}$

Table III. Two-way factor models

Model	**Matrix notation**	**Array index notation**
Direct fit to data	$\mathbf{X} = \mathbf{AB}'$	$x_{IJ} = a_{IR} b_{JR}$
Indirect fit (via covariances)	$\mathbf{C} = \mathbf{A\Phi A}'$	$cov_{II'} = a_{IR} \phi_{RR'} a_{I'R'}$
Orthogonal indirect fit	$\mathbf{C} = \mathbf{AA}'$	$cov_{II'} = a_{IR} a_{I'R}$

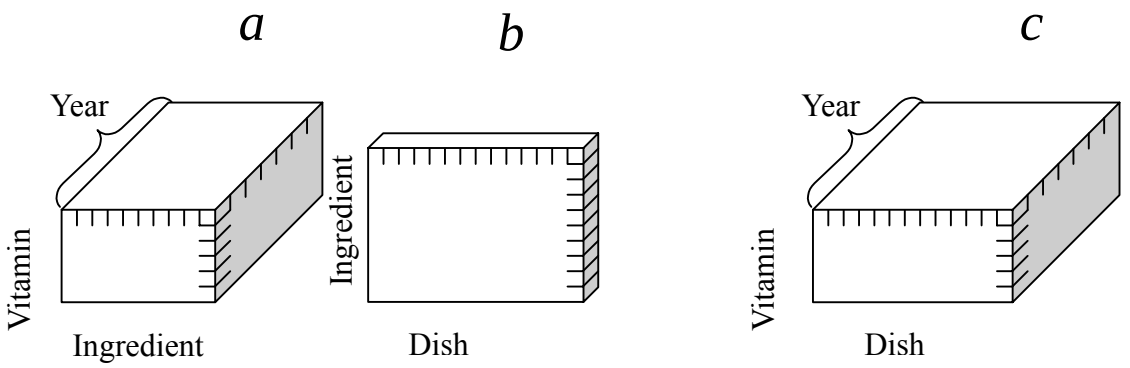
Table IV. Parafac1 and Parafac2

Model	**Matrix notation**	**Array index notation**
Parafac1 (for full array)	_____	$x_{IJK} = a_{IR} b_{JR} c_{KR}$ $x_{IJK} = a_{IR}\ b_{JR'}\ c_{KR''}\ \delta_{RR'R''}$ $x_{IJK} = (a_{Ir} b_{Jr} c_{Kr})_R 1_R = (a_{ir} b_{jr} c_{kr})_{IJKR} 1_R$
Parafac1 (for a representative slice)	$\mathbf{X}_k = \mathbf{A}\mathbf{D}_k\mathbf{B}'$	$x_{IJk} = a_{IR} d_{RR'k} b_{JR'}$ $x_{IJk} = a_{IR} (d_{rr})_{kR} b_{JR}$ $x_{IJk} = a_{IR} c_{kR} b_{JR}$ **or** $x_{IJk} = a_{IR} b_{JR} c_{kR}$
Parafac1 (for array unfolded horizontally)	$\mathbf{X}^{(IxJK)} = \mathbf{A}\ (\mathbf{C} \odot \mathbf{B})'$	$x_{I(JK)} = a_{IR} (b_{jr} c_{kr})_{(JK)R}$
Parafac1 (for array unfolded vertically)	$\mathbf{X}^{(IJxK)} = (\mathbf{A} \odot \mathbf{B})\ \mathbf{C}'$	$x_{(IJ)K} = (a_{ir} b_{jr})_{(IJ)R} c_{KR}$
Parafac2 (for full array)	_____	$cov_{II'K} = a_{IR} (c_{kr} \phi_{rr'} c_{kr'})_{RR'K} a_{I'R'}$
Parafac2 (for a representative slice)	$\mathrm{Cov}_k = \mathbf{A}\mathbf{D}_k\mathbf{\Phi}\mathbf{D}_k\mathbf{A}'$	$cov_{II'k} = a_{IR} c_{kR} \phi_{RR'} c_{kR'} a_{I'R'}$

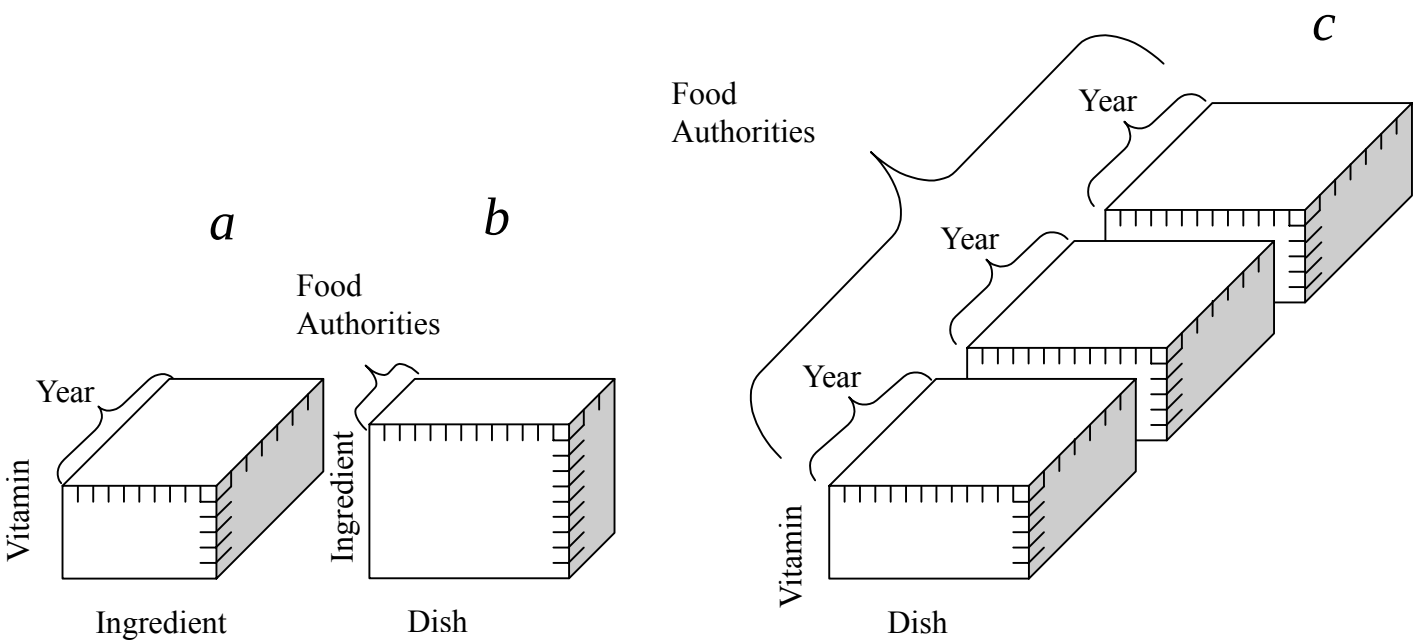
Table V. Tucker Models

Model	**Array index notation**
T4	$x_{IJKL} = a_{IR}\, b_{JS}\, c_{KT}\, d_{LU}\, g_{RSTU}$
T3	$x_{IJK} = a_{IR} b_{JS} c_{KT} g_{RST}$
T2	$x_{IJK} = a_{IR} b_{JS} g_{RSK}$
T1	$x_{IJK} = a_{IR} g_{RJK}$
'T0'	$x_{IJK} = g_{IJK}$
Symmetric T3 (e.g., for covariances)	$cov_{II'K} = a_{IR} a_{I'R'} c_{KT} g_{RR'T}$
Symmetric T2 (e.g., for MDS); also Carroll's IDIOSCAL	$cov_{II'K} = a_{IR} a_{I'R'} g_{RR'K}$

Figure 1. Array-matrix and array-array multiplication.

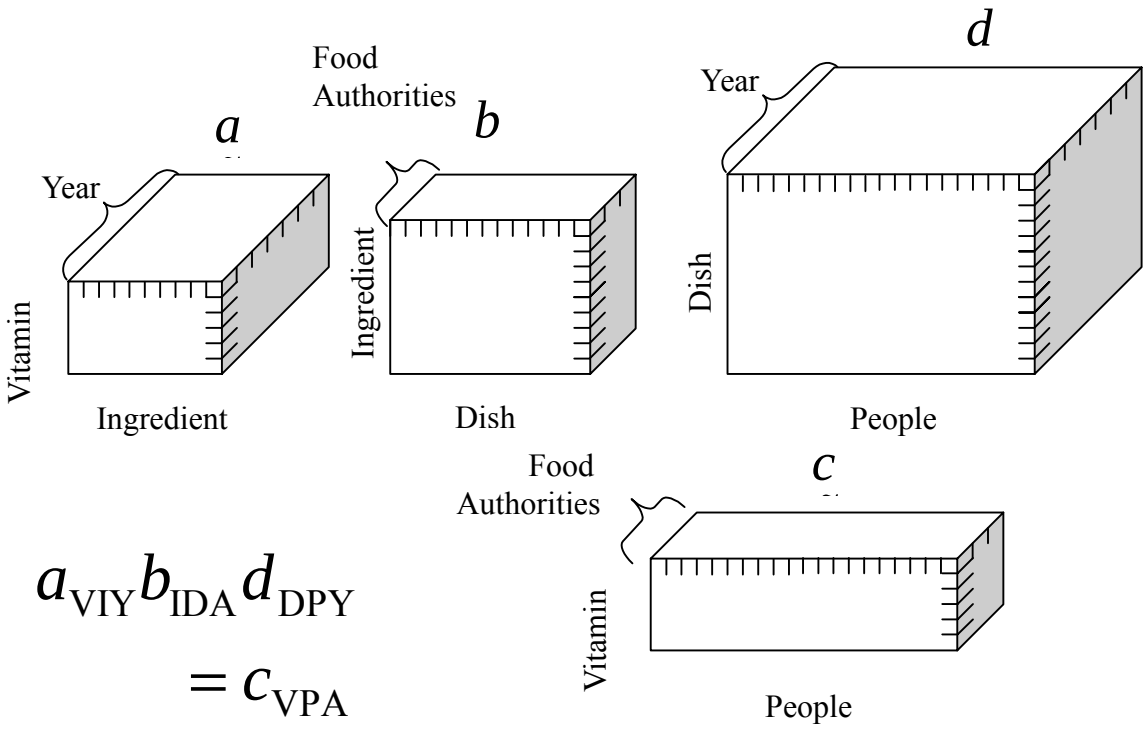


$$a_{\mathrm{VIY}} b_{\mathrm{ID}} = c_{\mathrm{VDY}}$$



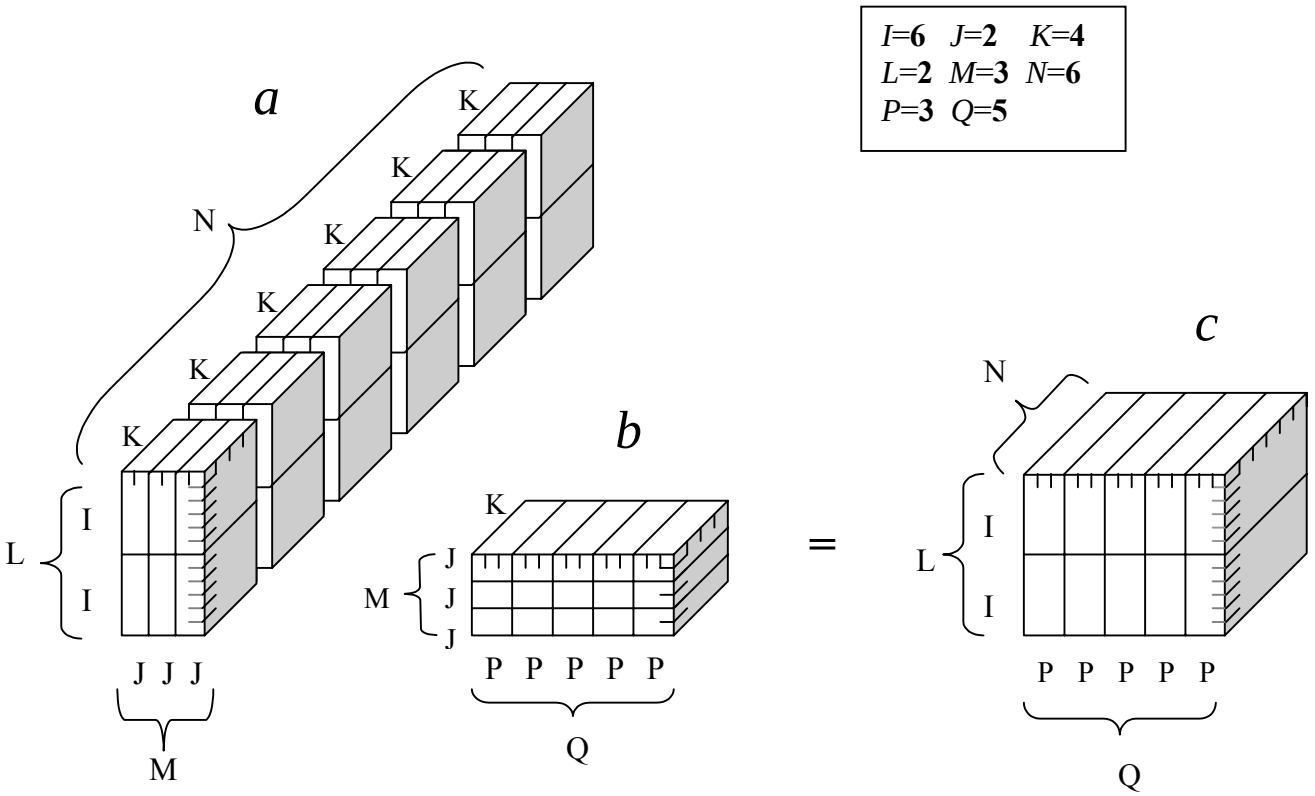
$$a_{\mathrm{VIY}} b_{\mathrm{IDA}} = c_{\mathrm{VDYA}}$$

Figure 2. A product of three arrays.



$$a_{\mathrm{VIY}} b_{\mathrm{IDA}} d_{\mathrm{DPY}} = c_{\mathrm{VPA}}$$

Figure 3. A product of two partitioned arrays.



$$a_{(\mathrm{IJKNlm})\mathrm{LM}}\, b_{(\mathrm{JPKmq})\mathrm{MQ}} = c_{(\mathrm{IPNlq})\mathrm{LQ}}$$