econometrics

MDPI

Article

Multidimensional Arrays, Indices and Kronecker Products

D. Stephen G. Pollock

Department of Economics, University of Leciceter, Leciceter LE1 7RH, UK; stephen_pollock@sigmapi.u-net.com

Abstract: Much of the algebra that is associated with the Kronecker product of matrices has been rendered in the conventional notation of matrix algebra, which conceals the essential structures of the objects of the analysis. This makes it difficult to establish even the most salient of the results. The problems can be greatly alleviated by adopting an orderly index notation that reveals these structures. This claim is demonstrated by considering a problem that several authors have already addressed without producing a widely accepted solution.

Keywords: multilinear algebra; Kronecker products; tensor products; index notation

Citation: Pollock, D. Stephen G. 2021. Multidimensional Arrays, Indices and Kronecker Products. *Econometrics* 9: 18. https://doi.org/10.3390/econometrics9020018

Academic Editor: Marc S. Paolella

Received: 18 November 2020
Accepted: 27 April 2021
Published: 28 April 2021

1. Introduction

The algebra of the Kronecker product is essential to multivariate statistical analysis. It is involved in the formulation of multivariate statistical models and in the derivation of their estimating equations. Such derivations commonly require the differentiation of functions of matrices in respect of their matrix arguments. This generates expressions that demand the use of the Kronecker product (see Pollock (1979, 1985) and Magnus (2010) for examples).

The necessary algebraic results have become available in a variety of texts that have arisen over a long period. Early texts on the subject were provided by Balestra (1976), Rogers (1980) and Graham (1981). Other accounts were given by Pollock (1979) and by Magnus and Neudecker (1988). More recently, there have been the texts of Steeb (1997) and of Turkington (2002).

Given that the subject is so well served, one might wonder what scope exists for further contributions. The repository of specialised results is extensive and most of what is needed is available, if looked for diligently. Nevertheless, the subject remains problematic. In many respects, it lacks sufficient transparency to allow easy access to its users, who are liable to call upon it only to satisfy their occasional needs.

Much of the problem lies in the notational difficulty of the subject. This is not primarily a matter of notational complexity. It is a more a result of the use of a notation that conceals the underlying structure of the objects of the analysis. A litany of results has been created that are not readily proven and that are mostly immemorable.

The contention that underlies this paper is that many of the difficulties can be relieved by adopting an appropriate notation involving an explicit use of indices. In the context of a treatise on differential geometry, Cartan (1952) famously decried the confusion that can arise from what he described as *une debauche d'indices*. He suggested that a clearer understanding of the geometry could be achieved by appealing to abstract concepts. This paper offers a different opinion. An orderly index notation is an indispensable aid to efficient computation. It assists rather that impedes ready access to vector space interpretations.

The continued adherence of econometricians and statisticians to the conventional notation of matrix algebra, when dealing with Kronecker products, is explained by one of their essential purposes, which is to cast complex multivariate models into the formats of simpler models for which there exist well developed theories. An early example of the use of the Kronecker product in econometrics was provided by Zellner (1962) in his treatment of the so-called seemingly unrelated regression equations. His treatment enabled

the system of equations to be cast in the format of a single multiple regression equation, which enabled him to exploit the existing results pertaining to such equations.

This paper is concerned primarily with matters of computation. Therefore, the next section considers the alternative ways in which computers handle multidimensional arrays. Section 3 presents an index notation, which is designed to assist in computations, and it provides an analysis of the Kronecker product, which is the matrix of a tensor product with respect to a natural basis.

Section 4, which may be omitted without interrupting the flow of the exposition, describes the formal algebra of the tensor product, and it mentions some of its applications. Sections 5 and 6 deal with a particular problem concerning multiple Kronecker products, which, it is believed, will serve to illustrate the power of the index notation.

2. The Multidimensional Framework and the Computer

The natural framework for a multi-dimensional array is a set of points in the positive orthant of a multi-dimensional space that have integer-valued co-ordinates. These co-ordinates will serve as the indices of the elements of the array.

The simplest array is a linear array of one dimension, which is described as a vector. Next is a rectangular array of two dimensions, which is a described as a matrix. In three dimensions, there is a cubic array, and the progression continues with hypercubes of increasing dimensions. However, there is an obvious difficulty that limits the dimensions of a practical representation to no more than two. The elements in the interior of a cube are hidden from view, and an object of more than three dimensions is incapable of visualisation.

The only practical recourse for making higher-dimensional arrays accessible and amenable to manipulations is to map them into a linear array or into a matrix. Such a mapping establishes an ordering within the lower-dimensional array of the elements of the higher-dimensional array.

The simplest examples of such mappings are the ways in which a matrix $A = [a_{ij}; i = 1, 2, \ldots, M, j = 1, 2, \ldots, N]$ of M rows and N columns can be mapped into a vector. The row-major mapping joins successive rows of the matrix in a wide row vector. Here, the index j passes through N values for each increment of the index i. The index that locates the element a_{ij} within the vector is given by $(i-1)N + j$. The alternative column-major mapping joins successive columns of the matrix in a long column vector, and then the index that locates the element a_{ij} is given by $(j-1)M + i$.

These two mappings are commonly employed in assigning the elements of a matrix to contiguous cells of the memory of a computer. Whereas the *Pascal* and *C* compilers adopt the row-major principle, the FORTRAN compiler and the MATLAB interpreter adopt the column-major principle.

The space to store a matrix A of order M times N in the memory of a computer is commonly provided by a statement such as

$$A : \textbf{array}\ [1, \ldots, M][1, \ldots, N]. \tag{1}$$

Regardless of which of the two mappings has been used to store the matrix, its elements can be accessed and displayed readily in either order, and matrix can also be reconstituted and displayed in its original form. Thus, with the help of nested do-loops, the matrix can be displayed via a code such as

```
for i := 1 to M do
  begin {i}
    for j := 1 to N do
      begin {j}
        Write(a[i][j]);
        SkipSpace;
      end; {j}
    SkipLine;
  end; {i}
```
(2)

the row-major form would be displayed via

```
for i := 1 to M do
  for j := 1 to N do
    begin
      Write(a[i][j]);
      SkipSpace;
    end;
```
(3)

and the column-major form would be displayed by reversing the order of the do-loops:

```
for j := 1 to N do
  for i := 1 to M do
    begin
      Write(a[i][j]);
      SkipLine;
    end;
```
(4)

The extension to arrays of higher dimensions is straightforward. Consider a cubic array

$$A : \textbf{array}\ [1, \ldots, M][1, \ldots, N][1, \ldots, P], \tag{5}$$

in which the elements a_{ijk} are indexed by $i = 1, \ldots, M$, $j = 1, \ldots, N$, and $k = 1, \ldots, P$. On the assumption that the index k in now the most rapidly changing, the row-major mapping would give the element a position corresponding to $(i-1)NP + (j-1)P + k$, whereas, with i as the most rapid index, the column-major mapping would give it the position corresponding to $(k-1)MP + (j-1)P + i$.

It should be noted that, if the initial values of i, j and k are 1, and if $M = N = P = 10$, then, in the row-major mapping, it might be declared that k stands for units, j stands for tens and i stands for hundreds. Then, the formula for the position corresponds to how we count in base 10. More generally, if M, N and P were to take different values, then the index would correspond to a mixed radix representation of a number. As it is, with the initial values at one, we are liable to describe ijk within the row-major mapping as the index of a lexicogoraphic or dictionary ordering.

A three-dimensional array has three indices. Two of these indices will serve to define the framework of a matrix and the third index will serve to define the order of the resulting matrices. The indices can be assigned to these roles in six different ways. To keep track of the elements of a multidimensional array, it is clear that an index notation is called for. This will be provided in the next section together with the definitions of a few of the objects that arise most frequently in the processes of statistical computing. These objects are liable to be compounded from objects of lower dimensions. This gives rise to structures that one may be able to exploit in pursuit of computational efficiency.

To achieve computational efficiency, it is necessary to avoid displacing the values that are stored in the memory of the computer. They must be left in place, regardless of the changing nature of the structures to which they may give rise, whether these are vectors or matrices or the products of matrices. Such objects can be created by the manipulation of

pointers, which are variables that contain the addresses of the memory locations in which the values are stored.

It is notable that do-loops are implemented by pointers, which accounts for the ease with which their order can be altered within a nested sequence. Thus, whichever system of storage is used, one can rely on do-loops (or pointers) to pluck the elements from the memory in the orders that are required.

3. The Index Notation

The object that will be analysed in detail in subsequent sections of the paper is a threefold Kronecker product postmultuplied by a long vector. It is appropriate already to display this object, which may be denoted by

$$Y^v = (A \otimes B \otimes C)X^v. \tag{6}$$

Its factors are as follows:

$$\begin{aligned} A = \quad & [a_{jp}] = (a_{jp}e_j^p), \qquad B = [b_{kq}] = (b_{kq}e_k^q), \\ C = \quad & [c_{\ell r}] = (c_{\ell r}e_\ell^r), \qquad X^v = (x_{pqr}e_{pqr}). \end{aligned} \tag{7}$$

In these definitions, both the brackets $[,]$ and the parentheses $(,)$ are meant to signify that each index is varying over its designated range. The lower limits of the indices are unity, and their upper limits may be denoted by the corresponding capital letters, so that, for example, $j = 1, \ldots, J$. The alternative notation for the matrices and for the long vector X^v has been taken from a paper of Pollock (1985). It expresses the matrices as weighted combinations of the corresponding basic matrices. Thus, for example,

$$A = (a_{jp}e_j^p) = \sum_j \sum_p a_{jp}e_j^p, \tag{8}$$

where e_j^p is a basic matrix with a unit in the jth row and the pth column and with zeros elsewhere (the parentheses indicate that summations are to be taken in respect of the indices j and p that are associated with the basic matrices and which are repeated within the accompanying scalar elements). The pth column of the matrix A is $(a_{jp}e_j) = \sum_j a_{jp}e_j$, whereas its jth row is $(a_{jp}e^p) = \sum_p a_{jp}e^p$. Here, as in Equation (8), the summations are over the basis elements, which are vectors in these cases. Summations occur only when there are repeated instances of an index.

There are alternative ways of arranging the elements of the matrix to define different objects. The transpose A' of the matrix together with the long column vector A^c and the wide row vector A^r, which are described, respectively, as its column-major and row-major forms, may be denoted as follows:

$$\begin{aligned} A' = \quad & (a_{jp}e_j^p)' = (a_{jp}e_p^j), \\ A^c = \quad & (a_{jp}e_j^p)^c = (a_{jp}e_{pj}), \\ A^r = \quad & (a_{jp}e_j^p)^r = (a_{jp}e^{jp}). \end{aligned} \tag{9}$$

Observe that, in the mappings $A \to A^c$ and $A \to A^r$, the migrating index moves ahead of the index or the string of indices that it is joining. In the mappings $A' \to A^c$ and $A' \to A^r$, the migrating index joins from behind.

Whereas the matrix elements a_{jp} will remain fixed in the computer memory in the order indicated by their indices, the nature of a derived multidimensional object is indicated by the positioning of the indices on the basic matrices or vectors. Thus, e_j^p denotes a basic matrix with a unit in the jth row and the pth column, whereas e_{pj} is a long column vector with a unit in the $\{(p-1)J + j\}$th position and e^{jp} is a wide row vector with a unit in the $\{(j-1)P + p\}$th position. Elsewhere, there are only zero-valued elements. For the

basis vectors, the composite indices follow a lexicographic ordering, which accords with the conventional definition of a Kronecker product, as will be confirmed at the end of this section.

The translation from the elements that are held in the computer's memory to the various multidimensional objects should be by virtue of the pointer arithmetic that is associated with the basic arrays e_j^p, e_{pj} and e^{jp}. These matrices (or vectors) determine the format of the object or its "shape", which is the MATLAB terminology.

Observe that, in the case of the long column vector A^c, the elements are arrayed in the reverse of their lexicographic ordering, with the trailing index p as the leading classifier within the vector. In the case of the long column vector $X^v = (x_{pqr}e_{pqr})$ of (6), which is the transpose of the wide vector $X^r = (x_{pqr}e^{pqr})$, the elements follow the lexicographic ordering of the indices p, q, r that are associated with the basic vectors. It will also be necessary to consider the long vector $X^c = (x_{pqr}e_{rqp})$ in which the elements x_{pqr} are arrayed in reverse lexicographic order, which is the order adopted by the MATLAB program.

The composition of the matrix A with a matrix $D = [d_{pr}] = (d_{pr}e_p^r)$ may be denoted by

$$AD = (a_{jp}e_j^p)(d_{pr}e_p^r) = (\{a_{jp}d_{pr}\}_p e_j^r). \tag{10}$$

The final expression is derived by cancelling the superscripted column index p of the leading matrix with the subscripted row index p in the trailing matrix. The index p that is affixed to the right brace within the product is to emphasise that the elements within the braces are to be summed over the range of the repeated index. It may be omitted if the eyes can be relied on to recognise repeated indices. Thus

$$\{a_{jp}d_{pr}\}_p = \sum_p a_{jp}d_{pr}, \tag{11}$$

which is described as a contraction with respect to p.

The transformation $A'^c = \mathbb{C}A^c$ from $A^c = (a_{jp}e_{pj})$ to $A'^c = (a_{jp}e_{jp})$ is effected by the commutation matrix $\mathbb{C} = (e_{jp}^{pj})$. Thus

$$A'^c = (a_{jp}e_{jp}) = (e_{jp}^{pj})(a_{jp}e_{pj}) = \mathbb{C}A^c, \tag{12}$$

and it will be recognised that $A'^c = A^{r\prime} = A^v$, albeit that these identities do not extend to objects of higher dimensions.

It is easy to seen that $\mathbb{C}$ is an orthonormal matrix such that $\mathbb{C}' = \mathbb{C}^{-1}$. It has been given various names. Pollock (1979) has called it the tensor commutator, denoted by a capital T inscribed in a circle. Magnus and Neudecker (1979) described it as the commutation matrix, which they denoted by K, and Van Loan (2000) has described it as the shuffle matrix, denoted by S.

The notation for the Kronecker product of matrices is illustrated by the following example:

$$A \otimes B = (a_{jp}e_j^p) \otimes (b_{kq}e_k^q) = (\{a_{jp}b_{kq}\}e_{jk}^{pq}). \tag{13}$$

Here, the row indices j, k and the column indices p, q associated with the basic matrices both follow lexicographic orderings. The order of the matrices within a Kronecker product can be reversed by applying the tensor commutator to both the rows and the columns. Thus,

$$\begin{aligned} \mathbb{C}_{jk}(A \otimes B)\mathbb{C}'_{pq} &= B \otimes A \quad \text{or} \\ (e_{kj}^{jk})(\{a_{jp}b_{kq}\}e_{jk}^{pq})(e_{pq}^{qp}) &= (\{a_{jp}b_{kq}\}e_{kj}^{qp}). \end{aligned} \tag{14}$$

The subscripts on the $\mathbb{C}$ matrices indicate the indices that are to be commuted.

It is now possible to evaluate the product of (6). This can be expressed in the conventional matrix notation and in the index notation as follows:

$$\begin{aligned} Y^v &= (A \otimes B \otimes C)X^v \quad \text{or} \\ (y_{jk\ell}e_{jk\ell}) &= \{(a_{jp}e_j^p) \otimes (b_{kq}e_k^q) \otimes (c_{\ell r}e_\ell^r)\}(x_{pqr}e_{pqr}) \\ &= (\{a_{jp}b_{kq}c_{\ell r}\}e_{jk\ell}^{pqr})(x_{pqr}e_{pqr}). \end{aligned} \tag{15}$$

On replacing the row-major vector X^v by the column-major vector X^c and on reversing the order of the matrices, an alternative version of the product is created in the form of

$$\begin{aligned} Y^c &= (C \otimes B \otimes A)X^c \quad \text{or} \\ (y_{jk\ell}e_{\ell kj}) &= \{(c_{\ell r}e_\ell^r) \otimes (b_{kq}e_k^q) \otimes (a_{jp}e_j^p)\}(x_{pqr}e_{rqp}) \\ &= (\{a_{jp}b_{kq}c_{\ell r}\}e_{\ell kj}^{rqp})(x_{pqr}e_{rqp}). \end{aligned} \tag{16}$$

A series of commutation operations link the two forms. These forms might be regarded as generalisations of the equations

$$\begin{aligned} Y^v &= (AXB')^v = (A \otimes B)X^v, \\ Y^r &= (AXB')^r = X^r(A' \otimes B') \quad \text{and} \\ Y^c &= (AXB')^c = (B \otimes A)X^c, \end{aligned} \tag{17}$$

where $X^v = (x_{pq}e_{pq})$ and $X^c = (x_{pq}e_{qp})$, and where the other matrices are as defined previously. The equations of (17) tend to convey the idea that a multiple Kronecker product can be computed via pairwise matrix multiplications. To pursue such a strategy, it is necessary to cast the vectors X^v, X^r or X^c and their transforms into the shapes of matrices.

The various operations of reshaping that might be applied to the column-major vector $X^c = (x_{pqr}e_{rqp})$ would produce matrices that are denoted as follows:

$$X_p^c = (x_{pqr}e_p^{rq}), \quad X_q^c = (x_{pqr}e_q^{rp}), \quad X_r^c = (x_{pqr}e_r^{qp}). \tag{18}$$

Similar operations can be applied to the row-major vector $X^v = (x_{pqr}e_{pqr})$. It is important that the reshaping operations should be effected by manipulating pointers, instead of by moving the elements themselves.

The conversions $X^c \to X_p^c$ and $X^c \to X_r^c$ involve separating the trailing and leading elements, respectively, from the rest of the composite index rqp, whereby $e_{rqp} \to e_p^{rq}$ and $e_{rqp} \to e_r^{qp}$. These conversions will prove to be amenable to the `reshape` operator of MATLAB. To create X_q^c via MATLAB would require a commutation operation to bring the index q to the leading or trailing position. However, the creation of all of these matrices is readily achieved with do-loops that are nested appropriately. The MATLAB `reshape` operator is considered in detail in Section 6, and examples are provided in an appendix. In a recent paper, Fackler (2019) has show that some efficiency in computing the product of (6) can be gained by exploiting the `reshape` procedure of that program.

It may be helpful to demonstrate that the definition that has been provided in (13) for the Kronecker product $A \otimes B$ does correspond to the usual definition, which depends on displaying the subscripted elements within the context of their matrices. It is sufficient to consider the product of two square matrices of order 2 as follows:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \otimes \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} & a_{12}\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \\ a_{21}\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} & a_{22}\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \end{bmatrix} \tag{19}$$

$$= \begin{bmatrix} a_{11}b_{11} & a_{11}b_{12} & a_{12}b_{11} & a_{12}b_{12} \\ a_{11}b_{21} & a_{11}b_{22} & a_{12}b_{21} & a_{12}b_{22} \\ a_{21}b_{11} & a_{21}b_{12} & a_{22}b_{11} & a_{22}b_{12} \\ a_{21}b_{21} & a_{21}b_{22} & a_{22}b_{21} & a_{22}b_{22} \end{bmatrix}.$$

It can be seen that the composite row indices jk associated with the elements $a_{jp}b_{kq}$ follow the lexicographic sequence $\{11, 12, 21, 22\}$, as do the composite column indices pq, which is to say that the conventional definition of a Kronecker product is in accordance with the row-major scheme.

4. Tensor Products and Associated Notations

The majority of econometric texts that consider the Kronecker product make no mention of tensor products, notwithstanding the fact that a Kronecker product may be classified as a tensor product. An exception is the account of Pollock (1985), which defines tensors in the context of multilinear algebra.

There are two ways in which the algebra of tensors may be approached. The first way is via mathematical physics, where vectors are liable to subsist in three-dimensional or four-dimensional spaces. Tensors are used in expounding the theory of electrodynamics via Maxwell's equations—see Brau (2004)—the theory of special relativity—see Lawden (1967)—and the theory of the elasticity of solids—see Bower (2010). In such contexts, vectors and tensors are liable to acquire substantive connotations.

A second way of approaching tensors and tensor products is via the abstract theory of multilinear algebra that was propounded originally by Bourbaki (1948) and which has been expounded in the more recent texts of Greub (1967) and Marcus (1973). Such expositions take a coordinate-free approach that avoids specifying bases for the vector spaces and which does not preempt the choice of a metric.

The coordinate-free approach is well suited to a physical analysis that regards vectors as geometric objects that exist in space independently of any coordinate system or of its basis vectors. This flexibility is exploited in the theory of special relativity to enable geometric vectors to be viewed within alternative frames of reference. The natural basis is appropriate to an observer at rest within their own frame of reference. However, objects that are in motion relative to the observer require other coordinate systems, which are derived by applying the Lorentz transform to the natural basis. (see Marder (1968) and Susskind and Friedman (2018), for examples).

In the main, data analysts and computer scientists are liable to regard vectors as algebraic objects comprising an ordered set of elements, which are the coordinates of the vector relative to a natural basis consisting of orthogonal vectors of unit length comprised by an identity matrix. Our approach, in this section, is to express the basic definitions in terms of arbitrary vector spaces, denoted by $\mathcal{U}$ and $\mathcal{V}$ etc., and to interpret them immediately in terms of real finite-dimensional coordinate spaces. An n-dimensional real vector space will be denoted by $\mathcal{R}_n$.

If $\mathcal{V}$ is a vector space defined over a scalar field $\mathcal{F}$, then the dual space is the space of all linear functionals $\mathcal{V}^* = \mathcal{L}(\mathcal{V}, \mathcal{F})$ mapping from $\mathcal{V}$ to $\mathcal{F}$. Given that it is of the same dimension, the dual space $\mathcal{V}^*$ is isomorphic to the primal vector space $\mathcal{V}$. The relationship between $\mathcal{V}^*$ and $\mathcal{V}$ is reflexive or reciprocal, which is to say that the dual of the dual space is the primal space.

The dual of the vector space $\mathcal{R}_n$ may be denoted by $\mathcal{R}_n^*$. The elements of a coordinate vector space $\mathcal{R}_n$ are column vectors, which may be denoted by subscripted lowercase

symbols, such as x_1, y_2, whereas the elements of the corresponding dual space $\mathcal{R}_n^*$ are row vectors denoted by symbols with superscripts, such as x', x^*, x^1, y^2. Lowercase letters without subscripts, such as x, y, are liable to be regarded as column vectors. The inner product $x'y$ is a linear functional.

If $\mathcal{U}$ and $\mathcal{V}$ are vector spaces defined over a scalar field $\mathcal{F}$, then a mapping $\phi : \mathcal{U} \times \mathcal{V} \to \mathcal{F}$ from the Cartesian product of the spaces to the scalar field is a bilinear functional if, when $u_1, u_2 \in \mathcal{U}$ and $v_1, v_2 \in \mathcal{V}$ and when $\alpha, \beta \in \mathcal{F}$, there are

$$\begin{aligned} \phi(\alpha u_1 + \beta u_2, v_1) &= \alpha\phi(u_1, v_1) + \beta\phi(u_2, v_1), \\ \phi(u_1, \alpha v_1 + \beta v_2) &= \alpha\phi(u_1, v_1) + \beta\phi(u_1, v_2). \end{aligned} \tag{20}$$

The set of all such bilinear functionals is a vector space denoted by $\mathcal{L}(\mathcal{U}, \mathcal{V}; \mathcal{F})$. Familiar examples of the mapping $\mathcal{R}_n^* \times \mathcal{R}_n \to \mathcal{R}$ are as follows:

$$\begin{aligned} x'E_{ij}y &= (x_i e^i)(e_i^j)(y_j e_j) = (x_i e^i)(y_j e_i) = x_i y_j, \\ x'I_n y &= (x_i e^i)(\delta_{ij} e_i^j)(y_j e_j) = (x_i e^i)(\delta_{ij} y_j e_i) = \{x_i y_i\}_i, \\ x'Ay &= (x_i e^i)(a_{ij} e_i^j)(y_j e_j) = \{x_i a_{ij} y_j\}_{ij}, \end{aligned} \tag{21}$$

were $E_{ij} = (e_i^j)$ denotes a matrix with a unit in the ijth position and with zeros elsewhere. The second of these products may be written, more simply, as

$$x'y = (x_i e^i)(y_i e_i) = \{x_i y_i\}_i. \tag{22}$$

This is a bilinear functional, which can also be considered to be a linear functional, if one of the vectors is regarded as the means by which the other vector is mapped to the field $\mathcal{R}$.

Less familiar are the matrix forms of the mappings $\mathcal{R}_n \times \mathcal{R}_n \to \mathcal{R}$ and $\mathcal{R}_n^* \times \mathcal{R}_n^* \to \mathcal{R}$. Examples of these, in turn, are as follows:

$$A^r(x \otimes y) = (a_{ij} e^{ij})\{(x_i e_i) \otimes (y_j e_j)\} = (a_{ij} e^{ij})(x_i y_j e_{ij}) = \{a_{ij} x_i y_j\}_{ij} \tag{23}$$

and

$$(y' \otimes x')A^c = \{(y_j e^j) \otimes (x_i e^i)\}(a_{ij} e_{ji}) = (y_j x_i e^{ji})(a_{ij} e_{ji}) = \{y_j x_i a_{ij}\}_{ij}. \tag{24}$$

Reference to (17) will show that $A^r(x \otimes y) = (x'Ay)^r$ and that $(y' \otimes x')A^c = (x'Ay)^c$.

It is important to recognise that the vector space $\mathcal{L}(\mathcal{U}, \mathcal{V}; \mathcal{F})$ represents the means by which vectors $u \in \mathcal{U}$ and $v \in \mathcal{V}$ are mapped into the scalar field $\mathcal{F}$. It should be distinguished from the individual mappings of $\mathcal{U} \times \mathcal{V} \to \mathcal{F}$, each of which entails specific values of u and v. The distinction can be made clear by considering the functional $f = u'Av$, where $u = [u_1, \ldots, u_n]'$ and $v = [v_1, \ldots, v_m]'$ are column vectors and $A = [a_{ij}]$ is a matrix. The pair (u, v) are an element of the Cartesian product $\mathcal{U} \times \mathcal{V}$. The matrix A is an element of $\mathcal{L}(\mathcal{U}, \mathcal{V}; \mathcal{F})$, and f is a product of a specific mapping of $\mathcal{U} \times \mathcal{V} \to \mathcal{F}$.

A tensor product of vector spaces may be defined in a confusing variety of ways. However, it is appropriate to regard the tensor product $\mathcal{U} \otimes \mathcal{V}$ as unique linear mapping from the set $\mathcal{U} \times \mathcal{V}$. Thus, for example, Shephard (1966) defines it to be the dual of the vector space $\mathcal{L}(\mathcal{U}, \mathcal{V}; \mathcal{F})$ of bilinear functionals, so that each bilinear functional corresponds to a unique linear functional on $\mathcal{U} \otimes \mathcal{V}$. From the point of view of the real coordinate spaces, this amounts to nothing more than a change of notation, Nevertheless, what is revealed can be insightful.

A distinction must be made between the forms $\mathcal{U}^* \otimes \mathcal{V}$, $\mathcal{U}^* \otimes \mathcal{V}^*$ and $\mathcal{U} \otimes \mathcal{V}$. The first of these is liable to be described as a contravariant product, and it may be noted that $\mathcal{U}^* \otimes \mathcal{V} = \mathcal{V} \otimes \mathcal{U}^*$. The others may be described as covariant products, and it should be noted that the order of the two spaces is significant: $\mathcal{U} \otimes \mathcal{V} \neq \mathcal{V} \otimes \mathcal{U}$ and $\mathcal{U}^* \otimes \mathcal{V}^* \neq \mathcal{V}^* \otimes \mathcal{U}^*$. In the present context, *covariance* and *contravariance* denote relationships. However, in mathematical physics and in other contexts, they are regarded as attributes of the vector spaces.

If $x' \in \mathcal{R}_n^*$ and $y \in \mathcal{R}_n$, then their contravariant product is an element in $\mathcal{R}_n^* \otimes \mathcal{R}_n$ that has the form

$$x' \otimes y = y \otimes x' = (x_i e^i) \otimes (y_j e_j) = (x_i y_j e^i_j). \tag{25}$$

The corresponding covariant products in $\mathcal{R}_n \otimes \mathcal{R}_n$ and $\mathcal{R}_n^* \otimes \mathcal{R}_n^*$, respectively. are

$$x \otimes y = (x_i e_i) \otimes (y_j e_j) = (x_i y_j e_{ij}) \quad \text{and} \quad x' \otimes y' = (x_i e^i) \otimes (y_j e^j) = (x_i y_j e^{ij}). \tag{26}$$

These are elementary or decomposable products. Examples of non-decomposable tensor products of the three varieties are

$$\begin{aligned} A &= \textstyle\sum_{ij} a_{ij}(e_i \otimes e^j) = (a_{ij} e^j_i), \\ A^c &= \textstyle\sum_{ij} a_{ij}(e_j \otimes e_i) = (a_{ij} e_{ji}), \\ A^r &= \textstyle\sum_{ij} a_{ij}(e^i \otimes e^j) = (a_{ij} e^{ij}). \end{aligned} \tag{27}$$

These would become decomposable products if $a_{ij} = \alpha_i \beta_j$ for all i, j, albeit that a decomposable product may be the product of two or more non-decomposable products.

It is when coordinates are attributed to physical vectors in 3-dimensional space or in 4-dimensional spacetime that the terms covariant and contravariant come to denote attributes as opposed to reciprocal relationships. The usage can be demonstrated by considering a change of bases. Let $W = [w_1, w_2, \ldots, w_n]$ and $M = [m_1, m_2, \ldots, m_n]$ be alternative bases of a vector space $\mathcal{V}$. Then, there exists transformations $M = WA$ and $W = MA^{-1}$ that transform one basis into the other.

Consider a vector expressed as $v = v_1 e_1 + v_2 e_2 + \cdots + v_n e_n$ in terms of the natural basis $I_n = [e_1, e_2, \ldots, e_n]$ and, alternatively, as $v = Wx = My$, where $x = [x_1, x_2, \ldots, x_n]'$ and $y = [y_1, y_2, \ldots, y_n]'$ are the coordinate vectors relative to the bases W and M, respectively. Then, since $M = WA$, there are $Wx = WAy$, whence $x = Ay$ and $y = A^{-1}x$. Thus, the bases and the corresponding coordinates transform as follows:

$$\left.\begin{matrix} W \\ x \end{matrix}\right\} \rightarrow \left\{\begin{matrix} M = WA \\ y = A^{-1}x. \end{matrix}\right. \tag{28}$$

Let $A = (\lambda_i e^i_i)$ be a diagonal matrix. Then, the ith column of the transformed basis M is

$$m_i = (m_{ij} e_j) = (w_{ij} e^j_i)(\lambda_j e_j) = (\{w_{ij}\lambda_j\}_j e_i) \tag{29}$$

and the vector of coordinates relative the basis vectors of M is

$$y = (y_i e_i) = (\lambda_i^{-1} e^i_i)(x_i e_i) = (\lambda_i^{-1} x_i e_i). \tag{30}$$

They have a contravariant relationship with the basis vectors. The effect of halving the length of a basis vector will be the doubling of the value of an associated coordinate.

In general, vectors that change their scale inversely to changes in the scale of the reference axes (i.e. the basis vectors) are commonly described as contravariant. Dual vectors that change in the same way as the changes to scale of the reference axes are described as covariant. If $v = v_1 e_1 + v_2 e_2 + \cdots + v_n e_n = (v_i e_i)$ denotes a velocity vector, then its coefficients $[v_1, v_2, \ldots v_n]$, which have the dimension of *distance/time*, can said to constitute a contravariant vector. By contrast, a gradient vector of a dimension *quantity/distance* will be a covariant vector.

In the Einstein notation, which suppresses the summation sign, contravariant components carry upper indices, and the velocity vector will be denoted, typically, by

$$\mathbf{v} = v^1 \mathbf{e}_1 + v^2 \mathbf{e}_2 + \cdots + v^n \mathbf{e}_n = v^i \mathbf{e}_i, \tag{31}$$

where $\mathbf{e}_i$ denotes a basis vector. The understanding is that, whenever an index is repeated within an expression, there must be a corresponding summation. This easement of the

notation, which dispenses with the summation signs, relies on an understanding of the context of the expression.

The notation of the present paper adopts the same summation convention for expressions that are found within parentheses. Thus, $I_n = (\delta_{ij}e_i^j)$ denotes an identity matrix. However, in the absence of the parentheses, $a_{ij}e_i^j$ denotes a matrix with a_{ij} in the ith row and the jth column and with zeros elsewhere. The expression (e_i^j) from (21) denotes a matrix with a unit in that position and with zeros elsewhere. Since there are no repeated indices, the parentheses do not imply a summation.

It should be noted that, in this notation, scalar elements never carry superscripts. Furthermore, the notation refers invariably to algebraic or coordinate vectors. It does not encompass coordinate-free geometric vectors, which are common in physical contexts.

The diversity of the existing tensor notations is too great to be summarised here. However, numerous references to alternative notations have been given by Harrison and Joseph (2016). Mention should also be made of De Lathauwer et al. (2000) and Bader and Kolda (2006), who have provided accounts that have been widely referenced by computer scientists.

5. The Problem at Hand

In terms of the index notation, the product of (6) may be expressed as

$$\begin{aligned}(A \otimes B \otimes C)X^v &= (\{a_{jp}b_{kq}c_{\ell r}\}e_{jk\ell}^{pqr})(x_{pqr}e_{pqr}) \\ &= (\{a_{jp}b_{kq}c_{\ell r}x_{pqr}\}_{pqr}e_{jk\ell}) = Y^v.\end{aligned} \tag{32}$$

The final expression of the RHS is the product of the post multiplication of $A \otimes B \otimes C$ by X^v. It is derived by cancelling the (superscripted) column indices pqr, which are associated with the basic matrices within the leading factor, which is the Kronecker product, with the (subscripted) row indices in the trailing factor, which is the vector X^v. The indices pqr affixed to the right brace in the penultimate expression emphasise that the elements within the braces are to be summed over the repeated indices. The indices of summation can be inferred from the contents of the braces.

The final expression of (32) suggests various ways of computing the product Y^v. One way would be by a single contraction or summation in respect of the composite index pqr. In so far as the vector X^v is concerned, the composite index is liable to correspond to a set of adjacent addresses in the memory of the computer.

The number of multiplications involved in forming the Kronecker product is $JP \times KQ \times LR$, which is the product of the number of elements in the constituent matrices. The operation of forming Y^v via a single contraction in respect of the composite index pqr entails $PQR \times JKL$ multiplications and $(PQR - 1) \times JKL$ additions.

A more efficient way of computing the product is by successive contractions in respect of the indices r, q and p, separately and successively. This could be described as the divide-and-conquer procedure. Then, a sequence of products, in long-vector form, that would lead from X^v to Y^v are as follows:

$$\begin{aligned}X^v &= (x_{pqr}e_{pqr}), \\ W^v &= (w_{\ell pq}e_{\ell pq}) = (\{c_{\ell r}x_{pqr}\}_r e_{\ell pq}), \\ V^v &= (v_{k\ell p}e_{k\ell p}) = (\{b_{kq}w_{\ell pq}\}_q e_{k\ell p}) = (\{b_{kq}c_{\ell r}x_{pqr}\}_{qr}e_{k\ell p}), \\ Y^v &= (y_{jk\ell}e_{jk\ell}) = (\{a_{jp}v_{k\ell p}\}_p e_{jk\ell}) = (\{a_{jp}b_{kq}c_{\ell r}x_{pqr}\}_{pqr}e_{jk\ell}).\end{aligned} \tag{33}$$

Here, the elements of the matrices $C = [c_{lr}]$, $B = [b_{kq}]$ and $A = [a_{jp}]$ are adduced to the product in the course of three successive contractions. Passing from X^v to W^v via the contraction over r requires $R \times LPQ$ multiplications. Passing from W^v to V^v via the contraction over q requires $Q \times KLP$ multiplications. Passing from V^v to Y^v via the contraction over p requires $P \times JKL$ multiplications.

However, there are six different orders in which the contractions can be performed, which correspond to the permutations of the indices p, q, r, which are the column indices of the matrices A, B, and C, respectively. In comparing the efficiency of the alternative procedures, one need only count the number of multiplications, since these are far more expensive than are the additions (the orders of computational complexity are n^2 and n, respectively).

When $R = 2$, $Q = 4$ and $P = 3$, with $J = K = L = 2$ as before, there are 384 multiplications in the slow procedure. The number of multiplications in the divide-and-conquer procedure depends on the sequence of the contractions. The number of multiplications in the various the divide-and-conquer procedures are listed below with the sequence of contractions denoted by a sequence of matrices, where the order in which they are deployed should be read from right to left (as in a sequence of matrix multiplications):

$$\begin{array}{llll} A, & B, & C:120 & \qquad C, \quad B, \quad A:96 \\ B, & C, & A:112 & \qquad A, \quad C, \quad B:96 \\ C, & A, & B:88 & \qquad B, \quad A, \quad C:128 \end{array} \tag{34}$$

To minimise the computational burden by minimising the number of multiplications within the divide-and-conquer procedure, one should ensure the contractions in respect of p, q, and r are ordered such that numbers of the associated multiplications are declining. When the row orders of the matrices are equal, the most efficient sequence of contractions will be self-evident, and there will be no need to count the number of multiplications.

The calculation of the product of (32) can de accomplished using nested do-loops. Six loops are required that fall into two groups, which comprise the outer indices j, k, ℓ and the inner indices p, q, r. The former group are the indices of the product vector and the latter group are the indices of the successive contractions. Within each group, the order of the indices may be freely permuted.

However, it might pay to deploy the outer indices in the order listed above, since this will enable the elements of the product to be created on the fly in the order in which they would be printed in a column. Otherwise, the printing of the column would need to be delayed until all of its elements have been created.

To illustrate a code appropriate to such calculations, it is enough to consider the equation $Y^v = (A \otimes B)X^v$, which can be rendered in the index notation as

$$(y_{jk}e_{jk}) = (\{a_{jp}b_{kq}\}e_{jk}^{pq})(x_{pq}e_{pq}) = (\{a_{jp}b_{kq}x_{pq}\}_{pq}e_{jk}). \tag{35}$$

The code is as follows:

```
for j := 1 to J do
  for k := 1 to K do
    begin {jk}
      y[j][k] := 0;
      for q := 1 to Q do
        for p := 1 to P do
          y[j][k] := y[j][k] + a[j][p] * b[k][q] * x[p][q];
    end; {jk}
```
(36)

The code becomes increasingly prolix with a growing number of indices, each of which gives rise to a nested do-loop. However, such code can be encapsulated within a procedure or a subroutine with a concise interface.

6. Matrix Multiplications and the Reshaping Operations

Many authors have recognised the inefficiency of forming a multiple Kronecker product before postmultiplying it by a vector. They have proposed a variety of methods that rely on pairwise matrix multiplications that are accompanied by the shuffling of the

elements of the products in order to facilitate the succeeding matrix multiplications. These operations have been described as the reshaping of the matrices.

A method for solving the equation $Y^v = (A \otimes B \otimes C)X^v$ of (6) for X was proposed by Pereyra and Scherer (1973). A much simplified solution was provided by de Boor (1979), who showed that this could be achieved by a sequence of solutions involving the matrices A, B and C successively.

An indication or the way in which the product (6) might be calculated via a sequence of matrix multiplications was given in a paper on Dyksen (1987), which contained the identity $(A \otimes B)X^v = \{B(AX)'\}^c$, where $X = (x_{pq}e_p^q)$ and $X^v = (x_{pq}e_{pq})$, albeit that the necessary vectorisations of the matrices was not indicated.

A paper by Davio (1981) dealt at length with the matter of shuffling the elements of Kronecker product; but it did not relate such operations to pointers. Others, including Benoit et al. (2003), Langville and Stewart (2004) and Dayar and Orhan (2015) have also dealt with such issues. Programs that are aimed at computing the product of (6) have been provided by Constantine and Gleich (2009) and by Kredler (2015).

In a recent paper, Fackler (2019) has proposed that computations involving multiple Kronecker products can be achieved by ordinary matrix multiplications without displacing elements that are stored in the computer memory, as shuffling is liable to do. To illustrate how this method can be implemented in the MATLAB program, we may consider an input vector X^c, which follows a column-major scheme. Then, the method, which applies the matrices $A = [a_{jp}]$, $B = [b_{kq}]$ and $C = [c_{\ell r}]$ in succession to X^c, entails the following objects,

$$\begin{aligned} X^c &= (x_{pqr}e_{rqp}), \\ W^c &= (w_{qrj}e_{jrq}) = (\{a_{jp}x_{pqr}\}_p e_{jrq}), \\ V^c &= (v_{rjk}e_{kjr}) = (\{b_{kq}w_{qrj}\}_q e_{kjr}), \\ Y^c &= (y_{jk\ell}e_{\ell kj}) = (\{c_{\ell r}v_{rjk}\}_r e_{\ell kj}). \end{aligned} \tag{37}$$

In the process, the basis vector undergo the following transformation:

$$e_{rqp} \to e_{jrq} \to e_{kjr} \to e_{\ell kj}. \tag{38}$$

It can be seen that, as the indices p, q, r of the contractions drop off the end of the sequence, the new indices j, k, ℓ join at the head.

When the equations of (37) are given their appropriate shapes, they become

$$\begin{aligned} X_p^c &= (x_{pqr}e_p^{rq}), \\ W_j^c &= (w_{qrj}e_j^{rq}) = (\{a_{jp}x_{pqr}\}_p e_j^{rq}) = (a_{jp}e_j^p)(x_{pqr}e_p^{rq}), \\ V_k^c &= (v_{rjk}e_k^{jr}) = (\{b_{kq}w_{qrj}\}_q e_k^{jr}) = (b_{kq}e_k^q)(w_{qrj}e_q^{jr}), \\ Y_\ell^c &= (y_{jk\ell}e_\ell^{kj}) = (\{c_{\ell r}v_{rjk}\}_r e_\ell^{kj}) = (c_{\ell r}e_\ell^r)(v_{rjk}e_r^{kj}), \end{aligned} \tag{39}$$

and the sequence of matrix multiplications can be denoted by

$$AX_p^c = W_j^c \to BW_q^c = V_k^c \to CV_r^c = Y_\ell^c. \tag{40}$$

This entails two operations that are amenable to the `reshape` procedure:

$$\begin{aligned} W_j^c &= (w_{qrj}e_j^{rq}) \to W^c = (w_{qrj}e_{jrq}) \to W_q^c = (w_{qrj}e_q^{jr}), \\ V_k^c &= (v_{rjk}e_k^{jr}) \to V^c = (v_{rjk}e_{kjr}) \to V_r^c = (v_{rjk}e_r^{kj}). \end{aligned} \tag{41}$$

Here, the variant matrices W_j^c, W_q^c and V_k^c, V_r^c have shapes that are derived directly from W^c and V^c, respectively, which represent what is stored in the computer's memory. The `reshape` procedure is explained in detail in the Appendix A.

If it is required to reverse the sequence of matrix multiplications that are applied to the column-major vector X^c, then these must become post-multiplications, and the corresponding equations will be

$$\begin{aligned}(X_r^c)' &= (x_{pqr}e_{qp}^r),\\ (W_\ell^c)' &= (w_{\ell pq}e_{qp}^\ell) = (\{c_{\ell r}x_{pqr}\}_r e_{qp}^\ell) = (x_{pqr}e_{qp}^r)(c_{\ell r}e_r^\ell),\\ (V_k^c)' &= (v_{k\ell p}e_{p\ell}^k) = (\{b_{kq}w_{\ell pq}\}_q e_{p\ell}^k) = (w_{\ell pq}e_{p\ell}^q)(b_{kq}e_q^k),\\ (Y_j^c)' &= (y_{jk\ell}e_{\ell k}^j) = (\{a_{jp}v_{k\ell p}\}_p e_{\ell k}^j) = (v_{k\ell p}e_{\ell k}^p)(a_{jp}e_p^j).\end{aligned} \tag{42}$$

The sequence of matrix multiplications can be denoted by

$$(X_r^c)'C' = (W_\ell^c)' \to (W_q^c)'B' = (V_k^c)' \to (V_p^c)'A' = (Y_j^c)'. \tag{43}$$

However, reference to (15) indicates that, if the row-major vector $X^v = (x_{pqr}e_{pqr})$ is available, then the following sequence of matrix pre-multiplications can be pursued:

$$CX_r^v = W_\ell^v \to BW_q^v = V_k^v \to AV_p^v = Y_j^v. \tag{44}$$

According to the table of (34), which is based on the values $P = 3$, $Q = 4$, $R = 2$ and $J = K = L = 2$, neither of the sequences of (40) and (44) lead to a fully efficient computation. A recourse might be to re-order the matrices within the multiple Kronecker product to create the most efficient sequence, which would also require the elements within the vector Y^c to be re-ordered. Whether this would be allowable or convenient is a matter that depends on the substantive nature of the problem that is represented by the equation. However, it is reasonable to suppose that, in most cases, this will be easy to achieve,

Next is the apparent obstacle that the `reshape` procedure is specific to MATLAB. However, as Fackler (2019) has indicated, it should be straightforward to replicate the procedure in many of the available programming languages, using the arithmetic of pointers. Whereas the method of pairwise multiplication has been described for the case of a column-major scheme, there is no difficulty in adapting to a row-major scheme.

Our preference would be to calculate the product of (6) according to the scheme of (33), or some permutation thereof, that does not require matrix multiplications. However, the intention of this paper has not been to insist on one way of another of conducting the computations, provided that gross inefficiency is avoided. Instead, the aim has been to clarify the nature of the alternative procedures.

Funding: This research has received no external funding.

Conflicts of Interest: The author declares that there are no conflicts of interest.

Appendix A. Examples of the Reshape Procedure

It the MATLAB program, a matrix array is stored, according to a column-major scheme, as a long vector, formed by adjoining successive columns of the matrix. The `reshape` procedure divides the long column into successive segments of equal length, which are arrayed in a new matrix.

An example of the reshaping operation is the following transformation:

$$A = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix} \to B = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}. \tag{A1}$$

This would be accomplished by the command `B = reshape A(2,3)`.

To illustrate some of the structures into which a multidimensional array may be cast, consider a three-dimensional array of which the elements are denoted by x_{pqr}. The indices range from 1 up to P, Q and R, respectively.

We shall assume that $P = Q = 2$ and that $R = 3$. Then, the row-major array is a wide row vector

$$X^r = (x_{pqr}e^{pqr}) = [x_{111}\ x_{112}\ x_{113}\ x_{121}\ x_{122}\ x_{123}\ x_{211}\ x_{212}\ x_{213}\ x_{221}\ x_{222}\ x_{223}]. \tag{A2}$$

Reversing the order of the indices of the basic vectors and transposing gives

$$X^c = (x_{pqr}e_{rqp}) = [x_{111}\ x_{211}\ x_{121}\ x_{221}\ x_{112}\ x_{212}\ x_{122}\ x_{222}\ x_{113}\ x_{213}\ x_{123}\ x_{223}]'. \tag{A3}$$

which is a long column vector. Recall that the sequence of indices on the basis elements follow a lexicographic order. Furthermore, note that the mapping $X^r \to X^c$ involves a commutation of the indices of the basis vector followed by a transposition. It is not amenable to a simple reshape operation.

A single reshape operation effects the following transformation of X^c:

$$X^c = (x_{pqr}e_{rqp}) \to X^c_p = (x_{pqr}e^{rq}_p) = \begin{bmatrix} x_{111} & x_{121} & x_{112} & x_{122} & x_{113} & x_{123} \\ x_{211} & x_{221} & x_{212} & x_{222} & x_{213} & x_{223} \end{bmatrix}. \tag{A4}$$

Here, the leading index $p = 1, 2$ of the element x_{pqr} has become the row index of X^c_p. The composite column index qr follows two iterations of the sequence $11, 21, 12, 22, 13, 23$ in which $r = 1, 2, 3$ is evolving more slowly than $q = 1, 2$. This is an anti-lexicographic sequence.

An alternative reshape operation yields

$$X^c = (x_{pqr}e_{rqp}) \to X^c_{qp} = (x_{pqr}e^r_{qp}) = \begin{bmatrix} x_{111} & x_{112} & x_{113} \\ x_{211} & x_{212} & x_{213} \\ x_{121} & x_{122} & x_{123} \\ x_{221} & x_{222} & x_{223} \end{bmatrix}. \tag{A5}$$

Here, $r = 1, 2, 3$, which is the trailing index of the element x_{pqr}, has become the column index of X^c_{qp}. The composite row index qr follows three iterations of the sequence $11, 21, 12, 22$, which is also in an anti-lexicographic order.

Transposing this matrix gives

$$X^c_{qp} = (x_{pqr}e^r_{qp}) \to X^c_r = (x_{pqr}e^{qp}_r) = \begin{bmatrix} x_{111} & x_{211} & x_{121} & x_{221} \\ x_{112} & x_{212} & x_{122} & x_{222} \\ x_{113} & x_{213} & x_{123} & x_{223} \end{bmatrix}. \tag{A6}$$

The object that remains unchanged in the memory of the computer throughout these operations is $X^c = (x_{pqr}e_{rqp})$. It follows that the transformation

$$X^c_r = (x_{pqr}e^{qp}_r) \to X^c_p = (x_{pqr}e^{rq}_p) \tag{A7}$$

can be accomplished by a single reshape operation. This transformation is the prototype of the transformations of (41):

$$W^c_j = (w_{qrj}e^{rq}_j) \to W^c_q = (w_{qrj}e^{jr}_q), \tag{A8}$$

$$V^c_k = (v_{rkj}e^{jr}_k) \to V^c_r = (v_{rjk}e^{kj}_r). \tag{A9}$$

The matrices in question have either the leading or the trailing elements of the composite indices qrj and rjk as their row indices. The middle indices cannot become row indices without first resorting to a commutation operation, which would bring then to the head or the tail of a triple index.

References

Bader, Bret W., and Tamara G. Kolda. 2006. Algorithm 862: Matlab Tensor Classes for fast Algorithm Prototyping. *ACM Transactions on Mathematical Software (TOMS)* 32: 635–53. [CrossRef]

Balestra, Pietro. 1976. *La Dérivation Matricielle.* Paris: Éditions Sirey.
Benoit, Anne, Brigitte Plateau, and William J. Stewart. 2003. Memory-Efficient Kronecker Algorithms with Applications to the Modelling of Parallel Systems. Paper presented at the International Parallel and Distributed Processing Symposium (IPDPS'03), Nice, France, April 22–26.
Bourbaki, Nicolas. 1948. Algèbre Multilinéaire. In *Eléments de Mathematique*. Chapter 3 of Book II (Algèbre). Paris: Herman.
Bower, Allan F. 2010. *Applied Mechanics of Solids*. Boca Raton: Taylor and Francis.
Brau, Charles A. 2004. *Modern Problems in Classical Electrodynamics*. Oxford: Oxford University Press.
Cartan, Élie, 1952. *Géometrie des Espaces de Riemann*. Paris: Gauthier–Villars.
Constantine, Paul G., and David F. Gleich. 2009. kronmult: Fast and Efficient Kronecker Multiplication. Available online: https://www.mathworks.com/matlabcentral/fileexchange/23606 (accessed on on 27 April 2021).
Davio, Marc. 1981. Kronecker Products and Shuffle Algebra. *IEEE Transactions on Computers* C-30: 116–25. [CrossRef]
Dayar, Tŭgrul, and M. Can Orhan. 2015. On Vector-Kronecker Product Multiplication with Rectangular Factors. *SIAM Journal on Scientific Computing* 37: 526–43. [CrossRef]
de Boor, Carl. 1979. Efficient Computer Manipulation of Tensor Products. *ACM Transactions on Mathematical Software (TOMS)* 5: 173–82. [CrossRef]
De Lathauwer, Lieven, Bart L. R. De Moor, and Joos Vandewalle. 2000. A Multilinear Singular Value Decomposition. *SIAM Journal on Matrix Analysis and Applications* 21: 1253–78. [CrossRef]
Dyksen, Wayne R. 1987. Tensor Product Generalized Adi Methods for Separable Elliptic Problems. *SIAM Journal on Numerical Analysis* 24: 59–76. [CrossRef]
Fackler, Paul L. 2019. Algorithm 993: Efficient Computation with Kronecker Products. *ACM Transactions on Mathematical Software* 45: 1–9. [CrossRef]
Graham, Alexander. 1981. *Kronecker Products and Matrix Calculus with Applications*. Chichester: Ellis Horwood.
Greub, Werner H. 1967. *Multilinear Algebra*. Berlin: Springer.
Harrison, Adam P., and Dileepan Joseph. 2016. Numeric Tensor Framework: Exploiting and Extending Einstein Notation. *Journal of Computational Science* 16: 128–39. [CrossRef]
Kredler, Matthias. 2015. kronm: Fast Kronecker Matrix Multiplication. Available online: https://www.mathworks.com/matlabcentral/fileexchange/53382 (accessed on 27 April 2021).
Langville, Amy N., and William J. Stewart. 2004. The Kronecker Product and Stochastic Automata Networks. *Journal of Computational and Applied Mathematics* 167: 429–47. [CrossRef]
Lawden, Derek F. 1967. *An Introduction to Tensor Calculus and Relativity*. London: Methuen.
Magnus, Jan R. 2010. On the Concept of Matrix Derivative. *Journal of Multivariate Analysis.* 101: 2200–6. [CrossRef]
Magnus, Jan R., and Heinz Neudecker. 1979. The Commutation Matrix: Some Properties and Applications. *Annals of Statistics* 7: 381–94. [CrossRef]
Magnus, Jan R., and Heinz Neudecker. 1988. *Matrix Differential Calculus with Applications in Statistics and Econometrics*. Chichester: John Wiley and Sons.
Marcus, Marvin. 1973. *Finite Dimensional Multilinear Algebra—Part I*. New York: Marcel Dekker.
Marder, Leslie. 1968. *An Introduction to Relativity*. London: Longmans, Green and Co.
Pereyra, Victor, and Godela Scherer. 1973. Efficient Computer Manipulation of Tensor Products with Applications to Multidimensional Approximation. *AMS Mathematics of Computation* 27: 595–605. [CrossRef]
Pollock, D. Stephen G. 1979. *The Algebra of Econometrics*. Chichester: John Wiley and Sons.
Pollock, D. Stephen G. 1985. Tensor Products and Matrix Differential Calculus. *Linear Algebra an Its Applications* 67: 169–93. [CrossRef]
Rogers, Gerald S. 1980. *Matrix Derivatives*. New York: Marcel Dekker.
Shephard, Geoffrey C. 1966. *Vector Spaces of Finite Dimension*. Edinburgh: Oliver and Boyd.
Steeb, Willi-Hans. 1997. *Matrix Calculus and Kronecker Product with Applications and C++ Programs*. Singapore: World Scientific Publishing Co.
Susskind, Leonard, and Art Friedman. 2018. *Special Relativity and Classical Field Theory*. London: Penguin Books.
Turkington, Darrell A. 2002. *Matrix Calculus and Zero-One Matrices*. Cambridge: Cambridge University Press.
Van Loan, Charles F. 2000. The Ubiquitous Kronecker Product. *Journal of Computational and Applied Mathematics* 123: 85–100. [CrossRef]
Zellner, Arnold. 1962. An Efficient Method of Estimating Seemingly Unrelated Regression Equations and Tests for Aggregation Bias. *Journal of the American Statistical Association* 57: 348–68. [CrossRef]