Development of Rules and Program for Tensor-like Calculus

Yu Sato

Abstract

On digital signal processing or spectrum analysis, physical value often has multivariate function and it is considered that its variables increase with developing measurement technologies and analysis technics. Mathematical expression can be written physical relation, however as for the notation of multi-variable or multi-dimensional are able to more develop. Some binary operation of existing notation writes multivariable, that is tensor, is seemed to be defined clearly. Thus we developed a tensor-like calculation rule which indicates dot, cross and outer product in addition to normal four arithmetic operations. Also, we formulated the minimum number of permutation. Furthermore we formulated assignment of multi-dimensional array as single-dimensional array. These consideration provides the ease of manipulating multivariable function.

KeyWords: FFT, Multi-Dimensional array

1 Introduction

For data analyses or numerical calculations using a computer, we need to program algorithm in a given programing language. In this line of work, we must often translate equations into programming language, but a certain level of programing technical skill is required to do this task. To perform the data analysis efficiently, we should make efforts to consider how to reduce the time for learning such techniques, and about what knowledge or ideas gain the greatest profit by the smallest effort, when developing the initial abstract framework. On digital signal processing or spectrum analysis, physical value often has multivariate information, and arrays are needed to analyze them in computer programs. Some programing languages provide intelligent multidimensional arrays that can partially support some binary operations. However, in the case of this study, we encountered challenges in translating these calculations to a complex program, and furthermore, such a program usually uses iterations that cannot easily be written using mathematical expressions: therefore, the program and the mathematical expressions lack *one-to-one correspondence*. The loss of one-to-one correspondence characterizes human translation. In mathematics, multidimensional variables are written as tensors for ease of manipulation, which is sophisticated notation because the tensor can encompass various binary calculation in the small value of a letter. Tensors are useful in programs, so we considered how to handle data using tensors and developed an original definition of a tensor class.

2 Considerations of data handling

As a concrete example, we considered the handling of a data for a plasma potential V_s. We assume that the plasma potential V_s is dependent on radial position R and t. Thus,

$$V_s = V_s(t,R). \tag{1}$$

The V_s of the plasma is measured at a fixed point for a regular time. Then V_s and t are discretized and are assigned to i, so

$$(V_s) = (V_s)[i], \tag{2}$$
$$t = t[i], \tag{3}$$
$$R = R, \tag{4}$$

where, angular brackets [ ] indicated the enclosed variable is discrete.

Next, we assume that V_s is measured at the measurement point R, which is moved iteratively shot by shot. Therefore, V_s, t, and R are not only dependent on i but also depend on j.

$$V = V[i,j], \tag{5}$$
$$t = t[i], \tag{6}$$
$$R = R[j]. \tag{7}$$

It is clear that t and R are only depended on the i and j. Thus, as the value assigned to the parameter is defined with a condition, a dependent variable can be assigned to the independent variable (in this case, $V(t,R)$). Here, we decide to treat these variables as tensors. Let i and j be subscripts of tensors, so $V[i,j]$, $t[i]$, and $R[j]$ change to $(V_s)_{ij}$, t_{ij} and R_{ij}, but $t_{ij} = t_i$ and $R_{ij} = t_j$. Let $\mathbf{A} \equiv \left((V_s)_{ij}, t_j, R_i\right)$ and let the vector element of $\mathbf{A}$ be k so that $\mathbf{A} = A_{ijk}$. In conclusion, the $V_s(t,R)$ is able to be shown as the tensor rank of three.

Furthermore, we consider a case where $V_s(t,R)$ are applied to time-shift fast-Fourier transform (TSFFT), with center of t and data width Δd, the power spectrum S is written as follows,

$$
\begin{aligned}
S[i,l] &= \left(\mathscr{F}\left[V\left[\mathscr{S}\left[i,\Delta d\right]\right]\right]\right)[l]\left(\mathscr{F}\left[V\left[\mathscr{S}\left[i,\Delta d\right]\right]\right]^*\right)[l], && (8)\\
\mathscr{S}:\ (i,\Delta d) &\rightarrow \left[\frac{\Delta d}{2} - i,\ \frac{\Delta d}{2} + i\right), && (9)
\end{aligned}
$$

where, l is a data number that satisfies the following expression,

$$
f = f[l], \tag{10}
$$

where, f is frequency. This transformation is applied every R (which is indicated as j), so that $S = S[i,j,l]$. In conclusion,

$$
\begin{aligned}
S &= S[i,j,l], && (11)\\
f &= f[l], && (12)\\
R &= R[j], && (13)\\
t &= t[i]. && (14)
\end{aligned}
$$

Let $\mathbf{B} \equiv \left(S_{ijl}, f_{ijl}, R_{ijl}, t_{ijl}\right)$ and let the subscript of $\mathbf{B}$ be k so that $\mathbf{B} = B_{ijkl}$. Therefore, when as TSFFT is applied, the data results in a tensor with a rank of four.

We generalize the above results to data with physical value A which is measured experimentally,

$$
\begin{aligned}
A &= A[\mathbf{x}], && (15)\\
\mathbf{P} &= f[X], && (16)\\
f:\ x_n \in X &\rightarrow (P[x_n])_{n\in\Lambda}, && (17)\\
\mathbf{x} &\equiv \prod_{n\in\Lambda} x_n, && (18)\\
X &\equiv \{x_n\}_{n\in\Lambda}, && (19)\\
\Lambda &= \{n \in \mathbb{N} | 0 \leq n < N\}. && (20)
\end{aligned}
$$

where, the n is the condition number, x_n is the data number in that condition, and $\mathbf{P}$ is the vector for the parameter. The element for A and $\mathbf{P}$ is defined as $\mathbf{B}$ so that

$$
\mathbf{B} \equiv (A, P_0, \cdots, P_{N-1}). \tag{21}
$$

Thus, the A is treated as having a rank of one, and $\mathbf{B}$ can be written as $B_{i_0 \cdots i_N}$ and has the rank of $N+1$. Also, for an M-tuple vector field,

$$
\mathbf{B} \equiv (A_0, \cdots A_{M-1}, P_0, \cdots, P_{N-1}). \tag{22}
$$

2.1 How to handle data

The number of the parameters has at most 1 or 10 of order, which humans can easily handle. Dividing this parameter was considered to allow for more efficiency. We Taking into account this point, we developed a class (program) having parameters that are Nth rank tensor. This class is $\mathbf{B}$ in the sense of Eq. 21 or 22, and which has the element included as the dependent variable and independent variable. This class has some merits as follows,

1. Since the class has the physical value as attributes or member variables just below of the class, they possess good accessibility. For example, the characteristics behave well in plotted graphs..
2. Since the each of the parameters is assigned to the data number, calculation and management of new produced data is easy.

2.2 Memory saving

For example, in Eq. 2 and 3, t and R are first rank tensors, but to associate with V_s which is a second rank tensor, t and R are inflated to the second tensors. Excessive use of memory is a concern when handling big data. As a solution for this case, inflation of tensor rank is performed when accessing the data.

$$a_i 1_j = a_{ij}. \tag{23}$$

If the rank is 2, this tensor is handled as a matrix so that easily handling is possible using the matrix class in existence. However in the case of high-order tensors the handling is difficult. For example,

$$a_i 1_{jklm} = a_{ijklm}. \tag{24}$$

At present, a class with mass appeal that can handle high-order tensors does not exist, but it would be useful to have a data class that could handle such tensors easily. Therefore, we considered a tensor class that follows the rule of tensor calculus to developed a new tensor class.

3 Rules of tensor calculus

We defined some rules of tensor calculus that basically follow general rules; however, in consideration of the basic data type *array*, we expanded some rules of tensor calculus[1, 2, 3] and defined an original the rule of the tensor calculus. These novel rules provided the element product, which can be defined in Fortran or MATLAB, and so on. These high-level languages are also provided dots, crosses and outer products, thus the merit of using this rule is clear,

> If the rules of tensor calculus are implemented with a programing language, the mathematical notation corresponds to the program notation on a one-to-one basis.

This contribute the thinking the economy.

3.1 Definition of name

1. n dimensional array used in a program called a *tensor*; the variable is written as only one character.
2. If a variable needs the additional information, such as super or subscript defined below, to avoid collision between which and index of tensor, the class of variable and additional information is includes into parenthesis.

3.2 Definition of index

1. The index of a tensor is located at the upper-right or lower-right of the variable, called the superscript and subscript respectively. The superscript and subscript are distinguished as follows,

$$a_i \neq a^i, \tag{25}$$
$$a^i_j \neq a^j_i. \tag{26}$$

2. The number of the index is called its *rank*.
3. The index of a tensor is interchangeably.

$$a_{ij} = a_{ji}, \tag{27}$$
$$\varepsilon_{ijk} = \varepsilon_{jik}. \tag{28}$$

4. A tensor that has no index is a 0th rank tensor, and is called a *scalar*.

3.3 Definition of calculation

1. Calculations are performed from the left. If parentheses () are included, calculations inside parentheses are performed first.

2. For a calculation between a scalar and a tensor, the scalar operates to each element of the tensor.

$$
\begin{aligned}
a+b_i &= c_i, && (29)\\
a-b_i &= c_i, && (30)\\
a*b_i &= c_i, && (31)\\
a/b_i &= c_i, && (32)\\
b_i/a &= c_i. && (33)
\end{aligned}
$$

3. For a calculation between two tensors that have the same index name and location, the operation is performed between each element of the two tensors.

$$
\begin{aligned}
a_i+b_i &= c_i, && (34)\\
a_i-b_i &= c_i, && (35)\\
a_i*b_i=a_ib_i &= c_i, && (36)\\
a_i/b_i &= c_i, && (37)\\
b_i/a_i &= c_i. && (38)
\end{aligned}
$$

4. The dot product is performed when the two tensors which has same index but the location is odd.

$$
\begin{aligned}
a_ib^i &= c, && (39)\\
a^ib^j_i &= c^j. && (40)
\end{aligned}
$$

5. The outer product is performed when indices of the two tensors are not matched.

$$
\begin{aligned}
a_ib^j &= c^j_i, && (41)\\
a_ib_j &= c_{ij}. && (42)
\end{aligned}
$$

6. When the indices of the two tensors have same name and same location, but the rank of one of the two tensors is greater, the outer product of the elements product are performed.

$$
\begin{aligned}
a_ib^j_i &= c^j_i, && (43)\\
a_ib^{kj}_i &= c^{kj}_i, && (44)\\
a^j_ib^{kj}_i &= c^{kj}_i. && (45)
\end{aligned}
$$

7. In the case of the two tensors both having an odd index, the outer product is performed for odd indices, and the 6. is performed for same number of indices.

$$
a^j_ib^k_i = c^{jk}_i, \quad (46)
$$

in the case of $i=3$,

$$
\begin{aligned}
c^{jk}_1 &= a^j_1b^k_1, && (47)\\
c^{jk}_2 &= a^j_2b^k_2, && (48)\\
c^{jk}_3 &= a^j_3b^k_3. && (49)
\end{aligned}
$$

8. Element and dot product are not commutative.

$$
\begin{aligned}
a_ib^ic_i &= d_i, && (50)\\
a_ic_ib^i &= e, && (51)\\
d_i &\neq e, && (52)\\
a_ib^ic_i &\neq a_ic_ib^i. && (53)
\end{aligned}
$$

3.4 Definition of specific notation

1. If all elements in a tensor is same number $a \in \mathbb{C}$ and index is j, this tensor is written as a_j. For example, in the case of $a = 3$, this tensor is written as 3_j.

2. Triangular tensor T is defined as follows,

$$T_{ij} = \begin{cases} 1 & i \geq j \\ 0 & i < j \end{cases}. \tag{54}$$

3. A so-called Kronecker delta is defined as follows,

$$\delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}. \tag{55}$$

3.5 The minimum number of permutation

To calculate the cross product, permutation symbol ε is used. The ε has rank n and is written as follows,

$$\varepsilon_{a_1 a_2 a_3 \cdots a_n} = \begin{cases} +1 & \text{if } (a_1, a_2, a_3, \cdots, a_n) \text{ is an even permutation of} (1, 2, 3 \cdots, n) \\ -1 & \text{if } (a_1, a_2, a_3, \cdots, a_n) \text{ is an odd permutation of} (1, 2, 3 \cdots, n) \\ 0 & \text{otherwise} \end{cases}. \tag{56}$$

We need to define the rule of even and odd permutations. Let $\mathbf{A} = \prod_{i=0}^{|I|-1} A_i, \mathbf{B} = \prod_{i=0}^{|I|-1} B_i, A_i = \{a_i \in I\}_{i \in I}, B_i = \{b_i \in I\}_{i \in I}, I = \{i \in \mathbb{N}_0\}$, and $\mathbf{C} \equiv \mathbf{A} - \mathbf{B}$. If $\mathbf{A}$ is permutated to $\mathbf{B}$ with the minimum number of permutations t,

$$t = \begin{cases} |\mathbf{u}(\mathbf{C})| & |\mathbf{u}(\mathbf{C})| - |\mathbf{u}(-\mathbf{C})| \geq 0 \\ |\mathbf{u}(-\mathbf{C})| & |\mathbf{u}(\mathbf{C})| - |\mathbf{u}(-\mathbf{C})| < 0 \end{cases}, \tag{57}$$

$$\mathbf{u} : i \mapsto u(i), \tag{58}$$

$$u(x) \equiv \begin{cases} 1 & x > 0 \\ 0 & \text{otherwise} \end{cases}. \tag{59}$$

The number of t can be acquired.

$$\varepsilon_{a_1 a_2 a_3 \cdots a_n} = \begin{cases} +1 & \text{if } t \text{ is an even number} \\ -1 & \text{if } t \text{ is an odd number} \\ 0 & t = 0 \end{cases}. \tag{60}$$

3.6 Consideration how to reduce the rank of tensor

Many programing languages provide dot, cross, outer and element products for arrays. The rule 5. of tensor calculus mentioned in 3.3 includes dots and element products. Though which calculation is prior is arbitrarily decided, if dot product is prior, a duplicate index is generated as follows,

$$a_i^j b_k^{ij} = c_k^{jj}. \tag{61}$$

This is invalid in the rules of tensor calculus mentioned as 3.3 because the rules does not allowed the duplicate index on one tensor. However, for convenience of description, we permit this duplication. As a necessary method of transformation, we consider that following process to be valid,

$$\begin{aligned} c_k^{jj} \delta^{jj} 1_j &= d_k^{jj} 1_j \\ &= e_k^j. \end{aligned} \tag{62}$$

On the other hand, giving priority the element product,

$$\begin{aligned} a_i^j b_k^{ij} &= c_{ik}^{ij}, \\ c_{ik}^{ij}\delta_i^i 1_i 1^i &= e_k^j. \end{aligned} \tag{63}$$

Calculating dot product first is better than calculating the element product first, because the number of calculations is fewer.

4 Example of calculation

4.1 $a_i^j b_k^{ij} = e_k^j$

The index is $i = 2, j = 3, k = 2$. Though matrix-like notation is indicated, the calculation does not follow the matrix product.

$$\begin{aligned}
a_i^j &\equiv \begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \end{bmatrix}, \\
b_k^{ij} &\equiv \left[\begin{bmatrix} 3 & 3 & 3 \\ 3 & 3 & 3 \end{bmatrix}, \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \right], \\
a_i^j b_k^{ij} &= \begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \end{bmatrix} \left[\begin{bmatrix} 3 & 3 & 3 \\ 3 & 3 & 3 \end{bmatrix}, \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \right], \\
&= \left[\begin{bmatrix} 1 & 2 \\ 1 & 2 \\ 1 & 2 \end{bmatrix}^T \begin{bmatrix} 3 & 3 & 3 \\ 3 & 3 & 3 \end{bmatrix}, \begin{bmatrix} 1 & 2 \\ 1 & 2 \\ 1 & 2 \end{bmatrix}^T \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \right], \\
c_k^{jj} &= \left[\begin{bmatrix} 9 & 9 & 9 \\ 9 & 9 & 9 \\ 9 & 9 & 9 \end{bmatrix}, \begin{bmatrix} 3 & 3 & 3 \\ 3 & 3 & 3 \\ 3 & 3 & 3 \end{bmatrix} \right], \\
c_k^{jj}\delta^{jj}1_j &= \left[\begin{bmatrix} 9 & 9 & 9 \\ 9 & 9 & 9 \\ 9 & 9 & 9 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 3 & 3 & 3 \\ 3 & 3 & 3 \\ 3 & 3 & 3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right], \\
e_k^{jj}1_j &= \left[\begin{bmatrix} 9 & 0 & 0 \\ 0 & 9 & 0 \\ 0 & 0 & 9 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 3 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right], \\
e_k^j &= \left[\begin{bmatrix} 9 \\ 9 \\ 9 \end{bmatrix}, \begin{bmatrix} 3 \\ 3 \\ 3 \end{bmatrix} \right].
\end{aligned}$$

4.2 Assignment of matrix

A second order tensor can be written as a matrix. In this case, we assume a tensor a_{ij} of row i and column j.

$$\begin{aligned}
a_{ij} &= \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, \\
b^i &= (1,2,3), \\
c^j &= (4,5,6), \\
a_{ij}b^i c^j &= [1,2,3] \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} \\
&= [30,36,42] \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} = 552.
\end{aligned}$$

but,$a_{ij} = a_{ji}$, not $a_{ij} = a_{ji}^T$ (T indicates the transpose).

4.3 Cross product

The cross product of $\mathbf{A} \times \mathbf{B}$ following the rules of tensor calculus described in 3.3, is written as follows,

$$\mathbf{A} \times \mathbf{B} \equiv c_k = \varepsilon_k^{ij} A_i B_j, \tag{64}$$
$$= A_i \varepsilon_k^{ij} B_j, \tag{65}$$
$$= A_i B_j \varepsilon_k^{ij}. \tag{66}$$

If

$$A_i \equiv \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}, \tag{67}$$
$$B_j \equiv \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}, \tag{68}$$
$$\varepsilon_k^{ij} = \left[\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & -1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \right], \tag{69}$$

$\mathbf{A} \times \mathbf{B}$ in Eq. 64, 65 and 66 are written as follows,

$$
\begin{aligned}
\varepsilon_k^{ij} A_i B_j &= \left[\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & -1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}\right] \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} \\
&= \left[\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & -2 \\ 0 & 0 & 0 \\ 2 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 3 & 0 \\ -3 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}\right] \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} \\
&= \begin{bmatrix} 0 & 3 & -2 \\ -3 & 0 & 1 \\ 2 & -1 & 0 \end{bmatrix} \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} \\
&= \begin{bmatrix} -3 & 6 & -3 \end{bmatrix}, \\
A_i \varepsilon_k^{ij} B_j &= \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \left[\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & -1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}\right] \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} \\
&= \left[\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & -2 \\ 0 & 0 & 0 \\ 2 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 3 & 0 \\ -3 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}\right] \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} \\
&= \begin{bmatrix} 0 & 3 & -2 \\ -3 & 0 & 1 \\ 2 & -1 & 0 \end{bmatrix} \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} \\
&= \begin{bmatrix} -3 & 6 & -3 \end{bmatrix}, \\
A_i B_j \varepsilon_k^{ij} &= \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} \left[\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & -1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}\right] \\
&= \begin{bmatrix} 4 & 8 & 12 \\ 5 & 10 & 15 \\ 6 & 12 & 18 \end{bmatrix} \left[\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & -1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}\right] \\
&= \left[\left[\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 4 \\ 0 & -4 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & -8 \\ 0 & 0 & 0 \\ 8 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 12 & 0 \\ -12 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}\right],\right. \\
&\quad \left[\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 5 \\ 0 & -5 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & -10 \\ 0 & 0 & 0 \\ 10 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 15 & 0 \\ -15 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}\right], \\
&\quad \left.\left[\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 6 \\ 0 & -6 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & -12 \\ 0 & 0 & 0 \\ 12 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 18 & 0 \\ -18 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}\right]\right]
\end{aligned}
$$

$$
\begin{aligned}
&= \left[\begin{bmatrix} 0 & 12 & -8 \\ -12 & 0 & 4 \\ 8 & -4 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 15 & -10 \\ -15 & 0 & 5 \\ 10 & -5 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 18 & -12 \\ -18 & 0 & 6 \\ 12 & -6 & 0 \end{bmatrix}\right] \\
&= \left[\begin{bmatrix} 0 & 12 & -8 \\ -12 & 0 & 4 \\ 8 & -4 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 15 & -10 \\ -15 & 0 & 5 \\ 10 & -5 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 18 & -12 \\ -18 & 0 & 6 \\ 12 & -6 & 0 \end{bmatrix}\right] \\
&\quad \left[\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}\right] \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \\
&= \left[\begin{bmatrix} 0 & 12 & -8 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 \\ -15 & 0 & 5 \\ 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 12 & -6 & 0 \end{bmatrix}\right] \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \\
&= \left[\begin{bmatrix} 0 & 12 & -8 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ -15 & 0 & 5 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 12 & -6 & 0 \end{bmatrix}\right] \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \\
&= \begin{bmatrix} 0 & 12 & -8 \\ -15 & 0 & 5 \\ 12 & -6 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \\
&= \begin{bmatrix} -3 & 6 & -3 \end{bmatrix}.
\end{aligned}
$$

5 Conserving memory

The rules of tensor calculus partially allow the commutative law. Therefore, some calculations can result in the same result. For example,

$$
a^i b^j c^k d_{ijk} \quad \to \quad e^{ij} c^k d_{ijk} \to f^{ijk} d_{ijk} \to g. \tag{70}
$$

To get g, the equation needs four tensors $a^i, b^j, c^k, d_{ijk}, e^{ij}, f^{ijk}$. Thus, the number of the array is $i^4 \times j^4 \times k^3$ for calculation. On the other hand,

$$
d_{ijk} a^i b^j c^k \quad \to \quad e_{jk} b^j c^k \to f_k c^k \to g \tag{71}
$$

needs the array of $a^i, b^j, c^k, d_{ijk}, e_{jk}, f_k$, and the number of this is $i^2 \times j^3 \times k^4$. Comparing both,

$$
\frac{i^4 j^4 k^3}{i^2 j^3 k^4} \quad = \quad \frac{i^2 j}{k}. \tag{72}
$$

Therefore, if $i^2 j > k$ Eq. 70 is more efficient than Eq. 71, and if not then *vice versa*.

6 Transformation from a $1-$dimensional array to a $n-$dimensional array

There exists a tensor $a_{i_1 i_2 \cdots i_N \in \mathbb{N}_+}$. We consider that the index of $a_{i_1 i_2 \cdots i_N}$ can be written as a one-dimensional tensor a_i. This merits are as follows,

1. Any tensor rank can be written easily.
2. The transposition can be written easily.

If a vector $\mathbf{D}$ that indicates the each dimension of $a_{i_1 i_2 \cdots i_N}$ is written as

$$
\mathbf{D} \quad = \quad (i_1, i_2, \ldots, i_N) = D_j, \tag{73}
$$

and if the following expressions are defined as follows,

$$
\begin{aligned}
B_j &\equiv \left(b_j | b_j = \prod_{i=0}^{j-1} D_{i-1},\ D_{-1} = 1 \right), && (74)\\
C_j &\equiv \left(c_j | c_j = \prod_{i=0}^{j-1} D_i \right), && (75)\\
R^k &\equiv \mathbb{N}_0, && (76)\\
M^{jk} &\equiv \begin{cases} 1_k & [0, D_j) \\ 0_k & \text{otherwise} \end{cases}, && (77)\\
u_{ijk} &\equiv \begin{cases} 1_i & [B_j R_k, B_j R_k + B_l),\ i = i + C_j \\ 0_i & \text{otherwise} \end{cases}. && (78)
\end{aligned}
$$

$a_{i_1 i_2 \cdots i_N}$ can be written as follows,

$$
a_i = R^k M^{jk} B^j u_{ijk}. \tag{79}
$$

Example Let (i_1, i_2, i_3) be $(2,2,3)$. Using Eq. 79, $a_{i_1 i_2 i_3}$ can be written as,

$$
\begin{aligned}
R^k &= \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}, \\
M^{kj} &= \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}, \\
u_{ijk} &= u\left(\left[\begin{bmatrix} 6 & 3 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}, \begin{bmatrix} 6 & 3 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix} + \begin{bmatrix} 6 & 3 & 1 \end{bmatrix} \right], i = i + \begin{bmatrix} 12 & 6 & 3 \end{bmatrix} \right) \\
&= u\left(\left[\begin{bmatrix} 0 & 0 & 0 \\ 6 & 3 & 1 \\ 12 & 6 & 2 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 \\ 6 & 3 & 1 \\ 12 & 6 & 2 \end{bmatrix} + \begin{bmatrix} 6 & 3 & 1 \end{bmatrix} \right], i = i + \begin{bmatrix} 12 & 6 & 3 \end{bmatrix} \right) \\
&= u\left(\left[\begin{bmatrix} 0 & 0 & 0 \\ 6 & 3 & 1 \\ 12 & 6 & 2 \end{bmatrix}, \begin{bmatrix} 6 & 3 & 1 \\ 12 & 6 & 2 \\ 18 & 9 & 3 \end{bmatrix} \right], i = i + \begin{bmatrix} 12 & 6 & 3 \end{bmatrix} \right), \\
a_i &= \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 6 & 3 & 1 \end{bmatrix} u\left(\begin{bmatrix} [0,6) & [0,3) & [0,1) \\ [6,12) & [3,6) & [1,2) \\ [12,18) & [6,9) & [2,3) \end{bmatrix}, i = i + \begin{bmatrix} 12 & 6 & 3 \end{bmatrix} \right) \\
&= \begin{matrix} 0*6u_i([0,6)) & + & 0*3u_i([0,3)) & + & 0*1u_i([0,1)) \\ +1*6u_i([6,12)) & + & 1*3u_i([3,6)) & + & 1*1u_i([1,2)) \\ 0 & + & 0 & + & 2*1u_i([2,3)) \end{matrix}, \\
&= \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 6 & 6 & 6 & 6 & 6 & 6 \end{pmatrix} \\
&\quad + \begin{pmatrix} 0 & 0 & 0 & 3 & 3 & 3 & 0 & 0 & 0 & 3 & 3 & 3 \end{pmatrix} \\
&\quad + \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} \\
&\quad + \begin{pmatrix} 0 & 0 & 2 & 0 & 0 & 2 & 0 & 0 & 2 & 0 & 0 & 2 \end{pmatrix}, \\
&= \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \end{pmatrix}. && (80)
\end{aligned}
$$

That is, Eq. 80 indicates the follow assignment as one dimensional tensor,

$$
a_{i_1 i_2 i_3} = \left[\begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \end{bmatrix} \begin{bmatrix} 6 & 7 & 8 \\ 9 & 10 & 11 \end{bmatrix} \right]_{i_1=2, i_2=2, i_3=3}.
$$

7 Transpose of tensor

We defined the *transpose of tensor* as the map of vector-to-vector map defined as Eq. 79. Let $(D')_j$ denote transposed D_j. Let $(B')_j, (C')_j, (M')_{jk}$ and $(u')_{ijk}$ be B_j, C_j, M_{jk} and u_{ijk} acquired from $(D')_j$, let $(B'')_j$ be transposed B_j, and let $(a')_i$ be transposed a_i, then the following expression is valid,

$$a'_i = R^k \left(M'\right)^{jk} \left(B''\right)^k \left(u'\right)_{ijk}. \tag{81}$$

Example We consider the transposition from $(i_1, i_2, i_3) = (2,2,3)$ to $(i_1, i_3, i_2) = (2,3,2)$,

$$\begin{aligned}
R^k &= \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}, \\
\left(M'\right)^{jk} &= \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \\
\left(u'\right)_{ijk} &= u\left(\left[\begin{bmatrix} 6 & 2 & 1 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}, \begin{bmatrix} 6 & 2 & 1 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix} + \begin{bmatrix} 6 & 2 & 1 \end{bmatrix}\right], i = i + \begin{bmatrix} 12 & 6 & 2 \end{bmatrix}\right) \\
&= u\left(\left[\begin{bmatrix} 0 & 0 & 0 \\ 6 & 2 & 1 \\ 12 & 4 & 2 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 \\ 6 & 2 & 1 \\ 12 & 4 & 2 \end{bmatrix} + \begin{bmatrix} 6 & 2 & 1 \end{bmatrix}\right], i = i + \begin{bmatrix} 12 & 6 & 2 \end{bmatrix}\right) \\
&= u\left(\left[\begin{bmatrix} 0 & 0 & 0 \\ 6 & 2 & 1 \\ 12 & 4 & 2 \end{bmatrix}, \begin{bmatrix} 6 & 2 & 1 \\ 12 & 4 & 2 \\ 18 & 6 & 3 \end{bmatrix}\right], i = i + \begin{bmatrix} 12 & 6 & 2 \end{bmatrix}\right), \\
a_i &= \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} 6 & 1 & 3 \end{bmatrix} u\left(\begin{bmatrix} [0,6) & [0,2) & [0,1) \\ [6,12) & [2,4) & [1,2) \\ [12,18) & [4,6) & [2,3) \end{bmatrix}, i = i + \begin{bmatrix} 12 & 6 & 2 \end{bmatrix}\right) \\
&= \begin{array}{ccccc} 0*6u_i([0,6)) & + & 0*1u_i([0,2)) & + & 0*3u_i([0,1)) \\ +1*6u_i([6,12)) & + & 1*1u_i([2,4)) & + & 1*3u_i([1,2)) \\ 0 & + & 2*1u_i([4,6)) & + & 0 \end{array}, \\
&= \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 6 & 6 & 6 & 6 & 6 & 6 \end{pmatrix} \\
&\quad + \begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \end{pmatrix} \\
&\quad + \begin{pmatrix} 0 & 3 & 0 & 3 & 0 & 3 & 0 & 3 & 0 & 3 & 0 & 3 \end{pmatrix} \\
&\quad + \begin{pmatrix} 0 & 0 & 0 & 0 & 2 & 2 & 0 & 0 & 0 & 0 & 2 & 2 \end{pmatrix} \\
&= \begin{pmatrix} 0 & 3 & 1 & 4 & 3 & 5 & 6 & 7 & 8 & 9 & 7 & 11 \end{pmatrix}, \\
a_{i_1 i_2 i_3} &= \left[\begin{bmatrix} 0 & 3 \\ 1 & 4 \\ 2 & 5 \end{bmatrix}\begin{bmatrix} 6 & 9 \\ 7 & 10 \\ 8 & 11 \end{bmatrix}\right]_{i_1=2, i_2=3, i_3=2}.
\end{aligned}$$

We consider the transposition from $(i_1,i_2,i_3)=(2,3,4)$ to $(i_1,i_3,i_2)=(4,2,3)$,

$$
\begin{aligned}
R^k &= \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \end{bmatrix}, \\
\left(M'\right)^{jk} &= \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}, \\
\left(u'\right)_{ijk} &= u\left(\left(\begin{bmatrix} 6 & 3 & 1 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \end{bmatrix}, \begin{bmatrix} 6 & 3 & 1 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \end{bmatrix} + \begin{bmatrix} 6 & 3 & 1 \end{bmatrix}\right)\right. \\
&\quad \left., i = i + \begin{bmatrix} 24 & 6 & 3 \end{bmatrix}\right) \\
&= u\left(\left(\begin{bmatrix} 0 & 0 & 0 \\ 6 & 3 & 1 \\ 12 & 6 & 2 \\ 18 & 9 & 3 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 \\ 6 & 3 & 1 \\ 12 & 6 & 2 \\ 18 & 9 & 3 \end{bmatrix} + \begin{bmatrix} 6 & 3 & 1 \end{bmatrix}\right), i = i + \begin{bmatrix} 24 & 6 & 4 \end{bmatrix}\right) \\
&= u\left(\left(\begin{bmatrix} 0 & 0 & 0 \\ 6 & 3 & 1 \\ 12 & 6 & 2 \\ 18 & 9 & 3 \end{bmatrix}, \begin{bmatrix} 6 & 3 & 1 \\ 12 & 6 & 2 \\ 18 & 9 & 3 \\ 24 & 12 & 4 \end{bmatrix}\right) i = i + \begin{bmatrix} 24 & 6 & 3 \end{bmatrix}\right), \\
a_i &= \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 12 & 4 \end{bmatrix} u\left(\begin{bmatrix} [0,6) & [0,3) & [0,1) \\ [6,12) & [3,6) & [1,2) \\ [12,18) & [6,9) & [2,3) \\ [18,24) & [9,12) & [3,4) \end{bmatrix}, i = i + \begin{bmatrix} 24 & 6 & 3 \end{bmatrix}\right) \\
&= \begin{array}{ccccc} 0 & + & 0 & + & 0 \\ +1*1u_i([6,12)) & + & 1*12u_i([3,6)) & + & 1*4u_i([1,2)) \\ +2*1u_i([12,18)) & + & 0 & + & 2*4u_i([2,3)) \\ 3*1u_i([18,24)) & + & 0 & + & 0 \end{array} \\
&= \begin{pmatrix} 0 & 4 & 8 & 12 & 16 & 20 & 1 & 5 & 9 & 13 & 17 & 21 \\ 2 & 6 & 10 & 14 & 18 & 22 & 3 & 7 & 11 & 15 & 19 & 23 \end{pmatrix}, \\
a_{i_1 i_2 i_3} &= \left[\begin{bmatrix} 0 & 4 & 8 \\ 12 & 16 & 20 \end{bmatrix}\begin{bmatrix} 1 & 5 & 9 \\ 13 & 17 & 21 \end{bmatrix}\begin{bmatrix} 2 & 6 & 10 \\ 14 & 18 & 22 \end{bmatrix}\begin{bmatrix} 3 & 7 & 11 \\ 15 & 19 & 23 \end{bmatrix}\right]_{i_1=4,i_2=2,i_3=3}.
\end{aligned}
$$

8 Implementation of transposition in the program

It is possible to implement the transposition by implementing Eq. 81, or by the defining the recursive function. The merit of implementing Eq. 81 is reducing the number of memory accesses compared with implementing recursive function. However, execution time is still wasted since many zero calculations must be performed. This can be avoided by using Null and by

skipping zero calculations. Thus, Eq. 81 can be redefined as follows,

$$B_j \equiv \left(b_j | b_j = \prod_{i=0}^{j-1} D_{i-1},\ D_{-1} = 1 \right), \tag{82}$$

$$C_j \equiv \left(c_j | c_j = \prod_{i=0}^{j-1} D_i \right), \tag{83}$$

$$R^k \equiv \mathbb{N}_0, \tag{84}$$

$$M^{jk} \equiv \begin{cases} 1_k & [0, D_j) \\ (\mathrm{Null})_k & \text{otherwise} \end{cases}, \tag{85}$$

$$u_{ijk} \equiv \begin{cases} 1_i & [M^{jk} B_j R_k, M^{jk} B_j R_k + B_l),\ i = i + C_j \\ 0_i & \text{otherwise} \end{cases}, \tag{86}$$

$$a_i = R^k B^j u_{ijk}. \tag{87}$$

In the above expression, M^{jk} is implemented as inner u_{ijk}.

9 Implementation example

These rules of tensor calculus for manipulating multidimensional variables were used in the following example. Also, we developed a Python and C++ program that performs the rules. This program is an expansion of the useful multidimensional array, such as Fortran's and MATLAB's array or Python's numpy ndarray. We will show examples of these applications by describing a calculation of magnetic field in the following section.

9.1 The calculation of the confinement magnetic field

Assuming the confinement coils are allocated like those in the Heliac, which is a device confines plasma with magnetic field, we calculate the magnetic field. All coils in the Heliac are circles, so drawing them is easy, but arranging the poloidal coils is somewhat difficult because they are oriented helically. However, by using the calculation rules described in 3.3, a series formulation and calculation will become easy, as follows.

If $\mathbf{x}$ and $\mathbf{x}'$ let position vector before and after transform, the affine transform is written as follows,

$$\begin{bmatrix} \mathbf{x}' \\ 1 \end{bmatrix} = \begin{bmatrix} & \mathsf{A} & & \mathbf{t} \\ 0 & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}, \tag{88}$$

where, A is a linear transform such as rotation, shear, and scaling.

Some coils with radius of r' are located at $\mathbf{x}'$ and $\mathbf{x}$. $\mathbf{x}'$ and $\mathbf{x}$ are shown as the following relation,

$$\mathsf{x}' = \mathsf{J}\mathsf{x}. \tag{89}$$

If some coils with r' are located at r', and if a plane in which these coils are located is a toroidal cross-section, and if a circle with radius r, which centered at the origin of the xy plane is defined, the transform can be written as follows,

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\phi & -\sin\phi & 0 & (R + r\cos\theta)\cos\phi \\ \sin\phi & \cos\phi & 0 & (R + r\cos\theta)\sin\phi \\ 0 & 0 & 1 & r\sin\theta \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r'\cos\theta' \\ 0 \\ r'\sin\theta' \\ 1 \end{bmatrix}. \tag{90}$$

therefore, given the information of R, r, θ, ϕ and r', the coil can be located,

Let $\theta' \equiv \{\theta'_n |\ 0 \le x \le 2\pi,\ \theta'_n = an,\ a \in \mathbb{R}\}_{n \in \mathbb{N}}$ and let $\mathsf{x}' = \mathsf{x}'_n(R, r, \theta, \phi, r', \theta'_n)$. If $\mathbf{x}$ is the position vector of a coil, and if a linear element that has both edge points $\mathbf{x}'_{n+1}$ and $\mathbf{x}'_n$ form magnetic field $\mathbf{B}(\mathbf{x}, \mathbf{x}'_{n+1}, \mathbf{x}'_n)$, then $\mathbf{B}$ can be written as follows,

$$\mathbf{B}\left(\mathbf{x},\mathbf{x'}_{n+1},\mathbf{x}'_n\right) = \frac{\mu_0 I\mathbf{s}\left(\mathbf{x}'_{n+1},\mathbf{x}'_n\right)\times\mathbf{R}\left(\mathbf{x},\mathbf{x}'_{n+1},\mathbf{x}'_n\right)}{4\pi s\left(\mathbf{R}\left(\mathbf{x},\mathbf{x}'_{n+1},\mathbf{x}'_n\right)\right)^2} \left(\cos\left(\theta_1\left(\mathbf{x},\mathbf{x}'_{n+1},\mathbf{x}'_n\right)\right)-\cos\left(\theta_2\left(\mathbf{x},\mathbf{x}'_{n+1},\mathbf{x}'_n\right)\right)\right), \tag{91}$$

$$\mathbf{s}\left(\mathbf{x}'_{n+1},\mathbf{x}'_n\right) = \mathbf{x}'_{n+1}-\mathbf{x}'_n, \tag{92}$$

$$\alpha\left(\mathbf{x},\mathbf{x}'_{n+1}\right) = \mathbf{x}-\mathbf{x}'_{n+1}, \tag{93}$$

$$\beta\left(\mathbf{x},\mathbf{x}'_n\right) = \mathbf{x}-\mathbf{x}'_n, \tag{94}$$

$$\mathbf{R}_i\left(\mathbf{x},\mathbf{x}'_{n+1},\mathbf{x}'_n\right) = \alpha-\frac{\alpha\cos\theta_{1_i}}{s}\mathbf{s}, \tag{95}$$

$$\cos\left(\theta_1\left(\mathbf{x},\mathbf{x}'_{n+1},\mathbf{x}'_n\right)\right) = \frac{\alpha\cdot(-\mathbf{s})}{\alpha s}, \tag{96}$$

$$\cos\left(\theta_2\left(\mathbf{x},\mathbf{x}'_{n+1},\mathbf{x}'_n\right)\right) = \frac{\beta\cdot\mathbf{s}}{\beta s}. \tag{97}$$

The summation of all $\mathbf{B}$ at $\mathbf{x}'$ linear elements can be written as follows,

$$\mathbf{B}(\mathbf{x}) = \sum_{n=0}^{|\theta'|-1}\mathbf{B}\left(\mathbf{x},\mathbf{x'}_{n+1},\mathbf{x}'_n\right), \tag{98}$$

where, $|\theta'|$ is the number of θ'. If the number of i of coils exist, let $X'\equiv\{\mathbf{x}'_i\}_{i\in\mathbb{N}}$, then $\mathbf{B}$ can be written as follows,

$$\mathbf{B}(\mathbf{x}) = \sum_{\mathbf{x}'_i\in X'}\mathbf{B}\left(\mathbf{x},\mathbf{x}'_i\right). \tag{99}$$

Now we try to write the coil allocation and calculation of $\mathbf{B}$ are written, by using our developed tensor rules. Equation 3 and Eqs. 91 through 97 can be rewritten as follows,

$$\left(x'\right)_{ijk} = J^{j'}_{ijk}x_{ij'k}, \tag{100}$$

$$B_{ijk} = \frac{\mu_0 I\varepsilon^{lm}_j s_{ilk}R_{imk}}{4\pi s_{ijk}g^{pq}R_{ipk}R_{iqk}} \left(\frac{-g^{pq}\alpha_{ipk}s_{iqk}}{\sqrt{g^{pq}\alpha_{ipk}\alpha_{iqk}}\sqrt{g^{pq}s_{ipk}s_{iqk}}}-\frac{g^{pq}\beta_{ipk}s_{iqk}}{\sqrt{g^{pq}\beta_{ipk}\beta_{iqk}}\sqrt{g^{pq}s_{ipk}s_{iqk}}}\right), \tag{101}$$

$$s_{ijk} = x'_{ij(k+1)}-x'_{ijk}, \tag{102}$$

$$\alpha_{ijk} = x_{ijk}-x'_{ij(k+1)}, \tag{103}$$

$$\beta_{ijk} = x_{ijk}-x'_{ijk}, \tag{104}$$

$$R_{ijk} = \alpha_{ijk}-\frac{\sqrt{g^{pq}\alpha_{ipk}\alpha_{iqk}}\frac{-g^{pq}\alpha_{ipk}s_{iqk}}{\sqrt{g^{pq}\alpha_{ipk}\alpha_{iqk}}\sqrt{g^{pq}s_{ipk}s_{iqk}}}}{\sqrt{g^{pq}s_{ipk}s_{iqk}}}s_{ijk} = \alpha_{ijk}+\frac{g^{pq}\alpha_{ipk}s_{iqk}}{g^{pq}s_{ipk}s_{iqk}}s_{ijk}, \tag{105}$$

$$g^{pq} = \delta^{pq}, \tag{106}$$

where, i is the coil number, j is the element of space vector, k is the number of the linear elements of coils, p and q are ad-hoc variable, g is metric, and δ is Kronecker delta. If B_{ijk} is summed for i and j, then,

$$B_j = 1^{ik}B_{ijk}. \tag{107}$$

Using the developed rules and the program, Eq. 100 and Eqs. 101 to 106 can be implemented into program almost directly as shown in Fig. 1 and 2.

The results of the coil allocation and field calculations are shown in Fig. 3 and 4. However, Fig. 4 is not the magnetic field, but the poloidal cross-section of magnetic surfaces with magnetic field line tracing, which is needed to calculate the magnetic field.

```
In [ ]: def toroidal_coils(r=0.07, R=0.48, rc=0.05, pitch=15, dtheta=0.1, coilnum=27):

            d = 2*pi/coilnum
            phi = arange(0,2*pi, d)

            theta = tensor(arange(0,2*pi+dtheta,dtheta),"i", "d")
            theta2 = tensor(phi,"k", "d")
            phi = tensor(phi,"k", "d")


            xc = tensor([r * t_cos(theta), 0, r * t_sin(theta), ot], idx="j", ud="d")
            J = tensor([ t_cos(phi), - t_sin(phi), 0, (R + rc * t_cos(pitch * theta2)) * t_cos(phi)],
                       [ t_sin(phi), t_cos(phi),   0, (R + rc * t_cos(pitch * theta2)) * t_sin(phi)],
                       [ 0,          0,            1, rc * t_sin(pitch * theta2)],
                       [ 0,          0,            0, 1], "j", "d")
            xc = xc * J
            xc.transpose(["j",0])
            xc = xc[1:-1,]
            return xc
```

Figure 1: The program for allocating toroidal coils (the scripts are written using Python).

```
In [2]: def calc_B(x,xc, I=300):
            mu = 4 * pi * 10 ** -4
            o = tensor(ones(3), idx="j", ud="u")
            ep = tensor_ps(3, idx="mlj", ud="uud")

            I.transpose(["s", 0])
            I = I[1:]
            xc.transpose(["s", 0])
            s = xc[1:] - xc[:-1,]

            a = x - xc[:-1,]
            b = x - xc[1:]

            a_norm = t_sqrt((a ** 2).sum("j"))
            b_norm = t_sqrt((b ** 2).sum("j"))
            s_norm = t_sqrt((s ** 2).sum("j"))

            cosa =  a.cud("j") * s / (a_norm * s_norm)
            cosb =  b.cud("j") * s / (b_norm * s_norm)
            R = a - a_norm * cosa * s / s_norm

            B = (ep *  s.cidx("j", "m") * R.cidx("j", "l"))  * mu * I / ((R ** 2).sum("j") * 4 * pi) * (cosa - cosb)
            return B
```

Figure 2: The program for calculating the magnetic field (the scripts are written using Python).

10 Summary

Novelly developed rules and a program for tensor calculus were described. Even as computer power is enhanced, human ability will change little. However, the data we handle is increasing and calculation is becoming more complex yearly. This study supports any notation that can reduce the amount of values to consider or that can reduce the amount of coding in a program. For these reason, we developed these rules and program.

References

[1] David Kay. *Schaums Outline of Tensor Calculus (Schaum's Outline Series)*. McGraw-Hill, 1 edition, 2 2011.

[2] William D. D'haeseleer, William N.G. Hitchon, James D. Callen, and J. Leon Shohet. *Flux Coordinates and Magnetic Field Structure: A Guide to a Fundamental Tool of Plasma Theory (Scientific Computation)*. Springer, softcover reprint of the original 1st ed. 1991 edition, 12 2011.

[3] George B. Arfken and Hans J. Weber. *Mathematical Methods for Physicists, Seventh Edition: A Comprehensive Guide*. Academic Press, 7 edition, 1 2012.

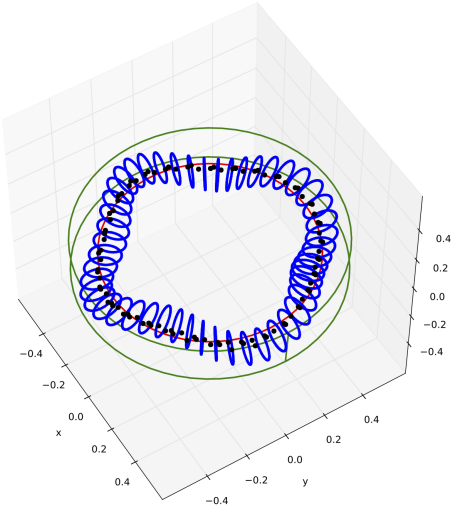

Figure 3: Coil allocation test.

Figure 4: Poloidal cross section of magnetic surfaces.