Maxima によるテンソル計算

足立健朗

1. テンソルの多重リストによる表現と、テンソルの基本演算実装の試み

数体 $\mathbb{K}$ 上のベクトル空間 V, W に対して、V と W のテンソル積 (空間) $V \otimes W$ とは、以下の条件を満たすものとして特徴づけられる $\mathbb{K}$ 上のベクトル空間である。

(1) 双線形写像

$$\iota \colon V \times W \longrightarrow V \otimes W$$

が存在して、

(2) 任意の $\mathbb{K}$ 上のベクトル空間 X と、任意の双線形写像 $f \colon V \times W \to X$ に対して、線形写像 $F \colon V \otimes W \to X$ が存在して $f = F \circ \iota$ が成り立つ。すなわち、図式

$$\begin{array}{ccc} V \times W & \xrightarrow{\iota} & V \otimes W \\ & \searrow^{f} & \downarrow{\scriptstyle F} \\ & & X \end{array}$$

が可換になる。

実際には $V \otimes W$ は一意的には定まらない。もし、$((V \otimes W)', \iota')$ が上の条件を満たすものとすると、同形写像 $\varphi \colon V \otimes W \to (V \otimes W)'$ が存在して、$\iota' = \varphi \circ \iota$ が成り立つ。したがって、$V \otimes W$ は同形類の意味では一意的である。

$\mathbb{K}$ 上の k 個のベクトル空間 $V_1, V_2, \ldots, V_k$ に対しても、そのテンソル空間 $V_1 \otimes V_2 \otimes \cdots V_k$ が上の条件の双線形という言葉を多重線形に置き換えることで同様に定義されるが、ここでは、

$$V_1 \otimes V_2 \otimes \cdots V_k := ((V_1 \otimes V_2) \otimes V_3) \cdots \otimes V_k)$$

(このように、括弧の位置を固定しておくと以下の話しとの整合性が保たれる。)

ベクトル空間 V, W にそれぞれ基底 $(\mathbf{v}_1, \ldots, \mathbf{v}_r)$, $(r = \dim V)$, $(\mathbf{w}_1, \ldots, \mathbf{w}_s)$, $(s = \dim W)$ が与えられている場合には、$V \otimes W$ を (形式的な記号からなる) 基底

$$\left(\mathbf{v}_1 \otimes \mathbf{w}_1, \mathbf{v}_1 \otimes \mathbf{w}_2, \ldots, \mathbf{v}_1 \otimes \mathbf{w}_s, \ldots, \mathbf{v}_r, \otimes \mathbf{w}_1, \ldots, \mathbf{v}_r \otimes \mathbf{w}_s\right)$$

によって張られるベクトル空間として構成的に定義できる。(後で **Maxima** 上でのテンソルの実装を考えて、基底の並びを辞書式に選んだ。) $V \otimes W$ の任意の元 f は $V \otimes W$ のテンソルと呼ばれる。テンソル f は、この基底により、

$$f = \sum_{i=1}^{r} \sum_{j=1}^{s} f^{ij}\, \mathbf{v}_i \otimes \mathbf{w}_j, \qquad (f^{ij} \in \mathbb{K},\, \forall i, \forall j)$$

Date: 2007/7/22, 2nd version.

と表すことができる。

同様に、$V_1, V_2, \ldots, V_k$ の基底を、それぞれ $(\mathbf{v}_{i_1}^{(1)})_{1\leq i_1\leq r_1}, (\mathbf{v}_{i_2}^{(2)})_{1\leq i_2\leq r_2}, \ldots, (\mathbf{v}_{i_k}^{(k)})_{1\leq i_k\leq r_k}$, $(r_1 = \dim\, V_1, r_2 = \dim\, V_2, \ldots, r_k = \dim\, V_k)$ とすれば、$V_1 \otimes V_2 \otimes \cdots \otimes V_k$ は基底

$$(\mathbf{v}_{i_1}^{(1)} \otimes \mathbf{v}_{i_2}^{(2)} \otimes \ldots \otimes \mathbf{v}_{i_k}^{(k)})_{1\leq i_1\leq r_1, 1\leq i_2\leq r_2, \ldots, 1\leq k\leq r_k}$$

(ただし、$i_1, i_2, \ldots, i_k$ について辞書式順序で並べる。つまり、順序のついた基底付きのベクトル空間として考える。) によって張られるベクトル空間として構成的に定義される。任意のテンソル $f \in V_1 \otimes V_2 \otimes \cdots \otimes V_k$ は

$$f = \sum_{i_1=1}^{r_1}\sum_{i_2=1}^{r_2}\cdots\sum_{i_k=1}^{r_k} f^{i_1 i_2 \cdots i_k}\, \mathbf{v}_{i_1} \otimes \mathbf{v}_{i_2} \otimes \cdots \otimes \mathbf{v}_{i_k}, \qquad (f^{i_1 i_2 \cdots i_k} \in \mathbb{K}, \forall i_1, \forall i_2, \ldots, \forall i_k)$$

と表される。この文書は、**Maxima** 上でテンソルを扱うことを目標としているので、テンソル空間としては

$$\mathbb{R}^{r_1} \otimes \mathbb{R}^{r_2} \otimes \cdots \otimes \mathbb{R}^{r_1}$$

あるいは

$$\mathbb{C}^{r_1} \otimes \mathbb{C}^{r_2} \otimes \cdots \otimes \mathbb{C}^{r_k}$$

さらには、

$$V_1 \otimes V_2 \otimes \cdots \otimes V_k$$

において、各 V_j が $\mathbb{R}$ または $\mathbb{C}$ 上の有限生成多項式環上の加群である場合だけを想定する。以下の説明では $\mathbb{R}^{r_1} \otimes \mathbb{R}^{r_2} \otimes \cdots \otimes \mathbb{R}^{r_k}$ を用いることにする。各 $\mathbb{R}^{r_j}$ においては、標準基底

$$\mathcal{E}^{(r_j)} = (\mathbf{e}_1^{(r_j)}, \ldots, \mathbf{e}_{r_j}^{(r_j)})$$

を考える。したがって、任意のテンソル $f \in \mathbb{R}^{r_1} \otimes \mathbb{R}^{r_2} \otimes \cdots \otimes \mathbb{R}^{r_k}$ を

$$f = \sum_{i_1=1}^{r_1}\sum_{i_2=1}^{r_2}\cdots\sum_{i_k=1}^{r_k} f^{i_1 i_2 \cdots i_k}\, \mathbf{e}_{i_1}^{(r_1)} \otimes \mathbf{e}_{i_2}^{(r_2)} \otimes \cdots \otimes \mathbf{e}_{i_k}^{(r_k)}, \qquad (f^{i_1 i_2 \cdots i_k} \in \mathbb{R}, \forall i_1, \forall i_2, \ldots, \forall i_k)$$

を表す。

このように表示されたテンソルを **Maxima** で表現するためのデータ型として多重リストを用いることにする。後ろにある添字ほど、深い入れ子内での位置を指し示すものと約束する。例えば、$f \in \mathbb{R}^2 \otimes \mathbb{R}^3$,

$$f = \sum_{i_1=1}^{2}\sum_{i_2=1}^{3} f^{i_1 i_2}\, \mathbf{e}_{i_1} \otimes \mathbf{e}_{i_2}$$

ならば、f の **Maxima** 上での表現は

$$[[f^{11}, f^{12}, f^{13}], [f^{21}, f^{22}, f^{23}]]$$

であり、逆に各成分 $f^{i_1 i_2}$ は **Maxima** 上では

```
f[i_1][i_2];
```

で参照できる。また、例えば $f \in \mathbb{R}^2 \otimes \mathbb{R}^2 \otimes \mathbb{R}^3$,

$$f = \sum_{i_3}^{3} \sum_{i_2}^{2} \sum_{i_1}^{2} f^{i_1 i_2 i_3}\, \mathbf{e}_{i_1} \otimes \mathbf{e}_{i_2} \otimes \mathbf{e}_{i_3}$$

ならば、その **Maxima** 上での表現は

$$[[[f^{111}, f^{112}, f^{113}], [f^{121}, f^{122}, f^{123}]], [[f^{211}, f^{212}, f^{213}], [f^{221}, f^{222}, f^{223}]]]$$

であり、各成分 $f^{i_1 i_2 i_3}$ は **Maxima** 上では

```
f[i_1][i_2][i_3];
```

で参照できる。

テンソル $f \in \mathbb{R}^{r_1} \otimes \mathbb{R}^{r_2} \otimes \cdots \otimes \mathbb{R}^{r_k}$ の型 (type of tensor) は、 $(r_1, r_2, \ldots, r_k)$ であると言うことにする。**Maxima** では、型を正整数のリスト $[r_1, r_2, \ldots, r_k]$ で表されるものとする。与えられたテンソルの型を調べる **Maxima** の関数として **tensortype** を、条件式

$$\mathrm{tensortype}(f) = [r_1, r_2, \ldots, r_k]$$

が真であるように実装する。

同じ型を持つテンソルどうしの和、テンソルのスカラー倍は、テンソルの型を変えない演算である。これらの演算に対して、新たに Maxima の演算子を定義する必要はない。$f, g \in \mathbb{R}^{r_1} \otimes \mathbb{R}^{r_2} \otimes \cdots \otimes \mathbb{R}^{r_k}$, $a \in \mathbb{R}$ に対して、$f + g$ および af を **Maxima** で実現するには、**Maxima** 上で、それぞれ、式

```
f + g
```

および

```
a * f
```

を評価すればよい。

テンソルを考える上で重要なのが テンソルどうしの積 (テンソル積) である。すなわち、$f \in \mathbb{R}^{r_1} \otimes \cdots \otimes \mathbb{R}^{r_k}$, $g \in \mathbb{R}^{s_1} \otimes \cdots \otimes \mathbb{R}^{s_l}$ に対して、$f \otimes g \in \mathbb{R}^{r_1} \otimes \cdots \otimes \mathbb{R}^{r_k} \otimes \mathbb{R}^{s_1} \otimes \cdots \otimes \mathbb{R}^{s_l}$ を対応させる演算である。上述のようにテンソルを多重リストとして扱うとしたことと、**Maxima** が Common Lisp で記述され、それ自身 Lisp 的な操作が可能なことから、テンソル積を **Maxima** の演算子として、シンプルに実装できることを例示する。テンソル積 $f \otimes g$ を **Maxima** では、

```
f @x@ g
```

と表現することにする。

テンソル積とともに、応用上重要なテンソルに関する演算として、添字の縮約 (contraction) と呼ばれるものがある。まずは、

$$\begin{array}{ccc} \mathbb{R}^{r_1} \otimes \cdots \otimes \mathbb{R}^{r_{k-1}} \otimes \mathbb{R}^{r_k} \times \mathbb{R}^{r_k} & \xrightarrow{\ \lfloor\_\ \rfloor\ } & \mathbb{R}^{r_1} \otimes \cdots \otimes \mathbb{R}^{r_{k-1}} \\ f \times \mathbf{v} & \longmapsto & f \lfloor\_ \mathbf{v} \end{array}$$

(ベクトルを右からテンソル積して、その後縮約する) を一つの例として実装する。**Maxima** では $\mathbf{v} \lfloor\_ f$ を

```
v @tvc f
```

と表すことにする。

一般の縮約と言うのは、条件 $r_i = r_j,\ (i < j)$ が満たされているとき、

$$\mathbb{R}^{r_1} \otimes \cdots \otimes \mathbb{R}^{r_i} \otimes \cdots \otimes \mathbb{R}^{r_j} \otimes \cdots \otimes \mathbb{R}^{k} \longrightarrow \mathbb{R}^{r_1} \otimes \cdots \otimes \widehat{\mathbb{R}^{r_i}} \otimes \cdots \otimes \widehat{\mathbb{R}^{r_j}} \otimes \cdots \otimes \mathbb{R}^{k}$$

と言う写像のことである。(ただし $\widehat{V}$ はその部分を取り除くことを意味する。) これは、テンソルをその i, j 成分については正方行列 (成分は $k-2$ 階のテンソル) と考えトレースを取ったものである。これを、**Maxima** 上では

```
cntrct(f)
```

という関数として実現する。

2. Maxima によるテンソル計算の実装例

```
/*********************************************
 *                  tensor.mac                 *
 *                                             *
 *     general tensor operations on Maxima     *
 *                                             *
 *     (c) Kenrou Adachi, 2007/1/12-           *
 *         Version 2.0    2007/3/7             *
 *                                             *
 *********************************************/

load(eigen)$  /* inprod の使用に必要 */

/* tensortype, tensorrank, zerotensor, tensorp sametensortypep */

/* 多重リストの car 部の length を再帰的に調べていき リストにする。 */
/* テンソルでなくても動いてしまう。 */

pseudotype(f) := block([ttyp, ttmp],
    ttmp : copylist(f),              /* f のコピー */
    ttyp : cons(length(ttmp), []),   /* 疑似テンソルタイプの表す逆転リスト */

    rectyp(typ, tmp) := block([ftmp],              /* 再帰ルーチン */
        ftmp : first(tmp),                         /* tmp の car */
        if listp(ftmp) then         /* さらにリストが多重になっているならば */
            rectyp(cons(length(ftmp), typ), ftmp) /* ftmp の長さを typ に cons */
       else                                       /* ftmp が式ならば */
           typ                                    /* なにもせずに typ を返す。
*/
    ),

    reverse(rectyp(ttyp, ttmp))     /* リストを逆転して求める結果を得る。 */
)$


/* 本当のテンソルに対してテンソルの型を返す。 */
/* テンソルでないときには nil を返す。       */

tensortype(f) := block([typ, za, ta],  /* depending on zerotensor below */
    typ : pseudotype(f),           /* f の疑似テンソル型 */
    za : zerotensor(typ),          /* typ をテンソル型とする ゼロテンソル */
    ta : 0 * f,                    /* f の要素をすべて 0 にする。 */
```

```
    if za = ta then                 /* f が本当にテンソルならば za = ta に */
        typ                         /* なるはず。*/
    else
        nil
)$


/* 与えられた型のテンソルを返す */

zerotensor(ttype) := block([n, rtyp, rlst, xero],
    n : length(ttype),                  /* テンソルの階数 */
    rtyp : reverse(ttype),              /* テンソル型の逆リスト */
    rlst : first(rtyp),                 /* 再深部のリストの長さ */

    zerolist(m) := block(               /* [0,0,...,0] (長さ m) を作る。*/
        if m = 0 then
            []
        else
            cons(0, zerolist(m - 1))
    ),

    rec(m, rz) := block(                /* [rz,rz,...,rz] (長さ m) を作る。 */
        if m = 0 then
            []
        else
            cons(rz, rec(m - 1, rz))
    ),

    for i : 1 thru n do             /* ttype = [r1,r2,...,rn], rlst=rn   */
        if i = 1 then               /* のとき、まず長さ rlst の [0,0,...,0] */
        (                           /* を作り、それを xero とする。         */
            xero : zerolist(rlst),  /* つぎに、長さr(n-1)の [xero,...,xero] */
            rtyp : rest(rtyp)       /* を作り、改めてそれを xero とする。*/
        )                           /* 以下、帰納的にこれを繰り返す。        */
        else
        (
            rlst : first(rtyp),
            xero : rec(rlst, xero),
            rtyp : rest(rtyp)
        ),
```

```
    xero
)$


/* 対象がテンソルかどうか判定する */

tensorp(a) := block([za, ta],
    za : zerotensor(pseudotype(a)),
    ta : 0 * a,

    if za = ta then                /* 考え方は tensortype と同じ */
        TRUE
    else
        FALSE
)$


/* 与えられた２つの対象が同じ型のテンソルかどうかを調べる。 */

sametensortypep(a, b) := block([za : 0 * a,  zb : 0 * b],
    if za = zb then               /* 考え方は tensortype と同じ */
        TRUE
    else
        FALSE
)$


/* テンソルの階数を返す。 */

tensorrank(a) := length(tensortype(a))$


/* tensorproduct */

/* a の各リーフに b のツリーをぶらさげる。(リーフの値を b の各リーフに乗じて) */
/* a, b がテンソルかどうかをチェックしない */
/* a, b がテンソルでないときの動作は私は知らない。(興味がない) */

pseudoproduct(a, b) := block(
    if not listp(first(a)) then             /* a のすぐ下のノードが式ならば */
        maplist(lambda([x], x * b), a)      /* それはリーフである。         */
    else                                    /* そうでなければ、それは再び   */
```

```
        maplist(lambda([x], pseudoproduct(x, b)), a)   /* テンソルである  */
)$

/* a, b が本当のテンソルのとき、pseudoproduct する。 */
/* 結果としてそれがテンソル積を表す。                 */

tensorproduct(a, b) := block(
    if tensorp(a) and tensorp(b) then
        pseudoproduct(a, b)
    else
        nil
)$


infix("@x@")$


a @x@ b := tensorproduct(a, b)$


/* exchangetensortype */


/* ルートのすぐ下のノードとその下のノードの役割を入れ換える       */
/* 最初の２つの入れ子を行列と考えたとき、転置を取ることに当たる。 */
/* したがって、階数１のテンソルでは動かない。                     */

exchinext(a) := block([b, c, ni, nip],
    ni  : length(a),
    nip : length(a[1]),
    b   : [],

    for j : 1 thru nip do
    (
        c : [],
        for k : 1 thru ni do
            c : endcons(a[k][j], c),
        b : endcons(c, b)
    ),
    b
)$
```

```
/* 第 i 段にある各ノードに exchinext を適用する。*/

exchnext(a, i) := block([n],
    n : tensorrank(a),

    if (i > n - 1) or (i < 1) then
    (
        print("Index is out of range"),
        FALSE
    )
    else if i = 1 then
        exchinext(a)
    else
        maplist(lambda([x], exchnext(x, i - 1)), a)
)$


/* テンソルの添字の入れ換え */

exchangetensortype(a, i, j) := block([b, sw],
    if not tensorp(a) then
    (
        print("Object is not a tensor"),
        FALSE
    )
    else if i = j then
    (
        print("Invalid indices"),
        FALSE
    )
    else if i > j then
        exchengetensortype(a, j, i)
    else if j = i + 1 then                /* f^{...i j...}            */
        exchnext(a, i)                    /* ---> f^{...j i...}       */
    else
    (
        b : copylist(a),
        for k : i thru j - 1 do           /* f^{...i...j...}          */
            b : exchnext(b, k),           /*   ---> f^{... ...j i...} */
        for k : j - 2 thru i step -1 do
            b : exchnext(b, k),           /*   ---> f^{...j...i...}   */
```

```
        b
    )
)$


/* ailias */

exchttype(a, i, j) := exchangetensortype(a, i, j)$


/* tensorcontract, contractlast2, tensorvectorcontract */

/* テンソルが正方行列であるか判定する。 */

squarematrixlikep(a) := block(
    if listp(a[1][1]) then
        FALSE
    else if not length(a) = length(a[1]) then
        FALSE
    else
        TRUE
)$


/* 正方行列に対してトレースを求める */

matrixtrace(a) := block([n, tra],
    if not squarematrixlikep(a) then
    (
        print("Indices are not square-matrix-like"),
        FALSE
    )
    else
    (
        n : length(a),
        tra : sum(a[i][i], i, 1, n),
        tra
    )
)$


/* a の底から2段目のノードが正方行列であると仮定して、  */
```

```
/* それを、トレースを値とするリーフに置き換える。        */
/* 結果として最後の２つの添字に対して縮約したことになる。 */

contractlast2(a) := block(
    if squarematrixlikep(a) then
        matrixtrace(a)
    else
        maplist(lambda([x], contractlast2(x)), a)
)$


/* 第 i, j 番目の添字を 第 (n-1), n 番目に変換しておいて */
/* contractlast2 を適用。 ri != rj ならば nil を返す。 */
/* 結果として第 i, j 番目の添字についての縮約となる。      */

tensorcontract(a, i, j) := block([n, b],
    n : tensorrank(a),

    if n = 0 then
    (
        print("Object is not a tensor"),
        FALSE
    )
    else if (i = j) or (i < 1) or (i > n) or (j < 1) or (j > n) then
    (
        print("Invalid indices"),
        FALSE
    )
    else if i > j then
        tensorcontract(a, j, i)
    else if i = n - 1 then
        contractlast2(a)
    else
    (
        b : copylist(a),
        if not j = n then
            for p : j thru n - 1 do
                b : exchnext(b, p),
        if not i = n - 1 then
            for p : i thru n - 2 do
                b : exchnext(b, p),
        b : contractlast2(b),
```

```
        b
    )
)$


/* alias */

cntrct(a, i, j) := tensorcontract(a, i, j)$


/* tensortensorcontract, tensorvectorcontract */


simplistp(f) := block(
    if not listp(first(f)) then
        TRUE
    else
        FALSE
)$


/* vector = tensorrank 1 tensor */
/* Warning: the followings don't check Consistency of Indices */

tensorvectorcontract(a, b) := block(
    if (not tensorrank(b) = 1) or (not tensorp(a)) then
    (
        print("Objects have invalid types"),
        FALSE
    )
    else if simplistp(a) then
        inprod(a, b)
    else if simplistp(first(a)) then
        maplist(lambda([x], inprod(x, b)), a)
    else
        maplist(lambda([x], tensorvectorcontract(x, b)), a)
)$


/* alias */

tvcontract(a, b) := tensorvectorcontract(a, b)$
```

```
infix("@tvc")$


a @tvc b := tensorvectorcontract(a, b)$


tensortensorcontract(a, b, i, j) := block([n],
    n : tensorrank(a),

    if (not n = 0) or (not tensorp(b)) then
    (
        print("Objects may not be tensor"),
        FALSE
    )
    else
        tensorcontract(a @x@ b, i, n + j)
)$


/* alias */

ttcontract(a, b, i, j) := tensortensorcontract(a, b, i, j)$



/* end of tensor.mac */
```

E-mail address: kenrou@yo.rim.or.jp