テンソル計算プログラミング言語としてのEgisonの設計

Oct 9th, 2020

Satoshi Egi

Rakuten Institute of Technology, Rakuten, Inc.

Rakuten

プログラミング言語Egison

- 「プログラムをエレガントに記述する新しい方法を開拓したい」という動機で作られたプログラミング言語.
- パターンマッチ指向プログラミング言語以外にも数式処理システムとしての側面をもつ.
- テンソル解析の計算を簡潔に表現するための言語機能を提供している.
- 今日はこれらのテンソル関連の言語機能の設計を紹介する.

数式処理システムとは

シンボリックな計算をサポートするプログラミング言語.

未束縛の変数はシンボルとして扱われる.

$$x + x \to 2x$$
$$(x + y)^2 \to x^2 + 2xy + y^2$$

上記の計算をEgisonでは下記のように実行できる.

```
x + x -- 2x
(x + y)^2 -- x^2 + 2 * y * x + y^2
```

目次

目次

目次

テンソルとは

- ベクトルや行列を一般化したもの.
- ベクトルは一階のテンソル，行列は二階のテンソル.
- n階のテンソルは，n次元配列により表現される.

$$(a_1 \quad a_2 \quad a_3)$$

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix}$$

$$\left(\begin{pmatrix} a_{111} & a_{112} & a_{113} \\ a_{121} & a_{122} & a_{123} \end{pmatrix} \begin{pmatrix} a_{211} & a_{212} & a_{213} \\ a_{221} & a_{222} & a_{223} \end{pmatrix} \begin{pmatrix} a_{311} & a_{312} & a_{313} \\ a_{321} & a_{322} & a_{323} \end{pmatrix} \right)$$

目次

テンソル計算の性質

テンソル積・アダマール積・内積の組み合わせでほとんどの計算は表現できる.

テンソル積:

$$\begin{pmatrix} a_1 & a_2 \end{pmatrix} \otimes \begin{pmatrix} b_1 & b_2 \end{pmatrix} = \begin{pmatrix} a_1 b_1 & a_1 b_2 \\ a_2 b_1 & a_2 b_2 \end{pmatrix}$$

アダマール積:

$$\begin{pmatrix} a_1 & a_2 \end{pmatrix} \circ \begin{pmatrix} b_1 & b_2 \end{pmatrix} = \begin{pmatrix} a_1 b_1 & a_2 b_2 \end{pmatrix}$$

内積:

$$\begin{pmatrix} a_1 & a_2 \end{pmatrix} \cdot \begin{pmatrix} b_1 & b_2 \end{pmatrix} = a_1 b_1 + a_2 b_2$$

例. 行列の掛け算

行列の掛け算は,

- Aの行とBの列についてはテンソル積,
- Aの列とBの行については内積

をとる演算と解釈できる.

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{pmatrix}$$

テンソルの添字記法

シンボリックな添字で，テンソル積・アダマール積・内積のどれかを指定する.

テンソル積:

$$(a_1 \quad a_2) \otimes \begin{pmatrix} b_1 & b_2 \end{pmatrix} = \begin{pmatrix} a_1 b_1 & a_1 b_2 \\ a_2 b_1 & a_2 b_2 \end{pmatrix}$$

アダマール積:

$$(a_1 \quad a_2) \circ \begin{pmatrix} b_1 & b_2 \end{pmatrix} = \begin{pmatrix} a_1 b_1 & a_2 b_2 \end{pmatrix}$$

内積:

$$(a_1 \quad a_2) \cdot \begin{pmatrix} b_1 & b_2 \end{pmatrix} = a_1 b_1 + a_2 b_2$$

テンソル積:

$$(a_1 \quad a_2)_i \begin{pmatrix} b_1 & b_2 \end{pmatrix}_j = \begin{pmatrix} a_1 b_1 & a_1 b_2 \\ a_2 b_1 & a_2 b_2 \end{pmatrix}_{ij}$$

アダマール積:

$$(a_1 \quad a_2)_i \begin{pmatrix} b_1 & b_2 \end{pmatrix}_i = \begin{pmatrix} a_1 b_1 & a_2 b_2 \end{pmatrix}_i$$

内積:

$$(a_1 \quad a_2)^i \begin{pmatrix} b_1 & b_2 \end{pmatrix}_i = a_1 b_1 + a_2 b_2$$

テンソルの添字記法: 行列の掛け算

行列の掛け算は,

- Aの行とBの列についてはテンソル積,
- Aの列とBの行については内積

をとる演算と解釈できる.

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}^{i}_{j} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}^{j}_{k} = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{pmatrix}^{i}_{k}$$

目次

対称と歪対称

対称: 上三角と下三角が同じ値.　（対角成分の値は何でもOK.）

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{12} & a_{22} & a_{23} \\ a_{13} & a_{23} & a_{33} \end{pmatrix}$$

歪対称: 上三角と下三角の符号が逆.　（対角成分は0.）

$$\begin{pmatrix} 0 & a_{12} & a_{13} \\ -a_{12} & 0 & a_{23} \\ -a_{13} & -a_{23} & 0 \end{pmatrix}$$

独立成分はだいたい半分になる.

テンソルの対称性

数学や物理で現れるテンソルは対称性をもつことが多い.

たとえばリーマン曲率テンソルという4階のテンソルは多くの対称性をもつ.

$$R_{abcd} = g_{ai}\left(\frac{\partial \Gamma^{i}{}_{bd}}{\partial x_c} - \frac{\partial \Gamma^{i}{}_{bc}}{\partial x_d} + \Gamma^{j}{}_{bd}\Gamma^{i}{}_{jc} - \Gamma^{j}{}_{bc}\Gamma^{i}{}_{jd}\right)$$

$$R_{abcd} = -R_{bacd} \qquad \Gamma^{a}{}_{bc} = \Gamma^{a}{}_{cb} \qquad g_{ab} = g_{ba}$$

$$R_{abcd} = -R_{abdc}$$

$$R_{abcd} = R_{cdab}$$

数学者は対称性を意識しながら計算を省略する.

目次

目次

添字記法の導入

Egisonは添字記法をサポートしている.

テンソル積:

$$(a_1 \quad a_2)_i \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}_j = \begin{pmatrix} a_1 b_1 & a_1 b_2 \\ a_2 b_1 & a_2 b_2 \end{pmatrix}_{ij}$$

```
[| a_1, a_2 |]_i . [| b_1, b_2 |]_j
-- [| [| a_1 * b_1, a_1 * b_2 |],
      [| a_2 * b_1, a_2 * b_2 |] |]_i_j
```

アダマール積:

$$(a_1 \quad a_2)_i \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}_i = \begin{pmatrix} a_1 b_1 & a_2 b_2 \end{pmatrix}_i$$

```
[| a_1, a_2 |]_i . [| b_1, b_2 |]_i
-- [| a_1 * b_1, a_2 * b_2 |]_i
```

内積:

$$(a_1 \quad a_2)^i \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}_i = a_1 b_1 + a_2 b_2$$

```
[| a_1, a_2 |]~i . [| b_1, b_2 |]_i
-- a_1 * b_1 + a_2 * b_2
```

添字記法を導入する手法

以下の3つのアイデアが組み合わせることにより，添字記法が導入できる.

- スカラー関数とテンソル関数
- スカラー関数を適用したときの添字の簡約規則
- テンソル関数には引数のテンソルの添字の情報も渡される

添字記法を導入する手法

以下の3つのアイデアが組み合わせることにより，添字記法が導入できる.

- **スカラー関数とテンソル関数**
- スカラー関数を適用したときの添字の簡約規則
- テンソル関数には引数のテンソルの添字の情報も渡される

スカラー関数とテンソル関数

スカラー関数は，自動的に成分ごとに処理をmapする.

```
(\$u$v -> (u, v)) [| a, b |]_i [| x, y |]_j
-- [| [| (a, x), (a, y) |],
      [| (b, x), (b, y) |] |]_i_j
```

テンソル関数は，テンソルをそのまま引数として受け取る.

```
(\%u %v -> (x, y)) [| a, b |]_i [| x, y |]_j
-- ([| a, b |]_i, [| x, y |]_j)
```

添字記法を導入する手法

以下の3つのアイデアが組み合わせることにより，添字記法が導入できる.

- スカラー関数とテンソル関数
- **スカラー関数を適用したときの添字の簡約規則**
- テンソル関数には引数のテンソルの添字の情報も渡される

スカラー関数

スカラー関数は自動的に成分ごとに処理をmapする.

```
(\$u $v -> (u, v)) [| a, b |]_i [| x, y |]_j
-- [| [| (a, x), (a, y) |],
      [| (b, x), (b, y) |] |]_i_j
```

異なるシンボルの添字の場合

```
(\$u $v -> (u, v)) [| a, b |]_i [| x, y |]_i
-- [| (a, x), (b, y) |]_i
```

同じシンボルで添字の上下も一致する場合

```
(\$u $v -> (u, v)) [| a, b |]~i [| x, y |]_i
-- [| (a, x), (b, y) |]~_i
```

同じシンボルであるが添字の上下が異なる場合

スカラー関数

スカラー関数は自動的に成分ごとに処理をmapする.

```
(\$u $v -> (u, v)) [| a, b |]_i [| x, y |]_j
-- [| [| (a, x), (a, y) |],
      [| (b, x), (b, y) |] |]_i_j
```

異なるシンボルの添字の場合

```
(\$u $v -> (u, v)) [| a, b |]_i [| x, y |]_i
-- [| (a, x), (b, y) |]_i
```

同じシンボルで添字の上下も一致する場合

```
(\$u $v -> (u, v)) [| a, b |]~i [| x, y |]_i
-- [| (a, x), (b, y) |]~_i
```

同じシンボルであるが添字の上下が異なる場合

スカラー関数

+ や * はスカラー関数として定義されている.

```
[| a, b |]_i * [| x, y |]_j
-- [| [| a * x, a * y |],
      [| b * x, b * y |] |]_i_j
```

異なるシンボルの添字の場合

```
[| a, b |]_i * [| x, y |]_i
-- [| a * x, b * y |]_i
```

同じシンボルで添字の上下も一致する場合

```
[| a, b |]~i * [| x, y |]_i
-- [| a * x, b * y |]~_i
```

同じシンボルであるが添字の上下が異なる場合

上下添字~_と組み込み関数contract

組み込み関数contract(縮約)は上下添字の部分をリストに変換する.

```
contract [| a * x, b * y |]~_i
-- [a * x, b * y]

contract [| [| a, x |], [|b, y |] |]~_i~_j
-- [a, x, b, y]

contract [| [| a, x |], [|b, y |] |]~_i_j
-- [[| a, x |]_j, [| b, y |]_j]
```

添字記法を導入する手法

以下の3つのアイデアが組み合わせることにより，添字記法が導入できる.

- スカラー関数とテンソル関数
- スカラー関数を適用したときの添字の簡約規則
- **テンソル関数には引数のテンソルの添字の情報も渡される**

テンソル関数

テンソル関数は，テンソルをそのまま引数として受け取る.

```
(\%u %v -> (u, v)) [| a, b |]_i [| x, y |]_j
-- ([| a, b |]_i, [| x, y |]_j)
```

添字の情報も一緒に渡される.

テンソルのかけ算や，行列式の計算など，成分ごとにmapしては表現できない関数は，テンソル関数として定義する.

テンソル関数の例: テンソルのかけ算

```
def (.) %t1 %t2 := sum (contract (t1 * t2))
```

テンソル関数の例: テンソルのかけ算

```
def (.) %t1 %t2 := sum (contract (t1 * t2))
```

```
[| a, b |]~i . [| x, y |]_i
```

テンソル関数の例: テンソルのかけ算

```
def (.) %t1 %t2 := sum (contract (t1 * t2))
```

```
[| a, b |]~i . [| x, y |]_i
```

```
sum (contract ([| a, b |]~i * [| x, y |]_i))
```

テンソル関数の例: テンソルのかけ算

```
def (.) %t1 %t2 := sum (contract (t1 * t2))
```

```
[| a, b |]~i . [| x, y |]_i
```

```
sum (contract ([| a, b |]~i * [| x, y |]_i))
```

```
sum (contract [| a * x, b * y |]~_i)
```

テンソル関数の例: テンソルのかけ算

```
def (.) %t1 %t2 := sum (contract (t1 * t2))
```

```
[| a, b |]~i . [| x, y |]_i
```

```
sum (contract ([| a, b |]~i * [| x, y |]_i))
```

```
sum (contract [| a * x, b * y |]~_i)
```

```
sum [a * x, b * y]
```

テンソル関数の例: テンソルのかけ算

```
def (.) %t1 %t2 := sum (contract (t1 * t2))
```

```
[| a, b |]~i . [| x, y |]_i
```

```
sum (contract ([| a, b |]~i * [| x, y |]_i))
```

```
sum (contract [| a * x, b * y |]~_i)
```

```
sum [a * x, b * y]
```

```
a * x + b * y
```

高階関数と添字記法

高階関数と添字記法を組み合わせることができることがEgisonの大きな特徴.

$$g^{i_1 j_1} g^{i_2 j_2} \ldots g^{i_n j_n}$$

```
foldl (.) 1 (map (\x -> g~(i_x)~(j_x)) [1..n])
```

テンソルの宣言

添字の型（上下）によって異なる値を束縛できる.

添字の型によって異なる値を束縛したいときは，添字の型を指定する.

```
def v~ := [| a, b, c |]
def v_ := [| x, y, z |]

v~1 -- a
v_2 -- y
```

テンソルの宣言

テンソルの宣言のときに添字のシンボルを指定することもできる.

$$R_{abcd} = g_{ai}\left(\frac{\partial \Gamma^{i}{}_{bd}}{\partial x_c} - \frac{\partial \Gamma^{i}{}_{bc}}{\partial x_d} + \Gamma^{j}{}_{bd}\Gamma^{i}{}_{jc} - \Gamma^{j}{}_{bc}\Gamma^{i}{}_{jd}\right)$$

```
def R_a_b_c_d := withSymbols [i, j]
  g_a_i . (∂/∂ Γ~i_b_d x~c - ∂/∂ Γ~i_b_c x~d
             + Γ~j_b_d . Γ~i_j_c - Γ~j_b_c . Γ~i_j_d)
```

```
def R____ := withSymbols [a, b, c, d] (transpose [a, b, c, d]
  withSymbols [i, j]
    g_a_i . (∂/∂ Γ~i_b_d x~c - ∂/∂ Γ~i_b_c x~d
               + Γ~j_b_d . Γ~i_j_c - Γ~j_b_c . Γ~i_j_d))
```

目次

対称・歪対称テンソル

テンソルに対称性があるとき，上半分（または下半分）の計算を省略したい.

```
def T_i_j := generateTensor (\i j -> i + j) [3, 3]
-- [| [| 2, 3, 4 |],
      [| 3, 4, 5 |],
      [| 4, 5, 6 |] |]_#_#

def U_i_j := generateTensor (\i j -> i - j) [3, 3]
-- [| [| 0, -1, -2 |],
      [| 1, 0, -1 |],
      [| 2, 1, 0 |] |]_#_#
```

対称・歪対称テンソルの宣言

テンソルを定義するときに，テンソルが対称・歪対称であることを宣言できる.

```
def T{_i_j} := generateTensor (\i j -> i + j) [3, 3]
-- [| [| 2, 3, 4 |],
      [| 3, 4, 5 |],
      [| 4, 5, 6 |] |]_#_#
```

{}で囲むことで対称性を宣言する.

```
def U[_i_j] := generateTensor (\i j -> i - j) [3, 3]
-- [| [| 0, -1, -2 |],
      [| 1, 0, -1 |],
      [| 2, 1, 0 |] |]_#_#
```

[]で囲むことで反対称性を宣言する.

複数の対称性をもつテンソル

複数の対称性も同時に宣言できる.

```
def R[_a_b][_c_d] := withSymbols [i, j]
  g_a_i . (∂/∂ Γ~i_b_d x~c - ∂/∂ Γ~i_b_c x~d
                + Γ~j_b_d . Γ~i_j_c - Γ~j_b_c . Γ~i_j_d)
```

$$R_{abcd} = -R_{bacd}$$

$$R_{abcd} = -R_{abdc}$$

$$R_{abcd} = R_{cdab}$$

複数の対称性をもつテンソル

複数の対称性も同時に宣言できる.

```
def R{(_a_b)(_c_d)} := withSymbols [i, j]
  g_a_i . (∂/∂ Γ~i_b_d x~c - ∂/∂ Γ~i_b_c x~d
                + Γ~j_b_d . Γ~i_j_c - Γ~j_b_c . Γ~i_j_d)
```

$$R_{abcd} = -R_{bacd}$$

$$R_{abcd} = -R_{abdc}$$

$$R_{abcd} = R_{cdab}$$

()で囲むことで複数の添字をまとめる.

複数の対称性をもつテンソル

下記3つの対称性を同時に宣言できる.

```
def R{[_a_b][_c_d]} := withSymbols [i, j]
  g_a_i . (∂/∂ Γ~i_b_d x~c - ∂/∂ Γ~i_b_c x~d
                + Γ~j_b_d . Γ~i_j_c - Γ~j_b_c . Γ~i_j_d)
```

$$R_{abcd} = -R_{bacd}$$

$$R_{abcd} = -R_{abdc}$$

$$R_{abcd} = R_{cdab}$$

テンソル対称性の宣言の実装

下記のようにdesugarする.

```
def X{_i_j} := ...
```

```
def X_i_j :=
  let tmpX_i_j := ... in
    generateTensor
      (\i j -> if i > j then tmpX_j_i
               else tmpX_i_j)
      (tensorShape tmpX_#_#)
```

テンソルの成分が遅延評価されるため,
tmpXについて上半分の成分しか評価さ
れない.

目次

微分幾何学の研究者との共同研究

Thursonという数学者が考えた複雑な多様体のWodzicki-Chern-Simons不変量というものを計算した．東北大学の前田先生とBoston大学のRosenberg先生との共同研究で現在論文執筆中.

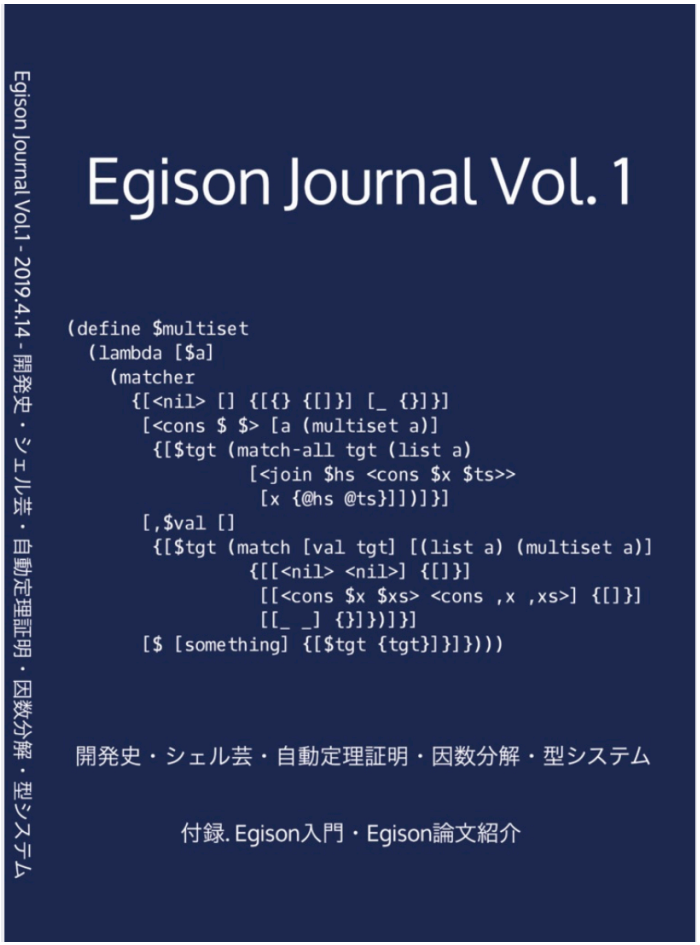
- https://github.com/egisatoshi/EMR-Paper-Computation

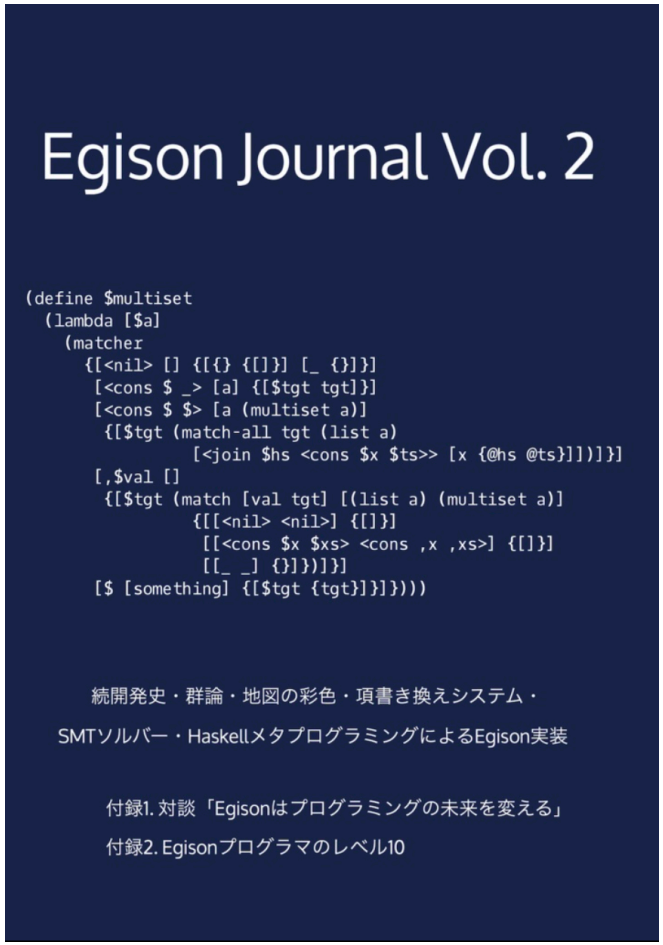
まとめ

プログラミング言語Egisonは,

- 高階関数と組み合わせることができるテンソルの添字記法をサポートしている.
- テンソルの対称性を宣言する便利な記法をサポートしている.

Egison Journal Vol. 1 and 2 を BOOTH にて発売中！





Rakuten