プログラミング言語Egisonで計算する微分幾何
Calculations in Differential Geometry with the Egison Programming Language

江木聡志

Satoshi Egi *

楽天技術研究所

Rakuten Institute of Technology

Abstract

Egison はユーザー定義関数を含む任意の関数について，特別な記述なしにテンソルの添字記法を使うことができるプログラミング言語である．微分形式を使った計算の簡潔な記述もサポートしており，ウェッジ積，外微分，ホッジ作用素，内部積をはじめとする微分形式についての作用素も Egison で簡潔に定義できる．それゆえ，Egison を使えば，添字記法を多用する微分幾何の計算を，既存の数式処理システムよりも，簡潔に表現できる．本稿は，リーマン曲率テンソルやラプラシアンなどといった微分幾何において重要な概念を計算するプログラムを紹介し，微分幾何の研究・学習において，Egison が役に立つことを示す．

Abstract

Egison is a programming language that allows users to use tensor index notation for arbitrary user-defined functions without requiring an additional description. Egison also supports a concise description of formulae with differential forms and allows users to concisely define operators for differential forms such as Wedge product, exterior derivative, Hodge operator, and interior product. As a result, users of Egison can describe calculations in differential geometry very concisely. This paper shows that Egison is useful for research and learning of differential geometry by demonstrating programs for calculating important notions in differential geometry such as Riemann curvature tensors and Laplacian.

1 はじめに

コンピューターは，ひとたびアルゴリズムをプログラムとして記述さえすれば，人間よりも圧倒的な速さで自動で計算結果を導く．そのため，コンピューターのユーザにとって，計算にかかる手間の大きさは，アルゴリズムをプログラムとしていかに簡潔に記述できるかに依存する．その結果，現在まで，アルゴリズムの記述を簡潔にするためのさまざまな工夫が研究されてきた．

コンピューターによる計算の対象は，現代数学の計算も含まれている．Wolfram 言語をはじめとする多くの数式処理システムは，この目的のために開発されたプログラミング言語である．数式処理システムは，一般的なプログラミング言語と異なり，数値的な計算だけでなく，シンボリックな計算 (e.g. $x + x = 2x$) もサポートする．これらの数式処理システムは，微分幾何をはじめ，多くの数学の研究に貢献してきた．

微分幾何は紙とペンによる計算がとくに大変な分野である．例えば，曲率が自明ではないもっとも単純な多様体は球面 S^2 であるが，そのリーマン曲率テンソルを手動で計算すると，初めての場合，30 分くら

*satoshi.egi@rakuten.com

いかかる．続けて，その次に単純な多様体であるトーラス T^2 のリーマン曲率テンソルを計算すると，またしても 30 分以上はかかる．実際の研究のためには，より高次元の複雑な多様体の曲率を計算することが多い．そのような場合，一ヶ月以上計算にかかることもある．

コンピューターを使えば，S^2 のリーマン曲率テンソルは，球面座標系，リーマン計量，クリストッフェル記号，リーマン曲率テンソルの定義をプログラムとして記述すれば，数秒で計算できる．T^2 については，S^2 のリーマン曲率テンソルを計算をするプログラムを書いた後であれば，座標系の設定以外の部分は使いまわせるため，数分の作業で計算できる．より高次元の複雑な多様体についても，計量の設定を変更するだけで，計算することができる．

しかし，既存の数式処理システムには，テンソルの添字記法をはじめとする微分幾何の計算を表現するために広く使われている記法をサポートしていないという問題があった．プログラミングを専門としない数学の研究者にとって，プログラミングのために，ふだんノートや黒板に記述しているよりも煩雑な数式の記法を覚えることをは，あまり楽しい作業ではない．そのため，幅広い分野の数学について，簡潔な記法で数式をプログラムとして記述できるようにすることは，より多くの研究がコンピューターの恩恵を受けられるようにするために重要である．

私が開発したプログラミング言語 Egison[1] は，従来の数式処理システムよりも簡潔に微分幾何の計算をプログラムとして記述できる．とくにテンソルの添字記法をプログラム中でそのまま使うことができる．また，外微分やホッジ作用素のような微分形式のための作用素もプログラムで簡潔に定義するための仕組みも提供している．さらに，未定義関数 $f(x, y)$ の引数を省略して，f と記述することをプログラムでも許す機能ももっている．これらの機能は，数学者の紙の上での定義や頭の中の認識に近いプログラミングを可能にする．本稿の目的は，これらの機能を紹介・実演し，微分幾何の研究・教育に Egison が役に立つことを読者に示すことである．

Egison 処理系には，上記のような新規の機能だけではなく，実用のための汎用的な機能も実装されている．たとえば，Egison は LaTeX 形式や Mathematica 形式での計算結果の出力をサポートしている．Egison は Jupyter Notebook とも連携しており，LaTeX 形式の出力をインタラクティブに数式として表示できる．Mathematica 形式の出力は，Egison で書いたプログラムの出力を Mathematica にわたすことを簡単にする．この機能は，たとえば，Mathematica でなければできない数式の簡約や，Egison にまだ実装されていない機能 (たとえば積分など) を使いたい場合に役に立つ．

本稿の構成を説明する．2 節では，本稿を読む上で必要な Egison の基本的な文法と使い方を解説する．3 節では，さまざまな多様体のリーマン曲率を計算できる Egison プログラムを紹介する．4 節では，ウェッジ積，外微分，ホッジ作用素，内部積，リー微分をはじめとする微分形式のための演算子を Egison で定義する．また，それらの演算子を利用して，ホッジ・ラプラシアンの計算，曲率形式の公式によるリーマン曲率の計算，オイラー形式の計算を実演する．5 節では，本稿を振り返り，結論を述べる．

2　微分幾何の計算を簡潔に記述するための Egison の工夫

次節以降で，具体的なプログラムの例を紹介する前に，Egison とその独自機能について紹介する．

2.1　Egison 入門

本節は，本稿を読みはじめるにあたって必要最低限の Egison の基本的な文法と使い方を解説する．

2.1.1 Egison の文法

関数適用は，関数とその引数を “(” と “)” で囲むことにより，表現される．“(”，“)” で囲まれている一つ目の要素に適用する関数を記述する．そして，二つ目以降の要素に，その関数に渡される引数を記述する．たとえば，$\cos\theta$(関数 cos にたいする引数 θ の適用) は以下のように Egison で表現される．

```
(cos θ)
```

四則演算のための関数 “+”，“-”，“*”，“/” も前置記法で適用されることに注意してほしい．たとえば，$r\cos\theta$ は，以下のように記述される．

```
(* r (cos θ))
```

cos や sin のように引数の数が固定されている関数もあれば，“+” や “*” のように任意の数の引数をとれる関数もある．たとえば，以下の例において “*” は 3 つの引数をとっている．

```
(* r (sin θ) (cos φ))
```

ベクトルは，その成分を “[|” と “|]” で囲むことにより，表現される．

```
[| (* r (sin θ) (cos φ)) (* r (sin θ) (sin φ)) (* r (cos θ)) |]
```

行列やテンソルは，ベクトルをネストすることにより表現される．以下は 3 次の単位行列を表現している．

```
[| [| 1 0 0 |] [| 0 1 0 |] [| 0 0 1 |] |]
```

define 式により，変数に値を束縛することができる．たとえば，以下の define は，“X” という変数に，さきほどのベクトルを束縛する．Egison には，これから定義する変数の先頭に “$” を付加する慣習がある．

```
(define $X [| (* r (sin θ) (cos φ)) (* r (sin θ) (sin φ)) (* r (cos θ)) |])
```

2.1.2 Egison の使い方

Egison 処理系をインストールしていれば，egison コマンドで Egison インタプリタを起動できる．“egison -t [ファイル名]” とコマンドラインに入力すれば，ファイル名で指定されたプログラムを実行する．

“-M” オプションで出力形式を指定することができる．“egison -M latex” を実行すると，LaTeX 形式で結果を出力する．“egison -M mathematica” を実行すると，Mathematica 形式で結果を出力する．

```
$ egison -M latex
Egison Version 3.7.14 (C) 2011-2018 Satoshi Egi
https://www.egison.org
Welcome to Egison Interpreter!
> (+ (sqrt a) b)
#latex|\sqrt{a} + b|#
```

```
$ egison -M mathematica
Egison Version 3.7.14 (C) 2011-2018 Satoshi Egi
https://www.egison.org
Welcome to Egison Interpreter!
> (+ (sqrt a) b)
#mathematica|Sqrt[a] + b|#
```

LaTeX 出力は Jupyter Notebook[3] とも連携している．Jupyter Notebook の Egison プラグイン [2] を利用すれば，図 3，図 4 のようにインタラクティブに計算結果が数式として表示される．

```
(define $min (lambda [$x $y] (if (less-than? x y) x y)))
```

(a) `min` 関数の定義

$$min(\begin{pmatrix}1\\2\\3\end{pmatrix}_i, \begin{pmatrix}10\\20\\30\end{pmatrix}_j) = \begin{pmatrix}min(1,10) & min(1,20) & min(1,30)\\ min(2,10) & min(2,20) & min(2,30)\\ min(3,10) & min(3,20) & min(3,30)\end{pmatrix}_{ij} = \begin{pmatrix}1 & 1 & 1\\ 2 & 2 & 2\\ 3 & 3 & 3\end{pmatrix}_{ij}$$

(b) 異なる添字をもつベクトルにたいする `min` 関数の適用

$$min(\begin{pmatrix}1\\2\\3\end{pmatrix}_i, \begin{pmatrix}10\\20\\30\end{pmatrix}_i) = \begin{pmatrix}min(1,10) & min(1,20) & min(1,30)\\ min(2,10) & min(2,20) & min(2,30)\\ min(3,10) & min(3,20) & min(3,30)\end{pmatrix}_{ii} = \begin{pmatrix}min(1,10)\\ min(2,20)\\ min(3,30)\end{pmatrix}_i = \begin{pmatrix}1\\2\\3\end{pmatrix}_i$$

(c) 同じ添字をもつベクトルにたいする `min` 関数の適用

図 1: `min` 関数の定義と適用例

```
(define $. (lambda [%t1 %t2] (contract + (* t1 t2))))
```

(a) “.” 関数の定義

$$\begin{pmatrix}1\\2\\3\end{pmatrix}^i \cdot \begin{pmatrix}10\\20\\30\end{pmatrix}_i = contract(+, \begin{pmatrix}10 & 20 & 30\\ 20 & 40 & 60\\ 30 & 60 & 90\end{pmatrix}^i_i) = 10+40+90 = 140$$

$$\begin{pmatrix}1\\2\\3\end{pmatrix}_i \cdot \begin{pmatrix}10\\20\\30\end{pmatrix}_i = contract(+, \begin{pmatrix}10\\40\\90\end{pmatrix}_i) = \begin{pmatrix}10\\40\\90\end{pmatrix}_i$$

$$\begin{pmatrix}1\\2\\3\end{pmatrix}_i \cdot \begin{pmatrix}10\\20\\30\end{pmatrix}_j = contract(+, \begin{pmatrix}10 & 20 & 30\\ 20 & 40 & 60\\ 30 & 60 & 90\end{pmatrix}_{ij}) = \begin{pmatrix}10 & 20 & 30\\ 20 & 40 & 60\\ 30 & 60 & 90\end{pmatrix}_{ij}$$

(b) “.” 関数の適用

図 2: “.” 関数の定義と適用例

2.2 テンソルの添字記法のプログラミングへの導入

テンソルの添字記法のプログラミングへの導入は，(i) **スカラー仮引数**と**テンソル仮引数**という 2 種類の仮引数の概念の導入と，(ii) 適切な添字の簡約ルールによってなされる．[4]

まず，スカラー仮引数とテンソル仮引数という 2 種類の仮引数の概念を説明する．スカラー仮引数とテンソル仮引数は，**スカラー関数**と**テンソル関数**という 2 種類の関数を定義するためにそれぞれ使われる．

スカラー関数は，テンソルを引数にとったとき，成分ごとに処理をマップする関数のことをいう．たとえば，“+” や，“-”，“*”，“/”，“∂ /∂ ” はスカラー関数である．図 1 は，スカラー関数の例として，`min` 関数を定義とその適用例を紹介している．Egison では、Scheme と同様 `lambda` 式を用いて関数を定義する．“$” が先頭に付加されている仮引数はスカラー仮引数となり，テンソルが引数として渡された場合，自動で成分ごとに関数がマップされる．成分ごとに関数の処理がマップされた結果，図 1b のように関数はテンソル積と同じ形で適用される．テンソルの添字の簡約ルールはシンプルである．図 1c のようにスカラー関数を適用した結果，同じシンボルの添字が 2 つ以上付加されたテンソルが生成された場合，その対角成分からなるテンソルに簡約されるというルールだけである．

テンソル関数は，テンソルを引数にとったとき，テンソルをテンソルとしてそのまま処理する関数のことをいう．たとえば，行列式を計算する関数や，テンソル同士のかけ算のための関数は，テンソルの成分ごとに自動でマップされては記述できないため，テンソル関数として定義する必要がある．図 2 は，テンソル

関数の例として，テンソルのかけ算をする “.” 関数の定義と適用例を紹介している．“%” が先頭に付加されている仮引数はテンソル仮引数となり，テンソルが引数として渡された場合，テンソルとしてそのまま渡される．図 2b のように，Egison は上添字と下添字の両方を組み込みでサポートしている．上下添字で同じシンボルが使われていた場合は，組み込み関数 `contract` を使って縮約することができる．

上添字は “~”，下添字は “_” を使ってプログラム上で表現される．たとえば，図 2b の一つ目の計算例は以下のようにインタプリタ上で実行できる．

```
> (. [| 1 2 3 |]~i [| 10 20 30 |]_i)
140
```

2.3 省略された添字の補完

添字が省略された場合の添字補完のルールを適切に設定すると，微分形式のための演算子を簡潔に定義できる．本節では，添字補完のルールだけを説明する．微分形式の演算子の定義の具体例は，4 節で紹介する．

以下，`A`，`B` を 2 階のテンソル，つまり行列であるとする．添字を省略した状態でスカラー関数に `A`，`B` を同時に適用すると，下記左のように同じ組み合わせの添字が補完される．関数適用の前に “!” が付加されていた場合は，下記右のように違う添字が補完される．

```
(+ A B) ; (+ A_t1_t2 B_t1_t2)
```

```
!(+ A B) ; (+ A_t1_t2 B_t3_t4)
```

あとで 4 節で紹介するように，適切にプログラミングすれば，“!” が使われるのは，微分形式の演算子の定義の中だけになる．そのため，基本的には，ユーザーは上記左のルールだけを意識すればよい．

2.4 関数シンボルによる未定義関数の引数の省略

数学では，シンボリックな関数の引数は，多くの場合，省略される．たとえば，名前が f，引数が x, y と決まっているが，中身は定義されていないシンボリックな関数 $f(x, y)$ は，紙の上では f と引数を省略して表記されることが多い．Egison はこの記法を**関数シンボル**としてプログラミングに導入した．本節はこの記法の使い方について説明する．

2.4.1 関数シンボルの微分

関数シンボルが特に力を発揮するのは，微分を扱う計算のときである．以下では `x`，`y` を引数にとる関数シンボル `f` を定義している．そして，その関数シンボル `f` を `x` や `y` について微分している．`f|x` は，関数 `f` を `x` で偏微分した結果の関数を表す．`f|x|y` は，関数 `f` を `x` で偏微分した後，さらに `y` で偏微分した結果の関数を表す．“|” は，“_” や “~” と同様 Egison 組み込みの記号である．以下に関数シンボルの微分を標準的な Egison 出力と LaTeX 形式の出力の両方の表示例を示す．∂ /∂は Egison で定義されているライブラリ関数である．

```
> (define $f (function [x y]))
> f
f
> (∂/∂ f x)
f|x
> (∂/∂ (∂/∂ f x) y)
f|x|y
```

```
> (define $f (function [x y]))
> f
#latex|f|#
> (∂/∂ f x)
#latex|\frac{\partial f}{\partial x}|#
> (∂/∂ (∂/∂ f x) y)
#latex|\frac{\partial^2 f}{\partial x \partial y}|#
```

f は x と y についての関数であるので，以下のように z で微分すると 0 になる．

```
> (∂/∂ f z)
0
```

物理の計算では，t, x, y, z の 4 つの引数をとる関数がよく登場する．もし関数シンボルのような仕組みがないと，式が長くなり，読み難くなるため，関数シンボルの仕組みはプログラムの読み書きしやすくする．

合成関数の微分にも対応していることが，Egison の関数シンボルの重要な特徴である．たとえば，関数シンボルは，以下のように合成関数の微分にも対応できる．function 式の仮引数が変数名 ($x や$y) ではなく式をとるのは，以下のように合成関数の微分に対応するためである．

```
> (define $f (function [x y]))
> (define $x (* r (cos θ)))
> (define $y (* r (sin θ)))
> (∂/∂ f r)
(+ (* f|x (cos θ)) (* f|y (sin θ)))
```

2.4.2 関数シンボルを成分にもつテンソル

テンソルの成分に関数シンボルがあらわれた場合もサポートしていることも，Egison の関数シンボルの重要な特徴である．以下の例のようにテンソルの成分が関数シンボルであった場合，テンソルを束縛している変数名に成分の位置の添字を付加した名前をもつ関数シンボルが生成される．この機能はプログラム上でベクトル場やテンソル場を表現するときに便利である．

```
> (define $g_i_j
    (generate-tensor
      (match-lambda [integer integer]
        {[[$n ,n] (function [x y z])]
         [[_ _] 0]})
      {3 3}))
> g_i_j
[| [| g_1_1 0 0 |] [| 0 g_2_2 0 |] [| 0 0 g_3_3 |] |]_i_j
> (∂/∂ g_i_j x)
[| [| g_1_1|x 0 0 |] [| 0 g_2_2|x 0 |] [| 0 0 g_3_3|x |] |]_i_j
```

generate-tensor 式は，テンソルを生成するのに便利な組み込み構文である．第一引数に成分の位置を引数にとり，その位置の成分の値を返す関数をとる．第二引数にテンソルのサイズをとる．返り値として，第二引数のサイズで，それぞれの成分が第一引数の関数により計算されたテンソルを返す．

3 リーマン曲率テンソルの計算

本節は，球面について，リーマン計量や，クリストッフェル記号，リーマン曲率テンソルといった様々な微分幾何的な量を Egison を使って計算するプログラムを解説する．図 3 は，Jupyter Notebook 上で球面のリーマン曲率テンソルを計算する Egison プログラムとその実行結果の出力である．さまざまな幾何学的不変量は，曲率から計算される．本節で解説するプログラムの計量の部分だけを変えれば，あらゆる多様体の曲率を計算できる．そのため，本節のプログラムを理解すれば，実際の微分幾何の研究にも Egison が使えるはずである．本節は，このプログラムの各箇所について順番に説明していくことによって進める．

クリストッフェル記号やリーマン曲率テンソルについては，[5] の説明がわかりやすい．

```
In [1]: (define $x [|θ φ|])

In [2]: (define $X [|(* r (sin θ) (cos φ))
                    (* r (sin θ) (sin φ))
                    (* r (cos θ))|])

In [3]: (define $e_i_j (∂/∂ X_j x-i))

In [4]: (define $g__ (generate-tensor (lambda [$i $j] (V.* e_i_# e_j_#)) {2 2}))

In [5]: g_#_#
```

$$\begin{pmatrix} r^2 & 0 \\ 0 & r^2 sin(\theta)^2 \end{pmatrix}_{\#\#}$$

```
In [6]: (define $g~~ (M.inverse g_#_#))

In [7]: (define $Γ_j_k_l
          (* (/ 1 2)
             (+ (∂/∂ g_j_l x-k)
                (∂/∂ g_j_k x-l)
                (* -1 (∂/∂ g_k_l x-j)))))

In [8]: (define $Γ~__ (with-symbols {i} (. g~#~i Γ_i_#_#)))

In [9]: Γ~1_#_#
```

$$\begin{pmatrix} 0 & 0 \\ 0 & -sin(\theta)cos(\theta) \end{pmatrix}_{\#\#}$$

```
In [10]: Γ~2_#_#
```

$$\begin{pmatrix} 0 & \frac{cos(\theta)}{sin(\theta)} \\ \frac{cos(\theta)}{sin(\theta)} & 0 \end{pmatrix}_{\#\#}$$

```
In [11]: (define $R~i_j_k_l
           (with-symbols {m}
             (+ (- (∂/∂ Γ~i_j_l x~k) (∂/∂ Γ~i_j_k x~l))
                (- (. Γ~m_j_l Γ~i_m_k) (. Γ~m_j_k Γ~i_m_l)))))

In [12]: R~1_2_#_#
```

$$\begin{pmatrix} 0 & sin(\theta)^2 \\ -sin(\theta)^2 & 0 \end{pmatrix}_{\#\#}$$

```
In [13]: R~2_1_#_#
```

$$\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}_{\#\#}$$

図 3: 球面のリーマン曲率テンソルの計算

3.1 球座標系

球座標系は，r, θ, ϕ の 3 つのパラメーターからなる．r は原点からの距離をあらわす．θ は z 軸との角度，緯度をあらわす．ϕ は xy 平面における x 軸からの回転量．経度をあらわす．球座標系の r の値を固定すれば，球面の座標系を得ることができる．

球座標系とデカルト座標系は，$x = r\cos\theta\sin\phi, y = r\cos\theta\cos\phi, z = r\sin\theta$ という関係式で結ばれている．

以上が，図 3 の [1] と [2] で表現されている内容である．

3.2 リーマン計量

リーマン計量は，各点における基底ベクトルの大きさや向きを表現する行列である．球座標系においては，基底ベクトルの大きさが，各点で異なる．たとえば，北極や南極に近い場所では，経度方向の基底ベクトルの大きさが，赤道付近にくらべて小さくなる．

図 3 の [3] で基底ベクトルが定義されている．[4] ではベクトルの内積を計算する関数 V.*を使って基底ベクトルの大きさが計算されている．

[5] では，[4] で定義したリーマン計量の値を表示している．“#” はダミーシンボルと呼ばれるシンボルとして使えるオブジェクトである．すべての “#” はそれぞれ違うシンボルとして扱われる．

[6] では，逆行列を計算する関数 M.inverse を使って，上添字がつくリーマン計量を求めている．リーマン計量には下添字がつく g_{ij} と上添字がつく g^{ij} の 2 種類がある．この同じ名前の 2 種類のテンソルを，同じ名前の変数 g に束縛するために，Egison は変数定義のときに，変数名だけでなくテンソルの添字の型も指定できるようになっている．“$g__” や “$g~~” のようにテンソルの添字の型は，変数名に続いて宣言される．

[3] では，$e_i_j というように添字の型だけでなく，添字に使われるシンボルの名前まで変数定義に記述されている．実は，これは下記のように Egison 内部で変換されて解釈される糖衣構文である．

```
(define $e_i_j ...) ;=> (define $e__ (with-symbols {i j} (transpose {i j} ...)))
```

with-symbols 式は，局所シンボルを生成する構文である．第一引数のリストで指定されたシンボル (上記の場合，i と j) を第二引数の式の中でシンボルとして使うことができる．with-symbols 式のこの機能のおかげで，もしプログラムの別の箇所で，i や j にたとえば具体的な整数が束縛されていたとしても，with-symbols 式の内部では，その影響を考えなくてよくなる．transpose 関数は，テンソルの成分を第一引数で指定された順番に転置する組み込み関数である．

3.3 クリストッフェル記号とリーマン曲率テンソル

[7] と [8] は，第一種クリストッフェル記号と第二種クリストッフェル記号の数式による定義をプログラムとして記述したものである．数式に非常に近いかたちでプログラムとして表現されている．直感的には，クリストッフェル記号 ${\Gamma^i}_{jk}$ は，基底ベクトル e_j を e_k 方向に動かしたときの e_i 方向への変化の具合を表現する．

$$\Gamma_{ijk} = \frac{1}{2}(\frac{\partial g_{ij}}{\partial x^k} + \frac{\partial g_{ik}}{\partial x^j} - \frac{\partial g_{kj}}{\partial x^i}) \qquad {\Gamma^i}_{jk} = g^{im}\Gamma_{mjk}$$

リーマン曲率テンソルも，[11] のように，その数式による定義をそのままプログラムとして記述できている．直感的には，リーマン曲率テンソル ${R^i}_{jkl}$ は，e_k と e_l によって作られる閉路に沿って e_j を動かしたときの e_i 方向への変化の具合を表現する．

$${R^i}_{jkl} = \frac{\partial {\Gamma^i}_{jl}}{\partial x^k} - \frac{\partial {\Gamma^i}_{jk}}{\partial x^l} + {\Gamma^m}_{jl}{\Gamma^i}_{mk} - {\Gamma^m}_{jk}{\Gamma^i}_{ml}$$

3.4 計量を変えてみる

本節で解説したプログラムの計量の部分を変えるだけで，いろいろな多様体のクリストッフェル記号やリーマン曲率テンソルを計算できる．

たとえば，トーラス T^2 のリーマン曲率テンソルを計算するには，球座標系をトーラスの座標系に変更すればよい．トーラスの座標系はたとえば，以下のように定義できる．

```
(define $x [| θ  φ |])

(define $X [|(* '(+ (* a (cos θ)) b) (cos φ))
             (* '(+ (* a (cos θ)) b) (sin φ))
             (* a (sin θ))|])
```

シングルクォート (“’”) が，上記の式の中で使われている．シングルクォートは Egison 処理系に展開したくない括弧を指定する役目を持っている．Egison 処理系は，数式を自動で積和標準形に展開する．(現状の Egison は数式の簡約については，シンプルな実装しかできていない．) シングルクォートのついた数式は，Egison 処理系に一つのシンボルと同じように扱われる．シングルクォートを適切に挿入することより，計算プロセスや，最終的な計算結果が単純になることがある．

また，計量を以下のように定義される Schwarzschild 計量に変えれば，原点を中心に一つだけ質量をもつ天体がある宇宙についての一般相対性理論の計算ができる．この計算結果を解釈すれば,「時空の曲がり=重力」ということが理解できる．[7]

```
(define $g__ [|[| (/ '(- (* c^2 r) (* 2 G M)) (* c^2 r)) 0 0 0 |]
              [| 0 (/ -1 (/ '(- (* c^2 r) (* 2 G M)) (* c^2 r))) 0 0 |]
              [| 0 0 (* -1 r^2) 0 |]
              [| 0 0 0 (* -1 r^2 (sin θ)^2) |]|])
```

4　微分形式を使った計算

2.3 節で言及したように，Egison は微分形式を使った計算を簡潔に表現する機能を持っている．本節では，この機能を使って微分形式を使った計算をプログラムした例を紹介する．図 4 は，極座標のラプラシアンをホッジ・ラプラシアンの公式を使って計算する Egison プログラムとその実行結果である．本節の前半は，このプログラムの解説をしながら進む．微分形式の理論については，[5, 6] の説明がわかりやすい．

4.1　外微分・ホッジ作用素

図 4 は前半 `[1]-[4]` は，極座標系のパラメーターと計量を設定している．その後，外微分・ホッジ作用素・外微分の随伴作用素という微分形式の基本的な演算子を `[5]-[8]` で定義している．`[6]` のホッジ作用素の定義には以下の公式を使っている．この定義のなかで使われている `subrefs` は，第二引数のシンボルのリストを，第一引数のテンソルの下添字として順番に付加する組み込み関数である．

$$*A = \sqrt{\det |g|} \cdot \epsilon_{i_1 \ldots i_n} \cdot A_{j_1 \ldots j_k} \cdot g^{i_1 j_1} \cdots g^{i_k j_k} \cdot e^{i_{k+1}} \wedge \ldots \wedge e^{i_n}$$

4.2　ホッジ・ラプラシアン

微分形式を使うと，ラプラシアンは $\Delta = d\delta + \delta d$ と座標系に依存しないように定義できる．この公式を使って表現されるラプラシアンはホッジ・ラプラシアンと呼ばれる．`[8]` においてこの公式が Egison で表現されている．`[8]` の定義において，0 形式と 2 形式の場合が特別扱いされている．これは，2 次元座標系における 2 形式にたいして d が，0 形式にたいして δ が，厳密には未定義であるためである．

`[9]` で定義した関数シンボルに，`[10]` でホッジ・ラプラシアンを適用している．3 節で解説したリーマン曲率テンソルと同様，ホッジ・ラプラシアンについても計量を変えるだけで，いろいろな座標系のラプラシアンを計算できる．

```
In [1]: (define $N 2)

In [2]: (define $x [|r θ|])

In [3]: (define $g__ [|[|1 0|] [|0 r^2|]|])

In [4]: (define $g~~ (M.inverse g_#_#))

In [5]: (define $d
          (lambda [%X]
            !((flip ∂/∂) x X)))

In [6]: (define $hodge
          (lambda [%A]
            (let {[$k (df-order A)]}
              (with-symbols {i j}
                (* (sqrt (abs (M.det g_#_#)))
                   (foldl . (. (subrefs A (map 1#j_%1 (between 1 k)))
                               (subrefs (ε' N k) (map 1#i_%1 (between 1 N))))
                          (map 1#g~[i_%1]~[j_%1] (between 1 k)))))))))

In [7]: (define $δ
          (lambda [%A]
            (let {[$r (df-order A)]}
              (* (** -1 (+ (* N r) 1))
                 (hodge (d (hodge A)))))))

In [8]: (define $Δ
          (lambda [%A]
            (match (df-order A) integer
              {[,0 (δ (d A))]
               [,N (d (δ A))]
               [_ (+ (d (δ A)) (δ (d A)))]})))

In [9]: (define $f (function [r θ]))

In [10]: (Δ f)
```

$$\frac{-\frac{\partial^2 f}{\partial \theta^2} - r\frac{\partial f}{\partial r} - r^2\frac{\partial^2 f}{\partial r^2}}{r^2}$$

図 4: 極座標のホッジ・ラプラシアンの計算

4.3 ウェッジ積・曲率形式

曲率形式の公式 $\Omega^i{}_j = d\omega^i{}_j + \omega^i{}_k \wedge \omega^k{}_j$ は，微分形式を使った座標系に依存しないリーマン曲率テンソルの公式である．微分形式による第二種クリストフェル記号の表現は，接続形式と呼ばれ，上記の公式に ω としてあらわれている．曲率形式の公式も Egison で簡潔に表現できる．以下のプログラム中に，ウェッジ積も定義されている．

```
(define $ω Γ~#_#_#)
(define $wedge (lambda [%X %Y] !(. X Y)))
(define $Ω (with-symbols {i j} (antisymmetrize (+ (d ω~i_j) (wedge ω~i_k ω~k_j)))))
```

4.4 オイラー形式とオイラー数

オイラー類は，積分したらオイラー数が得られる特性類として有名な特性類である．以下のプログラムは，トーラス T^2 のオイラー形式を計算している．

```
(define $euler-form (* (/ 1 (* 2 π)) (- Ω~1_2 Ω~2_1)))

euler-form ; [| [| 0 (/ (cos θ) (* 2 π)) |] [| (/ (* -1 (cos θ)) (* 2 π)) 0 |] |]
```

積分のための関数が Egison には実装されていないため，手計算でこれを積分した結果を記す．積分すると T^2 のオイラー数である 0 が得られる．

$$\chi(T^2) = \int d\theta d\phi \frac{\cos\theta}{2\pi} = \int d\theta(\cos\theta) = [(\sin\theta)]_0^\pi = (\sin\pi) - (\sin\theta) = 0$$

4.5 内部積・リー微分

内部積やリー微分も，Egison で簡潔に定義できることを紹介しておく．これらの演算子は，たとえば，流体力学の質量保存の法則やナビエ・ストークス方程式の微分形式による表現をするときに使う．

```
(define $ι
  (lambda [%X %Y]
    (with-symbols {i} (* (df-order Y) (. X...~i (antisymmetrize Y..._i))))))

(define $Lie
  (lambda [%X %Y]
    (match (df-order Y) integer
      {[,0 (ι X (d Y))] [,N (d (ι X Y))] [_ (+ (ι X (d Y)) (d (ι X Y)))]})))
```

5 まとめ

本稿は，手動で計算すると多くの手間がかかる微分幾何のテンソル計算を，Egison を使えば数式に近い表記でプログラムできることを紹介した．本稿で紹介したプログラムの一部を編集して使うだけでも，実際の微分幾何の研究に役に立つ．実際，著者は微分幾何学の研究者と共同研究をすでに実施しており，4 次元のあるシンプレクティック多様体のサークルバンドルの Wodzicki-Chern-Simons 形式を Egison で計算することにより，数学者の手計算のミスを修正したり，検証することに貢献した．この共同研究については，現在論文を準備中である．今後も，さらにこのような共同研究を展開していきたいと考えているので，微分幾何のプログラムによる計算に興味ある方はぜひお声がけいただきたい．

また，微分幾何の研究だけでなく，微分幾何の教育にも Egison は役に立つ．コンピューターを使って計算することによって，微分幾何の計算は圧倒的に楽にはなるが，それが理解の浅さに繋がるとは限らない．むしろ，微分幾何の複雑な計算をコンピューターにも理解できる形でプログラムとして正しく記述することにより，より深い理解を得られる場合もある．たとえば，微分形式のための演算子を学習者自身にプログラムとして定義させることにより，これらの演算子について新しい視点から深い理解を得ることができる．また，計算が楽になることにより，同じ長さの学習時間で学習者は従来よりも多くの例に触れることができる．本稿を発展させて『Egison で計算しながら学ぶ微分幾何』のような書籍を書きたいと計画している．

参 考 文 献

[1] The Egison Programming Language. `https://www.egison.org`, 2011.

[2] Egison kernel for Jupyter. `https://github.com/egison/egison_kernel`, 2018.

[3] Project Jupyter. `https://jupyter.org`, 2019.

[4] Satoshi Egi. Scalar and Tensor Parameters for Importing Tensor Index Notation including Einstein Summation Notation. *The Scheme and Functional Programming Workshop*, 2017.

[5] 小林昭七. 曲線と曲面の微分幾何（改訂版）. 裳華房, 1995.

[6] 森田茂之. 微分形式の幾何学. 岩波書店, 2005.

[7] 藤井保憲. 時空と重力 *(*物理学の廻廊*)*. 産業図書, 1979.