社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2007－ARC－172 (24)
2007－HPC－109 (24)
2007／3／1

Taylor級数のTemplateライブラリの開発

舘野裕文　平山弘　浅野直之

神奈川工科大学工学部

Taylor展開は数学的には基本的で重要なものであるが数値解析にはあまり利用されていない。Taylor級数の係数はdouble型だけでなくいろいろな型ものが必要である。本研究ではTaylor級数の係数がいろいろな型に対応させるために、Taylor級数ライブラリをtemplateプログラムライブラリにした。倍精度実数用には、誤差関数などいろいろな数学関数を準備した。他の型には標準的な関数を準備した。これを利用することによって、倍精度実数、分数、複素数、区間数を利用することができた。Taylor級数の係数をTaylor展開すれば2次元3次元のTaylor展開の可能である。

Developpement of Template Library for Taylor Series

Hirofumi Tateno, Hiroshi Hirayama and Naoyuki Asano

Kanagawa Institute of Technology

Though the Taylor series is mathematically basic and important, it has not been used enough in numerical analysis. We need not only the double precision type coefficients but also various type ones for the Taylor series. In this paper, the template program library for Taylor series library was made to use for the Taylor series with various type of coefficients. Various mathematics functions like the error function etc. were prepared for double precision real number Taylor series. The standard function was prepared for other types. The double precision real number, the fraction, the complex number and the interval number can be used by using this. If the coefficient of the Taylor series is expanded in Taylor series, the two and three dimensions functions can be expanded in the Taylor series.

1. はじめに

Taylor 展開を使った数値計算法は、次のような特徴を持っている。特に常微分方程式の解法には強力な計算手法を与える。

（1）プログラムされた任意の関数を Taylor 展開[2-4]できる。
（2）常微分方程式の解を任意の次数の Taylor 展開[2]できる。
（3）逆関数の Taylor 展開が可能である。
（4）通常の数値計算と同じように高速に計算できる。

このような特徴を持つ Taylor 展開法を容易に使えるようにするため、template ライブラリを作成した。このライブラリを使用することによって係数が double、float だけでなく複素数や高精度計算が可能である。

本文では、これらの計算法、そのプログラムの特徴、使用方法等を説明する。

2. Taylor 展開

2.1.. Taylor 級数の表現

Taylor 級数はプログラムの中では、単なる係数を値とする配列を含む構造体（C++言語で

はclass文）で表現する。

このクラスの中には、Taylor 級数の係数だけでなく、展開位置などが入っている。展開位置を省略すると、Taylor 級数間の計算を行う時、展開位置が一致することを確かめる必要がなくなり高速化でき、定数は0次の Taylor 展開式として扱うことができ、プログラムが単純になるなどの利点がある。その反面、逆関数を計算するとき、展開位置の扱いが面倒になることになることもある。

Taylor 級数の演算は、平行移動によって展開点を任意の位置へ移すことができるので、一般性を失うことなしに、扱うことができる。展開点が同じ級数を次のように定義[2-3]する。

$$f(x) = f_0 + f_1 x + f_2 x^2 + f_3 x^3 + \cdots \quad (2.1)$$

$$g(x) = g_0 + g_1 x + g_2 x^2 + g_3 x^3 + \cdots \quad (2.2)$$

$$h(x) = h_0 + h_1 x + h_2 x^2 + h_3 x^3 + \cdots \quad (2.3)$$

2.2. 四則演算

$h(x)$ が $f(x)$ と $g(x)$ の和差積商のとき、f, g および h の係数は、それぞれ次のような関係になる。

和差：$h(x) = f(x) \pm g(x) \quad h_i = f_i \pm g_i$

積　：$h(x) = f(x)g(x) \quad h_n = \sum_{k=0}^{n} f_k g_{n-k}$

商：$h(x) = \dfrac{f(x)}{g(x)} \quad h_0 = \dfrac{f_0}{g_0}$、

$$h_n = \frac{1}{g_0}\left(f_n - \sum_{k=0}^{n-1} h_k g_{n-k}\right) \quad (n \geq 1)$$

2.3. 微積分演算

$h(x)$ が $f(x)$ の微積分であるとき、f および h の係数は、それぞれ次の関係になる。

微分：$h(x) = \dfrac{df(x)}{dx} \quad h_i = (i+1) f_{i+1}$

積分：$h(x) = \int_0^x f(x) \quad h_i = \dfrac{1}{i} f_{i-1}$

2.4. 指数関数

指数関数は

$$h(x) = e^{f(x)}$$

とおくと、微分方程式

$$\frac{dh}{dx} = h\frac{df}{dx}$$

を満たす。式（2.1）と式(2.3)を代入し、両辺の係数を比較することによって、次の関係が得られる。

$$h_0 = e^{f_0}, \quad h_i = \frac{1}{i}\sum_{j=1}^{i} j f_{i-j} f_j$$

Taylor 級数の指数関数の計算には、指数関数の計算は、1回しか入らないので、通常の四則演算とそれほど大きな違いにはならない。この方法と同様にして、対数関数、三角関数、べき乗なども計算できる。

2.5. 三角関数

三角関数の

$$g(x) = \sin f(x), \quad h(x) = \cos f(x)$$

は、次の微分方程式を満たす

$$\frac{dg(x)}{dx} = h(x)\frac{df(x)}{dx}, \quad \frac{dh(x)}{dx} = -g(x)\frac{df(x)}{dx}$$

この式から、係数に対する次のような関係式が得られる。

$$g_0 = \sin f_0, \qquad h_0 = \cos f_0$$

$$g_i = \frac{1}{i}\sum_{j=1}^{i} j f_j h_{i-j}, \quad h_i = -\frac{1}{i}\sum_{j=1}^{i} j f_j g_{i-j}$$

三角関数は、$\sin x$ と $\cos x$ を同時に計算すると、計算式が単純で見易い公式となる。これは $\sinh x$ や $\cosh x$ の場合も同様である。

3. 倍精度 Taylor 級数用関数

よく使われる係数が倍精度実数（double）である Taylor 級数用として、新しい C 言語で追加されている関数を準備した。さらに、Taylor 級数の逆関数など Taylor 級数用の関数を準備した。

3.1. 標準関数

倍精度用の関数として、これまで説明した関

数のほかに、C 言語、C++言語や Fortran などで標準で準備されている誤差関数やガンマ関数を準備した。

3.1.1. 誤差関数

誤差関数は

$$erf(x)=\frac{2}{\sqrt{\pi}}\int_0^x e^{-x^2}dx \qquad (3.1)$$

この関数は、微分方程式

$$\frac{d^2y}{dx^2}+2x\frac{dy}{dx}=0 \qquad (3.2)$$

を満たす。誤差関数 $erf(x)$ の微分 $\frac{2}{\sqrt{\pi}}e^{-x^2}$ は、容易に Taylor 展開できる。それを積分することによって、誤差関数 $erf(x)$ の Taylor 展開式が得られる。このときの積分定数は、プログラミング言語で定義されている誤差関数 $erf(x)$ で計算できる。

例えば $x=0.5$ のとき $\frac{2}{\sqrt{\pi}}e^{-x^2}$ を Taylor 展開すると、

$$\frac{2}{\sqrt{\pi}}e^{-x^2}=0.8789-0.8788(x-0.5)$$
$$-0.4394(x-0.5)^2+0.7323(x-0.5)^3$$
$$+0.03662(x-0.5)^4+\cdots$$

となる。これを積分して、$erf(0.5)$ を積分定数とすると

$$erf(x)=0.50205+0.8788(x-0.5)$$
$$-0.4394(x-0.5)^2-0.1465(x-0.5)^3$$
$$+0.1831(x-0.5)^4+\cdots$$

が得られる。

微分方程式(3.2)を Picard の逐次近似法[6]を使い Taylor 展開式を求めることもできる。

Taylor 展開プログラムには含まれていないが、誤差関数と同様な方法で、指数積分関数、Fresnel 積分関数などが同様な方法で計算できる。

3.1.2. ガンマ関数

ガンマ関数は、

$$\Gamma(x)=\int_0^\infty t^{x-1}e^{-t}dt \qquad (3.3)$$

と定義される関数である。これを x が十分大きい時成り立つ Stirling の公式を利用して計算する。この公式は漸近展開式で

$$\log\Gamma(x)\approx\left(x-\frac{1}{2}\right)\log x-x+\frac{1}{2}\log(2\pi)+\sum_{m=1}^{\infty}\frac{B_{2m}}{2m(2m-1)x^{2m-1}} \qquad (3.4)$$

である。ここで、B_{2m} は Bernoulli 数である。この式で次の項を加えても値が変わらなくなるまで計算した。

x が小さな数値のとき、適当な正の整数 N を選んで、$N+x$ が十分大きな数値なるようにして、次の式を計算する。

$$\Gamma(x)=\frac{\Gamma(N+x)}{\prod_{k=0}^{N-1}(k+x)} \qquad (3.5)$$

ここで作成した倍精度の Taylor 展開用のプログラムでは $N+x>20$ となるように N を選んで計算した。

たとえば、$x=1.5$ における Taylor 展開式を計算すると

$$\Gamma(x)=0.8862+0.03234\,(x-1.5)$$
$$+0.4148\,(x-1.5)^2-0.1073\,(x-1.5)^3$$
$$+0.1446\,(x-1.5)^4+\cdots$$

となる。この式では4次までしか表示してないが20時程度までは容易に計算できる。係数は4桁しか表示していないが実際の計算では倍精度15桁程度で計算している。

3.2. Taylor 級数のための関数

Taylor 級数を扱うために必要な関数を準備した。

3.2.1. 二乗関数（square(x)）

2乗する関数は通常のプログラム言語にはないが、Taylor 展開式の計算には非常に便利な関数である。これを使うことによって、2乗計算を乗算より2倍程度高速化される。

3.2.2. 逆関数（inv_func（x,a））

逆関数の計算は、方程式の零点を求めるのに便利であり、任意次数の逆関数が計算できるので、これを利用すれば、任意の次数の零点を

求める公式が得られる。

逆関数の計算は、$y = f(x)$ の逆関数が $x = f(y)$ となることを利用し、この両辺を微分して、

$$\frac{dy}{dx} = \frac{1}{f'(x)} = v(x) \qquad (3.6)$$

という逆関数の微分方程式を得る。初期条件 $y(x_0) = y_0$ と Picard の逐次近似法[5]を使い、

$$y_n = y_0 + \int_{x_0}^{x} v(y_{n-1})dx \qquad (3.7)$$

が得られる。この公式を n 回繰り返し行い、逆関数を n 次まで求める。

展開点 x_0 で n 次までの Taylor 級数

$$f(x) = f_0 + f_1(x-x_0) + f_2(x-x_0)^2 + \cdots + f_n(x-x_0)^n \qquad (3.8)$$

が与えられたとき、逆関数 $f^{-1}(x)$ の Taylor 級数

$$f^{-1}(x) = x_0 + x_1(x-f_0) + x_2(x-f_0)^2 + \cdots + x_n(x-f_0)^n \qquad (3.9)$$

計算する。

逆関数を求める関数 inv_func は、式(3.8)を与えて、(3.9)を求める関数である。

数値例として、次の関数を考える。

$$f(x) = e^x - x^2 + \log x - 3$$

この関数を $x = 2$ で Taylor 展開すると

1.0822+3.88906*(x-2)+2.56953*(x-2)^2
+1.27318*(x-2)^3+0.292252*(x-2)^4

となる。これの逆関数の Taylor 展開式を計算すると

2+0.166716*(x-1.0822)+0.0283614*
(x-1.0822)^2+0.0056632*(x-1.0822)^3
+0.00112957*(x-1.0822)^4

得られる。

3.2.3. Padé 展開(pade(f,p,n,q,m))

Taylor 展開は、容易に Padé 展開することができる。Padé 展開とは、以下のように Taylor 展開式を、有理関数に変形したものである。

$$a_0 + a_1x + a_2x^2 + \cdots = \frac{p_0 + p_1x + \cdots + p_Mx^M}{1 + q_1x + \cdots + q_Lx^L}$$

この両辺に右辺の分母を掛け、$M + L$ 次の係数まで一致するように、有理関数の係数を決定することによって得られる。

Padé 展開は、一般に、同じ次数の Taylor 展開式より高精度で、収束の良い式を与えることが多いので、常微分方程式の解の Taylor 展開式を Padé 展開することは、A 安定な常微分方程式の計算法を与える。

分子 M 次、分母 L 次式に Padé 展開したとき、$M \le L \le M+2$ のとき、この Padé 展開式で次のステップを計算した場合 A 安定である[1]ことが知られている。

3.2.4. 微分方程式の解の Taylor 展開

次のような微分方程式を考える。

初期条件 $y(x_0) = y_0$ としたとき

$$\frac{dy}{dx} = f(x,y) \qquad (3.10)$$

を解く。これを n 次までの Taylor 級数解[2][4]を得る関数を準備した。

void picard(void (*f)(x, y, dy), x0, y0, y, n) ;

これを呼び出すと初期条件 $y(x_0) = y_0$ を満たす $x = x_0$ における、n 次までの Taylor 級数解が Taylor 級数変数 y に得られる。

4. Taylor 級数 Template プログラム

計算は倍精度で計算されることが多いが、複素数や高精度で計算しなければならないことが起こる。このような目的のために、次のような template を作成した。

Taylor 級数

$$f(x) = f_0 + f_1(x-a) + f_2(x-a)^2 + f_3(x-a)^3 + \cdots$$

を次のように表現する。これを

```
template<typename T>
class taylor_template
{
        T     position ; // 展開位置
        T     coeffients[32] ; // 展開係数
} ;
```

とした。このようにする定義すると、加減算や関数計算など倍精度用に定義した関数の多くの関数がそのまま使うことができる。上の説明

では出てこないが、精度に依存した部分は変更が必要である。また、引数として使う型がtemplate クラスで定義された型の場合、計算に曖昧さが生じることがある。このような場合補助の template クラスを使うなどして曖昧さを無くす必要がある。

4.1. 倍精度計算

$$\frac{dy}{dx} = -\frac{1}{3}x^2y^2 \quad \text{初期条件} \quad y(2)=1 \quad (4.1)$$

この方程式を倍精度で解くためのプログラムは次のようになる。

```
#include "taylor_template.h"
typedef taylor_template<double> taylor ;
void func( const taylor& x, const taylor& y,
      taylor& dy )
{
      dy = -1.0/3.0*x*x*y*y ;
}
void main()
{
      taylor   y, p, q ;
      double    x0, y0 ;
      int       n ;
      x0 = 2 ; y0=1 ; n=10 ;
      picard( &func, x0, y0, y, n ) ;
      cout << y << endl ;
}
```

このプログラムの実行結果は

```
1-1.33333*(x-2)+1.11111*(x-2)^2
-0.703704*(x-2)^3+0.345679*(x-2)^4
-0.115226*(x-2)^5+0.00137174*(x-2)^6
+0.0365798*(x-2)^7-0.0368846*(x-2)^8
+0.0246406*(x-2)^9-0.0123287*(x-2)^10
```

となる。倍精度で計算することが多いが、これらの計算をチェックするために、分数や高精度で計算することもできる。

4.2. 分数計算

有理数のクラスを利用すれば、Taylor 展開の係数を厳密に計算できる場合がある。4.1 の例題は厳密に計算できる問題である。以下のプログラムの long_num.h は分数計算を定義しているファイルで、long_rational が分数を表すクラス名である。

このプログラムは、

```
#include "taylor_template.h"
#include "long_num.h"
Typedef taylor_template<long_rational>
    dtaylor ;
void func( const dtaylor& x, const dtaylor& y,
      dtaylor& dy )
{
      dy = dtaylor(-1)/dtaylor(3)*x*x*y*y ;
}
void main()
{
      dtaylor   y, p, q ;
      long_rational x0, y0 ;
      int       n ;
      x0 = 2 ; y0=1 ; n=10 ;
      picard( &func, x0, y0, y, n ) ;
      cout << y << endl ;
}
```

この場合の計算結果は、

```
1-4/3*(x-2)+10/9*(x-2)^2-19/27*(x-2)^3
+28/81*(x-2)^4-28/243*(x-2)^5
+1/729*(x-2)^6+80/2187*(x-2)^7
-242/6561*(x-2)^8+485/19683*(x-2)^9
-728/59049*(x-2)^10
```

となる。このような計算をすれば、厳密な計算ができるが、計算が遅くなり、分数計算のまま次のステップが計算できないなどの問題がある。

定数の型は宣言ができないため、例題のようにコンストラクターを使って定数の型を指定しなければならないことがある。

同様に高精度計算も可能で

```
1.0-1.3333333333333333333333333333*(x2.0)
+1.1111111111111111111111111111*(x-2.0)^2
-7.0370370370370370370370e-01*(x-2.0)^3
+3.4567901234567901234567e-01*(x-2.0)^4
-1.1522633744855967078189e-01*(x-2.0)^5
+1.3717421124828532235939e-03*(x-2.0)^6
+3.6579789666209419295839e-02*(x-2.0)^7
-3.6884621246761164456638e-02*(x-2.0)^8
+2.4640552761266067164558e-02*(x-2.0)^9
-1.2328743924537248725 63e-02*(x-2.0)^10
```

が得られる。

4.3. 複素数計算

複素数を初期条件とした次の問題を考える。

$$\frac{dy}{dx} = -\frac{1}{3}x^2y^2 \quad \text{初期条件} \quad y(2+i)=1+i$$

(4.2)

上の計算を実行するプログラムを C++言語に含まれる複素数を利用書くと

```
#include "taylor_simple.h"
#include <complex>
typedef complex<double> dcomplex ;
bool operator==(const dcomplex x, const dcomplex y )
{
    return real(x)==real(y) && imag(x)==imag(y) ;
}
typedef taylor_tmpl<dcomplex> taylor ;
void func( const taylor& x, const taylor& y, taylor& dy )
{
    dy = -dcomplex(1)/dcomplex(3)*x*x*y*y ;
}
void main()
{
    taylor  y, p, q ;
    dcomplex  x0, y0 ;
    int  n ;
    x0 = dcomplex(2,1) ; y0=dcomplex(1,1) ;
    n=10 ;
    picard( &func, x0, y0, y, n ) ;
    cout << y << endl ;
}
```

となる。これを実行すると

(1,1)+(2.66667,-2)*(x-((2,1)))+(-3.11111,-8.22222)*(x-((2,1)))^2+(-23.1111,2.2963)*(x-((2,1)))^3+(-10.0494,60.4691)*(x-((2,1)))^4+(147.177,67.2099)*(x-((2,1)))^5+(272.524,-328.804)*(x-((2,1)))^6+(-650.381,-920.673)*(x-((2,1)))^7+(-2793.56,1023.92)*(x-((2,1)))^8+(687.271,7823.14)*(x-((2,1)))^9+(20408.2,3630.9)*(x-((2,1)))^10

が得られる。この問題の場合、等号比較演算必要だったので、追加記述した。この問題では問題は起こらないが、C++言語の複素数には、逆三角関数等が定義されていないので、その部分も追加記述しなければならない。

5.まとめ

Taylor 展開プログラムを作成した。この計算法は数式処理と異なり高速であり、従来の数値計算では精度良く計算できない微分計算が容易に行えるなどの特徴を持つ。

この Taylor 展開を template クラスに変更することによって、複素数を係数にもつ Taylor 展開式を容易に扱えるようになった。

分数や高精度計算ができるクラスが準備されているならば、厳密な分数計算や高精度計算が可能になる。大きな問題に分数計算や高精度計算は行わないが、小さな問題でその精度の検証などに使える。

例題としては、使わなかったが、区間数を係数として持つ Taylor 級数も容易に出来る。Taylor 級数の微分係数の演算機能と区間数を組み合わせれば微分係数を精度よく計算が可能で区間幅の狭い区間数が容易に計算できる。

出力ルーチンを無視すれば、Taylor 展開の係数を Taylor 展開することが容易にできる。これを使えば、多次元の Taylor 展開も容易にできると思われる。

参考文献

[1] Hairer E., Wanner G., Solving Ordinary Differential Equations II, Springer-Verlag, 1991

[2] 平山、小宮、佐藤, Taylor 級数法による常微分方程式の解法, 日本応用数理学会論文誌, 12(2002), pp. 1-8

[3] 平山 弘, " Boole の総和公式による無限交代級数の加速法" ,日本応用数理学会、Vol 10. 4(2000), pp.319-326,2000

[4] Hirayama H., "Numerical Technique for Solving an Ordinary Differrential Equation by Picard's Methods", Integral Methods in Science and Engineering, Birkhauser, Berlin, 2002, pp.111-116

[5] 佐野理, キーポイント微分方程式, 岩波書店, 東京(1993)