Vol.2020-HPC-174 No.4
2020/5/13

関数の数値的Taylor展開とその応用

平山 弘[1,a)]

概要：有限項で打ち切った Taylor 級数の四則演算および関数計算は、プログラミング言語を使えば容易に行うことができる。C++言語や Fortran 等のオペラレーター・オーバーロード機能を使えば非常に使いやすくなる。通常の数値のように扱うことができる。
このプログラムを使うと、これらのプログラミング言語で記述された関数は容易に Taylor 展開できる。
無限ループを含む関数でも Taylor 展開できることを示す。Taylor 展開式が得られれば、これまで使われてこなかった関数の微分係数を含む多数の公式が数値計算等で利用できることを示す。関数の微分係数使った数値積分公式は単に計算が可能であると言うだけでなく従来からある高性能の数値積分と同等以上の性能を発揮することができることを示す。

キーワード：数値的 Taylor 展開，高精度微分係数，数値積分

Numerical Taylor expansion of functions and its applications

Hiroshi Hirayama[1,a)]

Abstract: The arithmetic operations and function calculations of Taylor series truncated by finite terms can be easily defined using programming languages. Using an operator overload function such as C ++ or Fortran makes it very easy to use. It can be treated like a numerical calculation.
Using this program, functions written in these programming languages can easily be expanded into Taylor series.
It is shown that Taylor expansion can be performed even for functions containing infinite loops. Obtaining the Taylor expansion formula shows that many formulas including derivatives of functions that have not been used so far can be used in numerical calculations and the like. Numerical integration formulas using the derivative of a function show that not only can they be calculated, but they can also perform at least as well as conventional high-performance numerical integration.

Keywords: Numerical Taylor series, Numerical integration, High-precition dirivative of functions

1. はじめに

Taylor 級数の係数を浮動小数点数の配列で表現し、有限項で打ち切った Taylor 級数をここでは Taylor 級数と呼ぶことにする。オペレータ・オーバロード機能を持つ C++言語、Fortran 等を使うと、これらの Taylor 級数間の演算を通常の数値計算のように扱うことができる。

このように計算された Taylor 級数の係数は、関数値、1 階および高階微分係数を階乗で割った数値になる。微分係数の計算に差分近似による計算を行っていないので、打ち切り誤差が入らないため通常の数値計算と同様に高精度で計算出来る。

本論文では、この Taylor 展開法を簡単に説明し、いろいろな関数の展開例を示す。特に通常の Taylor 展開法を適用するのが困難な例を示す。

Taylor 展開できると、高階微分係数が高精度出来るので、高階微分係数を使った Euler-Maclaurin 公式による数値積分の計算例を示す。この結果を見ると、よく使われている

[1] 神奈川工科大学創造工学部自動車システム開発工学科
Department of Vehicle System Engineering, Faculty of Creative Engneering, Kanagawa Institute of Technology, Shimo-Ogino 1030, Atsugi, Kanagawa, 243-0292, Japan
[a)] hirayama@kanagawa-it.ac.jp

高性能な数値積分公式と同程度の性能を持つ数値積分公式となることを示す。ここでの数値計算には、主に Visual Studio 2017 の C++言語を使用した。

2. Taylor 級数の計算法

ここでは、関数を Taylor 級数に展開する方法を簡単に説明する。詳しくは Rall[6] および 平山等 [1] を参照せよ。

展開位置が同じ Taylor 級数間の演算は、一般性を失うことなく、原点で展開された Taylor 級数であると仮定することができる。Taylor 級数の展開位置は、平行移動により任意の位置に移動できるため、原点で展開された Taylor 級数のみを考慮すれば十分である。n 次の Taylor 級数を次のように定義する。

$$f(x) = f_0 + f_1x + f_2x^2 + f_3x^3 + f_4x^4 \cdots + f_nx^n \quad (1)$$
$$g(x) = g_0 + g_1x + g_2x^2 + g_3x^3 + g_4x^4 \cdots + g_nx^n \quad (2)$$
$$h(x) = h_0 + h_1x + h_2x^2 + h_3x^3 + h_4x^4 \cdots + h_nx^n \quad (3)$$

2.1 Taylor 級数の四則演算

n 次の Taylor 級数の四則演算プログラムは容易に作成できる。以下の公式は、原点で展開された級数だけでなく、任意の点で展開された Taylor 級数でも有効である。

(1) 加減算

もし $h(x) = f(x) \pm g(x)$ ならば f, g の係数から h の係数は $h_j = f_j \pm g_j (j = 0, \cdots, n)$ と計算できる。

(2) 乗算

もし $h(x) = f(x)g(x)$ ならば f, g の係数から h の係数は $h_j = \sum_{k=0}^{j} f_i g_{j-i} (j = 0, \cdots, n)$ と計算できる。

(3) 除算

もし $h(x) = \dfrac{f(x)}{g(x)}$ ならば f, g, h の係数の関係式は $g_0 \neq 0$ ならば、次のようになる。

$$h_0 = \frac{f_0}{g_0}, \qquad h_j = \frac{1}{g_0}\left(f_j - \sum_{k=0}^{j-1} h_k g_{j-k}\right) \quad (j \geq 1)$$

もし $f_i = g_i = 0$ $(i = 0, \cdots, j)$ ならば、分子と分母を x^{j+1} で割り、その式を上の計算式で除算を行うことができる。

2.2 Taylor 級数の数学関数計算

多くの基本関数は、単純な微分方程式を満たす。この微分方程式を利用することにより、Taylor 級数の関数を容易に計算できる。

(1) 指数関数

もし $h(x) = e^{f(x)}$ ならば、次の微分方程式を満たす。

$$\frac{dh(x)}{dx} = h(x)\frac{df(x)}{dx}$$

(1) と (3) をこの微分方程式に代入し、同じ次数の係数を比較すると次の関係式が得られる。

$$h_0 = e^{f_0}, \qquad h_j = \frac{1}{j}\sum_{k=1}^{j} k h_{j-k} f_k \quad (j \geq 1)$$

(2) 対数関数

もし $h(x) = \log f(x)$ ならば、次の微分方程式を満たす。

$$f(x)\frac{dh(x)}{dx} = \frac{df(x)}{dx}$$

(1) と (3) をこの微分方程式に代入し、同じ次数の係数を比較すると次の関係式が得られる。この式から、次のような関係式が得られる。

$$h_0 = \log f_0,$$
$$h_j = \frac{1}{jf_0}\left(nf_j - \sum_{k=1}^{j-1} k h_k f_(j-k)\right), \quad (j = 1, \cdots, m)$$

(3) べき乗関数

もし $h(x) = f(x)^\alpha$(α は定数) ならば、次の微分方程式を満たす。

$$f(x)\frac{dh(x)}{dx} = \alpha\frac{df(x)}{dx}h(x)$$

(1) と (3) をこの微分方程式に代入し、同じ次数の係数を比較すると次の関係式が得られる。

$$h_0 = f_0^\alpha, \qquad h_j = \frac{1}{jf_0}\sum_{k=1}^{j}(k(\alpha+1) - j) f_k h_{j-k} \quad (j \geq 1)$$

$\alpha = \frac{1}{2}$ とすると平方根を計算できる。

(4) 三角関数

もし $g(x) = \sin f(x)$, $h(x) = \cos f(x)$ ならば、次の連立微分方程式を満たす。

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

(1)、(2) と (3) をこの微分方程式に代入し、同じ次数の係数を比較すると次の関係式が得られる。

$$g_0 = \sin f_0, \qquad g_j = \frac{1}{j}\sum_{k=1}^{j} k h_{j-k} f_k$$

$$h_0 = \cos f_0, \qquad h_j = -\frac{1}{j}\sum_{k=1}^{j} k g_{j-k} f_k \qquad (j = 1, \cdots, n)$$

三角関数は、このように sin と cos を同時に計算すると、計算式が単純で見易い公式となる。sin と cos を同時に計算する関数 sin_cos(x,s,c) を準備すると便利である。tan はこのようにして得られた sin と cos の Taylor 級数をわり算することによって得る。この事情は、以下の双曲線関数 sinh と cosh の場合も同様である。

(5) 微分

$h(x) = \frac{df(x)}{dx}$ のとき、次のような関係式が得られる。

$$h_n = 0, \quad h_j = (j+1)f_{j+1} \qquad (j = 0, \cdots, n) \qquad (4)$$

このように、最高次数の係数 h_n は、0 とする。

(6) 積分

$h(x) = \int_0^x f(t)dt$ のとき次のような関係式が得られる。

$$h_0 = 0, \quad h_j = \frac{1}{j} f_{j-1} \qquad (j = 1, \cdots, n) \tag{5}$$

定数項は、積分定数なので、任意で良いが、ここで作成したプログラムでは、0 とする。

(7) 逆三角関数

逆三角関数には次の 3 関数がある。

$$h(x) = \sin^{-1} f(x) \;=\; \mathrm{asin}(f(x))$$
$$h(x) = \cos^{-1} f(x) \;=\; \mathrm{acos}(f(x))$$
$$h(x) = \tan^{-1} f(x) \;=\; \mathrm{atan}(f(x))$$

これらの関数を微分すると、次の式が成り立つ。

$$\frac{d}{dx}\sin^{-1} f(x) \;=\; \frac{f'(x)}{\sqrt{1-f(x)^2}}$$
$$\frac{d}{dx}\cos^{-1} f(x) \;=\; -\frac{f'(x)}{\sqrt{1-f(x)^2}}$$
$$\frac{d}{dx}\tan^{-1} f(x) \;=\; \frac{f'(x)}{1+f(x)^2}$$

定数項を考慮して、積分すると次の式が成り立つ。

$$\sin^{-1} f(x) \;=\; \sin^{-1} f_0 + \int_0^x \frac{f'(t)}{\sqrt{1-f(t)^2}}dt$$
$$\cos^{-1} f(x) \;=\; \cos^{-1} f_0 - \int_0^x \frac{f'(t)}{\sqrt{1-f(t)^2}}dt$$
$$\tan^{-1} f(x) \;=\; \tan^{-1} f_0 + \int_0^x \frac{f'(t)}{1+f(t)^2}dt$$

他の数学関数についても、同様な微分方程式が得られる場合がある。得られた微分方程式から、Taylor 級数の数学関数を計算することができる。

これらの計算式から分かるように、n 次の Taylor 級数が与えられれば、それらの級数の加減乗算、指数対数関数、三角関数等が n 次まで正確に計算できる。

3. Taylor 級数の具体的な計算例

関数を Taylor 級数展開するには、次の公式を使う。

$$f(a+x) = f(a)+f'(a)x+\frac{f''(a)}{2!}x^2+\cdots+\frac{f^{(n)}}{n!}x^n+\cdots \tag{6}$$

関数 $f(x)$ を $x = a$ で Taylo 展開するには、次の公式によって行うことができる。

$$\begin{aligned} f(x) \;&=\; f(a+(x-a)) = f(a) + f'(a)(x-a) \\ &+\; \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^{(n)}}{n!}(x-a)^n + \cdots \end{aligned} \tag{7}$$

この公式は式 (6) で x を $(x-a)$ に変更することによって、得ることができる。$(x-a)$ を t と置いて、x に $a+t$ を代入して、t の Taylor 級数を計算する。その後 t を $(x-a)$ と置き換えると (7) の Taylor 級数が得られる。

3.1 多項式の計算

次の関数を $x = 2$ で次の $f(x)$ を Taylor 展開する。

$$f(x) = 1 + x + x^2 \tag{8}$$

これを展開するために、$f(2 + (x - 2))$ を計算する。$y = (x - 2)$ と置いて計算すると

$$f(x) = 1 + (2 + y) + (2 + y) * (2 + y) = 7 + 5y + y^2$$

y を x の式に戻すと求める展開式になる。

$$f(x) = 7 + 5(x - 2) + (x - 2)^2$$

前節の Taylor 展開の計算は、Taylor 級数間の計算式であるので、関数 $f(x)$ の x に代入するのは、x を Taylor 展開したものである。関数 $f(x)$ を $x = 2$ で展開するには、x を $x = 2$ で Taylor 展開した式 $2 + (x - 2)$ を代入する。

これを C++言語用のプログラムは以下のようになる。

```
 1 :  #include "taylor_template.h"
 2 :  typedef taylor_template<double> taylor ;
 3 :  int main()
 4 :  {
 5 :      taylor x, f ;
 6 :      x = taylor( 2.0, 2.0, 1.0);
 7 :      cout << "x=" << x << endl ;
 8 :      f = 1+x+x*x ;
 9 :      cout << "f=" << f << endl ;
10 :  }
```

関数 taylor(a,b,c) は $a + b(x - a)$ を定義する関数なので、これを使って 6 行目で taylor(2.0, 2.0, 1.0) と書いて、$2.0 + 1.0(x - 2.0)$ を定義し、変数 x に代入する。7 行目で代入された Taylor 展開式が出力する。8 行目で関数 $f(x)$ を計算して、その結果を 9 行目で出力する。

出力は以下のようになる。

```
x=2+(x-2)
f=7+5*(x-2)+(x-2)^2
```

計算結果は、$f(2) = 7$、$f'(2) = 5$、$f''(2) = 2$ を意味する。この計算に、差分近似は使っていないので、高精度で計算出来る。

3.2 平方根を含む計算

次の関数を $x = 2$ で次の $f(x)$ を Taylor 展開する。

$$f(x) = \sqrt{7 - x^2} \tag{9}$$

これを展開するために、$f(2 + (x - 2))$ を計算する。$y = (x - 2)$ と置いて計算すると次のようになる。

$$f(2 + y) = \sqrt{7 - (2 + y)^2} = \sqrt{3 - 4y - y^2}$$

平方根を次のべき乗関数の計算式を使って計算する。

$$h_0 = f_0^{\alpha}, \qquad h_n = \frac{1}{nf_0}\sum_{k=1}^{n}(k(\alpha+1)-n)f_k h_{n-k} \quad (n \geq 1)$$

$\alpha = \frac{1}{2}$、$f_0 = 3,\ f_1 = -4$、$f_2 = -1$ として計算する。定数項 $h_0 = \sqrt{3} = 1.73205$、$h_1 = -\frac{2}{3}\sqrt{3} = -1.1547$、$h_2 = -\frac{7}{18}\sqrt{3} = -0.673575$、... と得られる。平方根の計算では、定数項の計算では平方根の計算が必要であるが、1 次以上の係数は四則演算の結果に定数項を乗算したものになる。この特徴は、指数関数、対数関数などでも同様な性質がある。これを倍精度で計算する C++言語用のプログラムは以下のようになる。

```
 1 :  #include "taylor_template.h"
 2 :  typedef taylor_template<double> taylor ;
 3 :  int main()
 4 :  {
 5 :      taylor x, f ;
 6 :      x = taylor( 2.0, 2.0, 1.0);
 7 :      cout << "x=" << x << endl ;
 8 :      f = sqrt(7-x*x) ;
 9 :      cout << "f=" << f << endl ;
10 :  }
```

このプログラムを実行すると以下のような結果になる。

```
x=2+(x-2)
f=1.73205-1.1547*(x-2)-0.673575*(x-2)^2
 -0.44905*(x-2)^3-0.43034*(x-2)^4-0.461524*(x-2)^5
```

上の計算結果から、$f(x) = \sqrt{7-x^2}$ の高階微分係数の容易に計算出来る。

3.3 ゼロ割を含む計算

次の関数を $x = 0$ で Taylor 展開する。

$$f(x) = \frac{x}{e^x - 1} \tag{10}$$

$f(0)$ とすると、ゼロ割が生じ Taylor 展開式の定数項は計算出来ない。このような場合にも Taylor 級数として計算すれば、容易に計算出来る。(10) は、次のように、分子と分母に分けて Taylor 展開すると次のようになる。

$$f(x) = \frac{x}{e^x - 1} = \frac{x}{x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots} \tag{11}$$

分子と分母を x で割ると次のようになる。

$$f(x) = \frac{x}{e^x - 1} = \frac{1}{1 + \frac{x}{2!} + \frac{x^2}{3!} + \cdots} \tag{12}$$

この式から、$f(0) = 1$ が得られる。Taylor 級数の割り算を行うことによって、$f(x)$ の Taylor 展開式が得られる。

C++言語用のプログラムは以下のようになる。

```
 1 :  #include "taylor_template.h"
 2 :  typedef taylor_template<double> taylor ;
 3 :  int main()
 4 :  {
 5 :      taylor x, y ;
 6 :      x = taylor( 0.0, 0.0, 1.0);
 7 :      y = x/(exp(x)-1) ;
 8 :      cout << y << endl ;
 9 :  }
```

ここで、コンストラクター taylor(a, b, c) は $b+c(x-a)$ を定義する。この計算では、除算演算で分子と分母の Taylor 級数の定数項がゼロになるので、分子と分母を x で割ってから除算を行っている。(10) の $f(x)$ を $x = 0$ で Taylor 展開し 10 次まで表示すると以下のようになる。係数の絶対値が 10^{-10} 以下のものはゼロとして、表示しないと以下のようになる。

```
1-0.5*x+0.0833333*x^2-0.00138889*x^4
 +3.30688e-05*x^6-8.2672e-07*x^8+2.08768e-08*x^10
```

4. 微分方程式満たさない関数の Taylor 展開

Riemann の Zeta 関数 ($\zeta(s)$) を解とする微分方程式は、公式集を見ても存在しない。このため、これまで挙げた例題のように簡単に Taylor 展開をすることはできない。

ここでは、無限級数の形式になっている次の公式を使って計算する。

$$\begin{aligned}\zeta(s) &= \sum_{k=1}^{\infty}\frac{1}{k^s} \\ &= \frac{1}{1-2^{1-s}}\sum_{n=0}^{\infty}\frac{1}{2^{n+1}}\left(\sum_{k=0}^{n}(-1)^k\begin{pmatrix} n \\ k \end{pmatrix}(1+k)^{-s}\right)\end{aligned}$$

ここでは、$\zeta(x)$ を $x = 4$ で Taylor 展開する。$\zeta(4+(x-4))$ として、上の式に代入して、Taylor 展開する。これを実行するするプログラムは次のようになる。

```
 1: #include <iostream>
 2: #include "taylor_template.h"
 3: #include <cmath>
 4: #include <cstdio>
 5: using namespace std ;
 6: int nn ;
 7: typedef taylor_template<double> taylor ;
 8: taylor zeta( const taylor &s )
 9: {
10:     taylor   t, u, u0 ;
11:     double  p, cmb ;
12:     nn=0 ;
13:     u = 0 ;
14:     p = 1 ;
15:     for( int n=0 ; ; n++ )
16:     {
17:         nn++ ;
18:         t=0 ;
19:         cmb = 1 ;
20:         for( int k=0 ; k<=n ; k++ )
```

```
21:         {
22:             t += cmb*exp( -log(double(k+1))*s ) ;
23:             cmb *= k-n ;
24:             cmb /= k+1 ;
25:         }
26:         p *= 2 ;
27:         u0 = u ;
28:         u += t/p ;
29:         nn++ ;
30:         if(abs((u[6]-u0[6])/u[6])<1.0e-14)break ;
31:     }
32:     u /= 1.0-exp(log(2.0)*(1.0-s)) ;
33:     return u ;
34: }
35: int main()
36: {
37:     taylor x, y ;
38:     x = taylor( 4.0, 4.0, 1.0 ) ;
39:     cout << "x=" << x << endl ;
40:     y = zeta( x ) ;
41:     cout << "y=" << y << endl ;
42:     for( int i=0 ; i<15 ; i++ )
43:     {
44:         printf("%2d : %20.15f\n", i, y[i] ) ;
45:     }
46:     cout << "nn=" << nn << endl ;
47: }
```

公式では計算は無限ループになっているので、有限回で計算を打ち切らなければならない。このプログラムでは 6 次の係数の相対誤差が 1.0e-14 以下になった場合、計算を打ち切るようになっている。これを実行し、Taylor 展開を 16 次まで出力する。14 次までの係数は高精度で出力すると次の結果が得られる。

```
x=4+(x-4)
y=1.08232-0.0689113*(x-4)+0.0325291*(x-4)^2
 -0.0121068*(x-4)^3+0.00412531*(x-4)^4
 -0.00137502*(x-4)^5+0.000457551*(x-4)^6
 -0.000152426*(x-4)^7+5.08043e-05*(x-4)^8
 -1.69349e-05*(x-4)^9+5.64502e-06*(x-4)^10
 -1.88168e-06*(x-4)^11+6.27226e-07*(x-4)^12
 -2.09075e-07*(x-4)^13+6.96917e-08*(x-4)^14
 -2.32306e-08*(x-4)^15+7.74352e-09*(x-4)^16
0 :     1.082323233711138
1 :    -0.068911265896125
2 :     0.032529080683940
3 :    -0.012106808315220
4 :     0.004125308751018
5 :    -0.001375022157713
6 :     0.000457551091280
7 :    -0.000152425838544
8 :     0.000050804268571
9 :    -0.000016934914858
10 :    0.000005645016844
11 :   -0.000001881676094
12 :    0.000000627225505
13 :   -0.000000209075163
14 :    0.000000069691720
nn=102
```

この Taylor 展開式の定数項は $\zeta(4) = \frac{\pi^4}{90} = 1.082323233711138191516\cdots$ でなければならない。この値は上の表示した結果と完全に一致している。最後の行の nn=102 は、この計算には $k = 102$ までループ計算が必要だったことを示す。

$\zeta(x)$ を $x = 0.5$ で Taylor 展開し、0 から 14 次までの係数を出力すると次のような結果が得られる。

```
 0 :        -1.460354508809588
 1 :        -3.922646139209154
 2 :        -8.004178506964339
 3 :       -16.000551540886534
 4 :       -31.999888321685358
 5 :       -64.000005005517338
 6 :      -128.000000759080052
 7 :      -255.999999860590520
 8 :      -512.000000008374286
 9 :     -1024.000000000293312
10 :     -2047.999999999907232
11 :     -4096.000000000022737
12 :     -8192.000000000030923
13 :    -16384.000000000065484
14 :    -32768.000000000138243
```

数式処理ソフトウエア maxima を使うと $\zeta(0.5) = -1.460354508809587$ が得られる。上の計算結果と比較すると最後の 1 桁を除いて完全に一致する。高次の係数を見るとほぼ 2 のべき乗になっている。このことから $x = 1$ が特異点であることがわかる。

5. Euler-Maclaurin の総和公式による数値積分

Euler-Maclaurin の総和公式とは、つぎのような公式である。関数 $f(x)$ は区間 $[a.b]$ で連続で微分可能とする。積分区間 $[a, b]$ を n 等分して、台形公式を適用した計算値と厳密な積分値との差を微分係数を含む m 項で近似する公式である。$h = \frac{b-a}{n}$ と置くと、次のような公式になる。

$$h\left\{\frac{1}{2}f(a) + \sum_{k=1}^{n-1} f(a+kh) + \frac{1}{2}f(b)\right\} - \int_a^b f(x)dx \quad (13)$$
$$= \sum_{k=1}^{m} \frac{B_{2k}}{(2k)!} h^{2k} \left\{f^{(2k-1)}(b) - f^{(2k-1)}(a)\right\} + R_m$$

ここで、B_n は Bernoulli 数 (Bernoulli Number)、$h = \dfrac{b-a}{n}$ であり、

$$|R_m| \leq \frac{h^{2m+2}\,|B_{2m+2}|}{(2m+2)!}\int_a^b \left|f^{(2m+2)}(t)\right| dt \tag{14}$$

Bernoulli 数を次のように定義する。

$$\frac{x}{e^x-1} = \sum_{k=0}^{\infty} B_k \frac{x^k}{k!} \tag{15}$$

B_n の計算は、3.3 節の例題からわかるように、Taylor 展開式を計算することによって計算出来る。

この公式は、微分係数を含むため、これまであまり利用する利用されることはなかった。ここでは、この公式の性能評価を行う。

まず、積分区間の両端点において、Taylor 展開を計算する。この Taylor 展開を計算するプログラムは、通常の関数値の計算と宣言部分を除いてほぼ同じになる。このような Taylor 展開を行うためのテンプレート・プログラムも公開されている [2] ので、その計算は容易である。テンプレート機能を使えば Taylor 展開と関数値を計算するプログラムは多くの場合 1 個のプログラムで記述できる。実際この論文で示した計算例は、C++言語のテンプレート機能を使って 1 個の関数プログラムの形で記述した。たとえば、$0.92\cosh x - \cos x$ をプログラムで記述するには、次の様に書ける。

```
template<typename T>
T func( const T& x )
{
    T s ;
    s =0.92*cosh(x)-cos(x) ;
    return s ;
}
```

このように 1 個関数プログラムを書けば、関数計算や Taylor 展開の計算に使える。

次に分割数 n を決める。分割数がある程度推定できる場合、その値にする。何もなければ最小の 2 にする。この分割数を利用して、台形公式を利用して、積分の近似値を計算する。近似値を次の項を利用して、補正する。この級数は漸近級数なので漸近級数の手法を使って計算する。

$$c(k) = \frac{B_{2k}}{(2k)!}h^{2k}\left\{f^{(2k-1)}(b) - f^{(2k-1)}(a)\right\} \tag{16}$$

この補正項 $c(k)$ の絶対値が要求精度以下になるまで、補正項を減算して行く。補正項の絶対値が要求精度より小さくなったら、その項を使って補正し計算を止める。ここまでの計算が求める積分値になる。補正項の絶対値が要求精度より小さくならない場合や逆に大きくなった場合、補正するのを止め、分割数 n を 2 倍にして、台形公式による積分の計算に戻る。分割数を 2 倍にすると、その前の台形公式計算値が利用し易くするためである。この手順を繰り返す事によって積分値を計算する。

5.1 簡単な数値積分例

簡単な例として、次の積分を計算する。これを要求精度 10^{-9} で計算する。

$$I = \int_0^1 \frac{1}{1+x}dx = \log 2 \tag{17}$$

最初に、積分区間の両端点で Taylor 展開をする。展開は 20 次まで行った。その計算結果を以下に示す。$x = 0$ における Taylor 展開を 12 次まで表示すると次の様になる。

$$1-x+x^2-x^3+x^4-x^5+x^6-x^7+x^8-x^9+x^{10}-x^{11}+x^{12}$$

x = 1 における Taylor 展開を 4 次まで表示すると次の様になる。

$$0.5-0.25(x-1)+0.125(x-1)^2-0.0625(x-1)^3+0.03125(x-1)^4$$

$n = 2$ として、台形公式で数値積分すると、0.708333333333333 となる。これを補正する。この場合の補正項は収束が非常に遅く要求精度を満たすことができない。$k = 8$ で収束どころか逆に絶対値が増加し始める。

このため分割数を 2 倍の $n = 4$ として、台形公式を使って数値積分を行う。このときの積分値は、0.697023809523809 となる。この結果を再度補正する。ここでのは補正項も十分速く小さくなるとは言えないが、$k = 7$ で補正項が 3.1×10^{-9} となり要求精度より小さくなった。そこまでの計算値が積分値となる。積分値は 0.6931471804863029718 となる。要求精度通りの結果が得られた。このときの関数計算回数は、5 回でその内 2 回が Taylor 展開を計算するための関数計算であった。

分割数 n を増加させると、h は小さくなるので補正項 $c(k)$ はすばやく小さくなる。このため、高次の微分係数はあまり必要としなくなる。Taylor 展開は低次数でも Euler-Maclaurin の総和公式を利用できることになる。逆に Taylor 展開が高次数であるならば、分割数 n を小さく出来る。上の計算では 20 次の Taylor 展開を利用したが、それが適切かどうか考慮する必要がある。

5.2 数値例

Euler-Maclaurin の総和公式を利用した計算法の性能を評価するために、Kahaner の問題の中から両端点で特異性を持つため Taylor 展開できない問題等を除いた 13 問題について計算を行った。計算した問題を表 1 に示す。番号は Kahaner の問題の番号である。それを利用して計算した結果を示す。

実行結果を表 2 に示す。N は関数の計算回数、error はその積分ルーチンが出力した誤差である。関数の計算回

数の内 2 回は 10 次の Taylor 展開の計算である。EM は Euler-Maclaurin の総和公式を利用した計算、DE は二重指数型積分公式 [4](Double Exponential formula)、DAQN9 は、適応型ニュートン・コーツ法 [3] の結果はこの参考文献による結果である。IBM 型 64 ビットの浮動小数点演算による結果である。DE のプログラムとして、ネットで公開されている大浦 [5] の C 言語用 DE プログラムを C++ 言語に変換し使用した。

6. 終わりに

Taylor 展開を数値的に行う方法を簡単に説明した。この方法を利用すれば、解析的には非常に難しいと思われる関数も容易に Taylor 展開ができる。このような関数例として、Riemann の zeta 関数の Taylor 展開を行った。

微分係数を含むためこれまであまり利用されていない Euler-Maclaurin の総和公式を利用して、数値積分する方法を提案した。この方法は、単純でわかり易い計算法であるにも関わらず、多くの場合、有力な数値積分法と同等程度またはそれ以上の性能を発揮することを示した。

参考文献

[1] 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), 1–8.
[2] 平山, 舘野, 浅野, 川口, Taylor 級数演算ライブラリの使用法, 東北大学情報シナジーセンター大規模科学計算システム広報 SENAC, 40(2007) 29–68
[3] 二宮市三, 適応型ニュートン・コーツ積分法の改良, 情報処理学会誌,**21** (1980),504–513
[4] Takahasi, H. and Mori, M., Double exponential formula for numerical integration, Publ. RIMS, Kyoto Univ.,**9** (1974),121–141
[5] 大浦拓哉, Ooura’s Mathematical Software Packages, http://www.kurims.kyotou. ac.jp/ooura/index-j.html
[6] Rall,L.B., Automatic Differentiation-Technique and Applications, Lecture Notes in Computer Science, Vol.120, Springer-Verlag, New York(1981)

表 1　Test Problem

$No.$	Integral
1	$\int_0^1 e^x dx = 1.7182818285$
4	$\int_{-1}^1 0.92\cosh x - \cos x dx = 0.47942822669$
5	$\int_{-1}^1 \frac{dx}{x^4+x^2+0.9} = 1.5822329637$
8	$\int_0^1 \frac{dx}{x^4+1} = 0.86697298734$
9	$\int_0^1 \frac{2dx}{2+\sin(31.4159x)} = 1.1547006690$
10	$\int_0^1 \frac{dx}{1+x} = 0.69314718056$
11	$\int_0^1 \frac{dx}{e^x+1} = 0.37988549304$
12	$\int_0^1 \frac{x}{e^x-1} dx = 0.77750463411$
13	$\int_{0.1}^{10} \frac{\sin(314.159x)}{3.14159x} dx = 0.0090986452566$
16	$\int_0^{10} \frac{50}{3.14159(2500x^2+1)} dx = 0.49936380287$
17	$\int_{0.01}^1 (\frac{\sin(50\times x)}{50\times 3.14159x})^2 \times 50 dx = 0.11213956963$
18	$\int_0^\pi \cos(\cos x + 3\sin x + 2\cos 2x + 3\sin 2x + 3\cos 3x) dx = 0.83867632338$
20	$\int_{-1}^1 \frac{dx}{x^2+1.005} = 1.5643964441$

表 2　Performance of Quadrature Routines($\epsilon = 10^{-9}$)

	EM		DE		DAQN9	
$No.$	N	error	N	error	N	error
1	3	3.5e-11	33	3.4e-9	25	5.2e-18
4	3	2.5e-10	67	2.2e-9	25	3.5e-17
5	9	2.2e-10	65	6.3e-9	61	1.8e-11
8	9	1.7e-10	65	3.5e-9	91	8.9e-12
9	33	5.4e-10	519	3.7e-9	81	1.0e-5
10	5	3.1e-10	33	1.4e-9	21	3.9e-13
11	3	2.6e-10	33	7.6e-11	21	3.3e-18
12	3	7.3e-11	33	1.6e-9	21	2.2e-18
13	129	7.9e-10	461	1.2e-8	321	6.2e-7
16	5	3.2e-10	141	4.2e-9	91	5.2e-7
17	129	1.9e-10	555	5.7e-9	101	7.4e-4
18	33	1.1e-10	131	1.7e-8	51	5.4e-6
20	17	3.1e-13	65	6.3e-9	21	1.1e-8