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

2005－ARC－162（26）
2005－HPC－101（26）
2005／3／9

Taylor 展開を利用した高精度数値積分法

平山　弘、金子尚広、花井啓助

神奈川工科大学

Taylor 級数の四則演算や関数計算は、C++言語や Fortran を使うと容易に定義できる。四則演算、関数および条件文で記述された関数は高速に Taylor 展開できる。これを利用すると積分を任意次数まで容易に Taylor 展開することができる。この Taylor 展開を評価すると効果的な任意次数の高精度数値積分法になる。さらに、この方法は積分の多倍長精度にも適用できる。

High Precision Numerical Integration Method by Taylor Seires

Hiroshi Hirayama, Takahiro Kaneko and Keisuke Hanai

Kanagawa Institute of Technology

The arithmetic operations and functions of Taylor series can be defined by Fortran 90 and C++ language. The functions which consist of arithmetic operations, pre-defined functions and conditional statements can be expanded in Taylor series quickly. Using this, Taylor series of the integral can be expanded up to arbitrary order easily. Evaluating the Taylor series gives an effective, arbitrary and high precision numerical integration method for the integrals. In addition, it is shown that this integration method can be applied to multiple-precison evaluation of the integral.

1．はじめに

プログラムでよく使われる演算子（+,-,*, /など）を、被演算の型が異なる場合、別の意味を与えることができる機能、C++言語[1]の機能（operator overload）などを使い、有限項で打ち切った Taylor 級数の四則演算、Taylor 級数の関数演算を容易に定義することができる。この機能を使うと、通常のプログラムの形で与えられた任意の数値関数をわずかな変更で Taylor 級数に展開できるプログラムに変更できる。

Taylor 展開は、係数を倍精度の浮動小数点で表現した場合、通常の数値計算と同じ程度の速度で計算することができる。これまでのこの種の計算に使われている数式処理に比べ、桁違いに高速である。

本論文では、この高速性を利用して、積分を Taylor 展開し、その Taylor 展開を評価することによって積分値を計算する方法を提案する。この計算方法は計算機が発明される以前までは最も普通に使われる方法であったが、計算機上の計算方法としてはこれまであまり使われたことのない方法である。

2．Taylor 展開

この節では、関数を Taylor 展開するための基本的な考え方を説明し、その計算方法について簡単に論じる。詳しくは、Rall[6]、

Henrici[2]や平山[3-5]などに述べられている。

2.1 Taylor 級数の四則演算

Taylor 級数の四則計算のプログラムは、以下のように簡単に作ることができる。平行移動によって、展開位置を原点へ移すことができるので一般性を失うことなしに、原点で展開した式だけを扱うことができる。この級数を次のように定義する。

(2.1) $\mathrm{f}(x)=\mathrm{f}_0+\mathrm{f}_1 x+\mathrm{f}_2 x^2+\mathrm{f}_3 x^3+\mathrm{f}_4 x^4+\cdots$

(2.2) $\mathrm{g}(x)=\mathrm{g}_0+\mathrm{g}_1 x+\mathrm{g}_2 x^2+\mathrm{g}_3 x^3+\mathrm{g}_4 x^4+\cdots$

(2.3) $\mathrm{h}(x)=\mathrm{h}_0+\mathrm{h}_1 x+\mathrm{h}_2 x^2+\mathrm{h}_3 x^3+\mathrm{h}_4 x^4+\cdots$

2.1.1 加減算

$h(x)$ が $f(x)$ と $g(x)$ の和差のとき、f,g および h の係数は、次のような関係になる。

(2.4) $\mathrm{h}(x)=\mathrm{f}(x)\pm\mathrm{g}(x) \quad \mathrm{h}_i=\mathrm{f}_i\pm\mathrm{g}_i$

2.1.2 乗算

$h(x)$ が $f(x)$ と $g(x)$ の積のとき、f,g および h の係数は、次のような関係になる。

(2.5) $\mathrm{h}(x)=\mathrm{f}(x)\mathrm{g}(x) \qquad \mathrm{h}_n=\sum_{k=0}^{n}\mathrm{f}_k\,\mathrm{g}_{n-k}$

2.1.3 除算

$h(x)$ が $f(x)$ と $g(x)$ の商で与えられるとき、すなわち

(2.6) $\mathrm{h}(x)=\dfrac{\mathrm{f}(x)}{\mathrm{g}(x)}$

であるとき、$f(x)$ の係数と $h(x)$ の係数には、以下のような関係が成り立つ。

(2.7) $h_0=\dfrac{f_0}{g_0}$、 $\mathrm{h}_n=\dfrac{1}{g_0}\left(\mathrm{f}_n-\sum_{k=0}^{n-1}\mathrm{h}_k\mathrm{g}_{n-k}\right)$ $(n\ge 1)$

(2.7)の式は、(2.6)において、両辺に $f(x)$ を掛け、(2.1)、(2.2)、および(2.3)を代入して、展開する。両辺の同じ次数の係数が等しいことを利用して得られる。

2.1.4 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、次のような関係になる。

(2.7) 微分 $\mathrm{h}(x)=\dfrac{d\,\mathrm{f}(x)}{dx}$

$\mathrm{f}_m=0,\ \mathrm{h}_n=(n+1)\,\mathrm{f}_{n+1} \quad (n=0,\cdots,m-1)$

(2.8) 積分 $\mathrm{h}(x)=\int f(x)dx$

$\mathrm{h}_0=0, \qquad \mathrm{h}_n=\dfrac{1}{n}\mathrm{f}_{n-1} \qquad (n=1,\cdots,m)$

積分演算で積分定数 h_0 は任意であるが、ここでは、0 とする。

2.2 Taylor 級数の関数

他の多くの初等関数でも同様に係数間の漸化式を導くことができる。ここでは、Taylor 級数の指数関数の計算方法を示す。この方法は後に述べる常微分方程式の解の Taylor 展開の単純な例にもなっている。

$\mathrm{h}(x)=e^{\mathrm{f}(x)}$ とおくと、

(2.9) $\dfrac{d\,\mathrm{h}}{dx}=\mathrm{h}\dfrac{d\,\mathrm{f}}{dx}$

である。(2.9)に(2.1)、(2.3)を代入して、両辺の係数を比較することによって、次のような関係が得られる。

(2.10) $h_0=e^{f_0},\ \mathrm{h}_n=\dfrac{1}{n}\sum_{k=1}^{n}k\,\mathrm{h}_{n-k}\mathrm{f}_k \quad (n\ge 1)$

このように簡単に計算することができる。対数関数や三角関数の場合も同様な公式を得ることができる。

式(2.10)のように、Taylor 展開式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、Taylor 展開の指数関数の計算量は、通常の四則演算とそれほど大きな違いにはならない。このことは、対数関数や三角関数などでも同様である。

3．数値積分法

ここでは、Taylor 級数を使い数値積分する方法を説明する。まず、次のような積分を考える。

(3.1) $I=\int_a^b f(x)dx$

$f(x)$ を積分区間 $[a,b]$ の中点 $x=c$ で n 次の Taylor 級数に展開すると

(3.2)

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

となる。これを積分すると、不定積分 $F(x)$ は

$$F(x) = f_0(x-c) + \frac{f_1}{2}(x-c)^2 + \frac{f_2}{3}(x-c)^3 + \cdots + \frac{f_n}{n+1}(x-c)^{n+1} \tag{3.3}$$

が得られる。もし $F(x)$ が十分速く収束する級数ならば、$x=a$ と $x=b$ を $F(x)$ に代入することによって積分 I が求められる。すなわち

$$I = F(b) - F(a) \tag{3.4}$$

として計算できる。収束が遅い場合、$h = x-c$ とし、$F(c+h)$ を考える。このとき、

$$F(c+h) = f_0 h + \frac{f_1}{2}h^2 + \frac{f_2}{3}h^3 + \cdots + \frac{f_n}{n+1}h^{n+1} \tag{3.5}$$

となる。h を十分小さく取れば、(3.5)の級数は十分速く収束させることができる。

本計算では、積分の要求計算精度を ε としたとき、h を最終項の絶対値が要求精度になるように選ぶ。ε が相対的な要求精度であれば、最初の項との相対的な精度となる。したがって、絶対的および相対的な場合、それぞれ

$$\left|\frac{f_n}{n+1}h^{n+1}\right| \le \varepsilon \qquad \left|\frac{f_n h^n}{(n+1)f_0}\right| \le \varepsilon \tag{3.6}$$

が成り立つように h を選ぶ。このとき、h は、それぞれ

$$h = \sqrt[n+1]{\frac{(n+1)\varepsilon}{|f_n|}} \qquad h = \sqrt[n]{\frac{(n+1)|f_0|\varepsilon}{|f_n|}} \tag{3.7}$$

として計算する。この h を区間幅とすれば、積分は問題なく計算できる。この区間幅は、被積分関数のなめらかであるかまたは Taylor 級数の次数が高いほど広く取ることができる。f_n がゼロならば(3.7)の式は計算できなくなるが、その場合、次数を 1 次づつ下げて、(3.7)式を適用する。残り積分を計算するために、上と同様な操作を繰り返し適用し、積分区間を小さくし、最終的に全区間の積分を行う。

(3.7)の式は、h の $n+2$ 乗以上の項は h の $n+1$ 乗項に比べ十分小さいと仮定している。しかしながら、実際にそのようにならない場合もよく起こる。このようなことが起こっても計算できるように、h を余裕を持たせ、0.8 とか 0.9 倍程度の大きさを h の値としてとる。このようにとると、積分区間幅が小さくなるため Taylor 展開の計算回数は多くなるが、ほぼ確実に要求精度の積分値が得られる。

4 数値例

4.1 簡単な例

次のような積分を計算精度 $\varepsilon = 10^{-10}$ で計算することを考える。

$$I = \int_0^1 e^x dx = 1.71828182845904523\cdots \tag{4.1}$$

e^x を $x=0.5$ で 10 次まで、Taylor 展開すると

```
1.64872*x+0.824361*x^2+0.274787*x^3
+0.0686967*x^4+0.0137393*x^5+0.0022898
9*x^6+0.000327127*x^7+4.08909e-005*x^8
+4.54343e-006*x^9+4.54343e-007*x^10
```

が得られる。この式から、$h = 0.387707$ が得られる。区間[0.112293,0.887707]の積分値は、1.310713 となる。次に残りの 1 区間である区間[0, 0.112293]の積分を計算する。この積分を計算するために $x = 0.0561465$ で e^x を 10 次まで Taylor 展開する。展開式は

```
1.05775*x+0.528876*x^2+0.176292*x^3
+0.044073*x^4+0.0088146*x^5+0.0014691
*x^6+0.000209872*x^7+2.62339e-005*x^8
+2.91488e-006*x^9+2.91488e-007*x^10
```

となる。この式から、$h = 0.405304$ が得られる。このように h を取ると元の積分区間を越えるので、元の積分区間に一致するように h を選ぶと $h = 0.0561465$ となる。h は

十分小さいので、区間[0,0.112293]での積分値が計算できて、0.118840 となる。残りの区間[0.887707,1]を積分するために $x=0.942847$ で e^x を 10 次まで Taylor 展開すると

2.56987*x+1.28493*x^2+0.428311*x^3
+0.107078*x^4+0.0214155*x^5+0.00356926
*x^6+0.000509894*x^7+6.37368e-005*x^8
+7.08186e-006*x^9+7.08186e-007*x^10

となる。この式から h を計算すると $h=0.370875$ となるが、このように取ると元の積分区間を越えるので、元の積分区間に一致するように h を選ぶと $h=0.057153$ となる。

これを使うと区間[0.885694,1]の積分値は 0.288729 となる。各区間の積分値の総和を求め、積分値 1.718281828456585 が得られる。

もし、11 から 13 次の級数展開式を利用すると、上の積分は 2 分割で計算することができる。14 次以上の級数展開式を使えば、積分区間の分割は不要になり、1 回の級数展開で計算することができる。

この計算法を通常の数値積分法と比較するために、この問題を標本点 30 個使うガウス型数値積分法と 16 次 Taylor 級数を使い $e=10^{-14}$ とした本方法の計算時間を測定した。

計算機として 3.06GHz の Pentium 4 とコンパイラーとして Microsoft 社製の.NET2003 Visual C++を使用した。このとき、ガウス型数値積分法は 1.96μsec、本方法 1.63μsec であった。計算時間は 100 万回ループさせ測定した。このように本方法は通常の効率の良い数値積分法と同等の時間で計算できることがわかる。

4.2 Kahaner のテスト問題

本手法の特徴を調べるために、Kahaner のテスト問題[7-8]の計算を試みた。Kahner の問題には、特異点を持つ関数や微分係数が不連続なものも含まれるが、今回はそのような問題を除いた。以下の Table 1 にその結果を示す。

Taylor と表示されている部分には、本手法で計算したときの時間（マイクロ秒）を示す。計算機として Intel P4 3.06GHz を使用した。コンパイラーは Intel 社の C++Ver. 7.0 を使用した。要求精度を 10^{-9} とし、7 次から 22 次までの Taylor 展開を用いて計算した。上の時間はその中で最も良い時間を示している。

参考のため、AQE11D の積分ルーチン[6]で計算した時間を示す。この時間（マイクロ秒）は Intel P4 2.0GHz を使用した場合の結果である。最後の ratio は、もしクロックに比例してプログラムが動作すると仮定した場合の高速化率である。

これらの結果を見ると、Taylor 展開法は性質のよい関数に対しては非常に良い結果を示す。問題 9,13,14,15､18 からわかるように、動きが激しい指数関数（三角関数を含む）の逆数を含む場合は、計算効率が落ちることがわかる。指数関数による急激な減少を起こしてる領域では、Taylor 展開は、非常に広い領域で収束するように見えるが、実際にはそれほど広い領域では収束しないような現象が起こり、現在考えている 20 から 30 次程度の Taylor 展開では、関数を認識するのは非常に困難であるためである。特に問題 14 は非常に計算が難しい問題となっている。

このような問題の場合、積分区間の下限から Taylor 展開し、解析接続しながら計算する方法が単純で有力な方法である。表示された結果は、非常に小さな h が出現した場合、それを大きくするようにした計算ルーチンを使ったときの結果である。

被積分関数が鋭いピークを持つような場合、Taylor 展開を使う方法は、確実ピークを捕らえることができるため、問題 21 のような関数は比較的容易にピークを検出できる。しかしながら、関数自体は指数関数の逆数となっているためかなり計算時間が

かかる。

Table 1 Comparison of performance of the quadrature routine based on Taylor series

Error Requirement 1.0E-09						
No.	a	b	Integrand	Taylor	AQE11D	Ratio
1	0.0	1.0	e^x	0.60	3.09	3.4
4	-1.0	1.0	$0.92\cosh x - \cos x$	2.07	7.50	2.4
5	-1.0	1.0	$1/(x^4 + x^2 + 0.9)$	3.35	8.54	1.7
8	0.0	1.0	$1/(x^4 + 1)$	2.69	6.25	1.5
9	0.0	1.0	$2/(2 + \sin(31.4159x))$	58.59	86.7	0.98
10	0.0	1.0	$1/(1 + x)$	0.94	3.12	2.2
11	0.0	1.0	$1/(e^x + 1)$	1.04	4.23	2.7
12	0.0	1.0	$x/(e^x - 1)$	0.85	7.53	5.9
13	0.1	1.0	$\sin(314.1592x)/(3.141592x)$	109.3	255.	1.6
14	0.0	10.0	$\sqrt{50}e^{-50\cdot 3.14159x^2}$	386	59.6	0.10
15	0.0	10.0	$25e^{-25x}$	37.59	26.2	0.47
16	0.0	10.0	$50/(3.14159(2500x^2 + 1))$	13.67	29.8	1.4
17	0.01	1.0	$50(\sin(50\cdot 3.14159x)/(50\cdot 3.14159x))^2$	61.03	227	2.4
18	0.0	π	$\cos(\cos x + 3\sin x + 2\cos 2x + 3\sin 2x + 3\cos 3x)$	91.81	81.8	0.59
20	-1.0	1.0	$1/(x^2 + 1.005)$	1.40	9.06	4.3
21	0.0	1.0	$\frac{1}{\cosh^2(10(x-0.2))} + \frac{1}{\cosh^4(100(x-0.4))} + \frac{1}{\cosh^6(1000(x-0.6))}$	960	(170)	

5. 多倍長計算

Taylor 展開法は積分公式としては任意次数の公式だから、多倍長精度の計算に適している。多倍長精度を使うと積分公式の次数も高く取れるので、効率的に計算できると期待できる。

高精度のTaylor展開を利用した数値積分法でKahaner の問題を計算を行った。計算精度は、10進数で128桁、要求精度は50桁、次数は最大90次のTaylor展開式を利用して計算した。結果をTable 2に示す。

次数を上げると分割数は少なくなり多くの問題が分割数が10程度になった。問題９と問題13を比較すると通常の倍精度計算では問題13の方が計算時間がかかり難しい問題となっているが、計算次数が上がった場合、逆に問題9の方が分割数、計算時間も多くかかる。

問題21は、通常の精度では非常に計算しにくい問題であるが、高精度の計算では、やや難しい問題にはなるが、倍精度の計算のように他の問題に比べ非常に難しい問題になることはない。

Table 2 : Computational Results

No.	Result
1	1.718281828459045235360287471352662497757 25
4	0.479428226688801667358577961835307500641 39
5	1.582232963729672933117468949026169067924 32
8	0.866972987339911037573995163882870713652 18
9	1.154700669043713043406922209856028140298 56
10	0.693147180559945309417232121458176568075 50
11	0.379885493041722475368236626490320926160 22
12	0.777504634112248276417586545425710507192 48
13	0.009098645256569297069832987226191046605 80
14	0.500000211166100039341004677294863175526 91
15	1.000000000000000000000000000000000000000 00
16	0.499363802871016550828171090340696808551 94
17	0.112139569626709460839858145067099485632 41
18	0.838676342694429614542554699585856192387 67
20	1.564396444069049773091493015808472813088 46
21	0.210802735500549277375643255705729154360 91

6. おわりに

Taylor展開を利用した数値積分法を提案した。Taylor展開法は、係数を倍精度浮動小数点で表現した場合、通常の倍精度の数値計算と同程度の速さで計算でき、数式処理のように、遅くなることはない。 Taylor展開を利用した数値積分は、局所的な展開を用いるため関数の局所的な性質を捕らえることができるため、鋭いピーク型の関数を比較的に容易に計算できる。逆に、指数関数的に減少する関数では、通常の数値積分では単純に0とおけるような問題でも、１点でのTaylor展開だけでは単純に0とすることができないため非常に時間かかる問題になる。 今回は、特異点を含む問題を扱わなかったが、特異点の扱いについては、他の数値積分公式と同様に扱うことができると思われる[5]ので、それらの計算法を組み込み自動積分公式を作成する予定である。

参考文献

[1] Ellis M. A. and Stroustrup B., The Annotated C++ Reference Manual, Addison-Wesley,1990

[2] Henrici, P., Applied Computational Complex Analysis, Vol. 1, John Wiely & Sons, New York,Chap. 1, 1974

[3] Hirayama H., Numerical Technique for Solving an Ordinary Differrential Equation by Picard's Methods, Integral Methods in Science and Engineering/Editor P. Schiavone, C. Constanda, A. Mioduchowski, Birkhauser, Berlin(2002)

[4] 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

[5] 平山　弘, C++言語によるべき型特異点をもつ関数の数値積分,日本応用数理学会, Vol. 5, No.3, pp. 257-266, 1995

[6] Rall,L. B. , Automatic Differentiation-Technique and Applications, Lecture Notes in Computer Science, Vol.120, Springer-Verlag, Berlin-Heidelberg-New York(1981)

[7] 日比野,長谷川、二宮、細田、佐藤、二宮とFLR法の結合による新しい適応型積分法,情報処理学会論文誌, 44(2003), pp. 2419-2427

[8] 幸谷、鈴木、補外法に基づく数値積分法の実装と性能評価、情報処理学会研究報告、2004-HPC-99、pp.79-84