ハイパフォーマンス
コンピューティング
（２００３．３．１２）
93−19

べき級数展開法による数値積分法

菅家 英和　　平山 弘
神奈川工科大学機械システム工学専攻

Fortran、C++およびC#言語を使うことによって、べき級数の四則演算や関数計算を容易に定義できる。これらの演算と条件文で記述された関数は、容易にべき級数展開できる。　本研究ではこの方法を使うことにより通常の数値積分法と同等の速度で、数値積分を計算することができる。また、通常の数値積分法では求めることが難しいとされる積分区間の近くに特異点を持つ条件の悪い積分も効率的に計算することができることを示す。

Numerical Integration Method by Power Series

Hidekazu Kanke and Hiroshi Hirayama
Kanagawa Institute of Technology

The arithmetic operations and functions of power series can be defined by Fortran, C++ and C# language. The functions which consist of arithmetic operations, pre-defined functions and conditional statements can be expanded in power series. Using this, the numerical integration can be carried out at the same efficiency as other effective numerical integration method. Especially the numerical integration of which the integrand has some singular points near the integration path can be carried out very effective and easily.

1　はじめに

被積分関数をべき級数展開（Taylor 展開）し、その展開式を積分して、積分値を計算する方法は、解析計算では、ごく普通に使われる方法である。被積分関数を Taylor 級数に展開する方法としては、現時点では、係数が厳密な数式処理を使った方法しか使えないため、計算速度が遅くなり計算時間が問題にならない場合を除いて、この方法が使われることはほとんど無かった。

べき級数の係数を浮動小数点で表現することにより高速にべき級数展開 [5] することができる。この手法を利用して被積分関数をべき級数に展開し、その展開式を積分し積分値を求める。べき級数に展開された級数の収束が遅い場合、積分区間を小さく分割しその区間で関数をべき級数展開し、それを積分する。分割した全区間の積分値の総和を求め積分値を計算する。

本論文ではこの方法を使うことにより通常の数値積分法と同等の速度で、数値積分を計算することができる。また、通常の数値積分法では求めることが難しいとされる積分区間の近くに特異点を持つ条件の悪い積分も効率的に計算することができる。

宮広、野田 [6] によって指摘されている、通常の数値積分法 [1]（シンプソン、ロンバーグ、ガウス型、二重指数型、適応型ニュートンコーツ）ではうまく計算できない積分も容易に計算できることを示す。

$$I=\int_0^1 \frac{-1}{x^5-x^4-0.75x^3+x^2-0.25x-10^{-6}}dx \quad (1)$$

このような計算には、関数およびオペレーターのオーバーロードが可能な Fortran90、C++言語 [2]、C#言語が非常に有用である。設計思考

が古い言語（Fortran77、Java 言語、Basic）を使った場合、このような計算は不可能ではないが、これまで、このような計算がこれらの言語を使ってなされなかったことを考慮すると、実際上不可能であると言える。本計算では、C++言語（Borland C++ Builder Ver.6）と Fortran95（富士通製）を使用した。

2 数値積分法

Taylor 級数を使い数値積分する方法を説明する。関数を Taylor 展開するための基本的な考え方やその計算方法については Rall[7]、Henrici[3] や平山 [4] などに述べられている。

次のような積分を考える。

$$I = \int_a^b f(x)dx \tag{2}$$

$f(x)$ を $x = a$ で n 次の Taylor 級数に展開すると

$$f(x) = f_0 + f_1(x-a)^2 + f_3(x-a)^3 + \cdots + f_n(x-a)^n \tag{3}$$

となる。これを積分すると $n+1$ 次の不定積分 $F(x)$ が得られる。

$$F(x) = f_0(x-a) + \frac{f_1}{2}(x-a)^2 + \frac{f_2}{3}(x-3)^3 + \frac{f_3}{4}(x-a)^4 + \cdots + \frac{f_n}{n+1}(x-a)^{n+1} \tag{4}$$

もし $F(x)$ が十分速く収束する級数ならば、$x = b$ を $F(x)$ に代入することによって積分 I が求められる。収束が遅い場合、$c = a + h$ として、$F(c)$ を考える。このとき、$F(c)$ は

$$F(c) = f_0 h + \frac{f_1}{2} + \frac{f_2}{3}h^3 + \frac{f_3}{4}h^4 + \cdots + \frac{f_n}{n+1}h^{n+1} \tag{5}$$

となる。h を十分小さく取れば、$F(c)$ の級数は十分速く収束させることができるので、区間 $[a,b]$ の積分は $F(c)$ となる。区間 $[a,b]$ の積分は $F(c)$ と区間 $[c,b]$ の積分の和となる。

本計算では、要求精度を 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すなわち、

$$\left|\frac{f_n}{n+1}h^{n+1}\right| = \epsilon \tag{6}$$

が成り立つように h を選ぶ。

$$h = \sqrt[n+1]{\frac{(n+1)\epsilon}{|f_n|}} \tag{7}$$

として計算する。この h を積分の区間幅とすれば、積分は問題なく計算できる。この区間幅は、被積分関数がなめらかであるか、または Taylor 級数の次数の高いほど広く取ることができる。f_n がゼロならば上記の式は計算できなくなるが、その場合は次数を 1 次下げて、上記の式を適用する。

残りの積分を計算するために、上と同様な操作を繰り返し、積分区間を小さくし、最終的に全区間の積分を行う。

この計算方法は、

$$I = \int_a^b f(x)dx \tag{8}$$

の場合、微分方程式

$$\frac{dv}{dx} = f(x) \qquad v(a) = 0 \tag{9}$$

の初期値問題を Taylor 展開を利用して解き、$v(b)$ を求める問題に相当する。この計算法は適応型の解放に相当するものである。

Taylor 級数の収束範囲は展開位置から正負両方向きに同じ幅の範囲を持つが、本計算方法では、展開位置から正方向の収束範囲しか使っていない。展開位置を正方向に進めて展開すれば、一回の級数展開でより広い区間を積分することができる。このようにすれば、級数展開の回数を減らすことができる可能性がある。

3 プログラム例

次の積分を求める。

$$\int_0^1 \frac{1}{(1+x^2)}dx \tag{10}$$

```
1: #include "power.h"
2: power f(const power x)
3: {
```

```
 4:     return 1/(1+x*x) ;
 5: }
 6: void main()
 7: {
 8:     double s ;
 9:     power x,y ;
10:     x=power(0,1) ; //x=0 で x を展開
11:     y=f(x);         //x=0 で f を展開
12:     y=integrate(y);// yを積分
13:     s=eval(y,0.3); //積分値を計算
14:     x=power(0.3,1);// x=3 で x を展開
15:     y=f(x);         //x=3 で f を展開
16:     y=integrate(y); // y を積分
17:     s+=eval(y,0.7); // 積分値を計算
18:     cout << s << endl ;
19: }
```

実行結果　0.785398

1 行目では Taylor 級数、積分を定義するヘッダーファイル（power.h）を読み込む。2,3 行目で積分する関数を指定している。10,11 行目で $x=0$ で Taylor 展開し、12 行目で Taylor 展開したものを積分している。13 行目で 0 から 0.3 までの範囲の積分値を計算している。14,15 行目で $x=0.3$ で Taylor 展開し、16 行目で Taylor 展開したものを積分している。17 行目で 0.3 から 1 までの積分値を求めそれに 13 行目に計算した積分値を加えることにより 0 から 1 までの積分値を求めている。このようにして積分値を求めることができる。

4　数値計算例

4.1　簡単な例

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

$$\begin{aligned} I &= \int_0^1 e^x dx \\ &= 1.71828182845904523536028747135 2\cdots \end{aligned}$$

e^x を $x=0$ で 10 次まで、テイラー展開すると

$$1 + x + 0.5x^2 + 0.166667x^3 + 0.041667x^4$$
$$+0.00833333x^5 + 0.00138889x^6$$
$$+0.000198413x^7 + 2.48016 \times 10^{-5}x^8$$
$$+2.775573 \times 10^{-6}x^9 + 2.75573 \times 10^{-7}x^{10}$$

が得られる。この式から $h = 0.452873$ が得られる。ここまでの区間 $[0, 0.452873]$ の積分は 0.572824 となる。$x = 0.452873$ で e^x を 10 次まで Taylor 展開すると

$$1.57282 + 1.57282x + 0.786412x^2$$
$$+0.262137x^3 + 0.0655343x^4$$
$$+0.0131069x^5 + 0.00218448x^6$$
$$+0.000312068x^7 + 3.90085 \times 10^{-5}x^8$$
$$+4.33428 \times 10^{-6}x^9 + 4.33428 \times 10^{-7}x^{10}$$

となる。この式から、$h = 0.432821$ が得られる。この区間 $[0.452873, 0.885694]$ の積分値は 0.572824 となる。$x = 0.885694$ で e^x を 10 次まで Taylor 展開すると

$$2.42467 + 2.42467x + 1.21233x^2$$
$$+0.404111x^3 + 0.101028x^4 + 0.0202056x^5$$
$$+0.00336759x^6 + 0.000481085x^7$$
$$+6.01356 \times 10^{-5}x^8 + 6.68173 \times 10^{-6}x^9$$
$$+6.68173 \times 10^{-7}x^{10}$$

となる。この式から h を計算すると $h = 0.414487$ 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 $h = 0.114306$ となる。これを使うと区間 $[0.885694, 1]$ の積分値は 0.293616 となる。各区間の積分値の総和を求めると積分値 1.7182818284506860 が得られる。

3 区間で分割された区間で展開された Taylor 展開式を積分することによって、不定積分を与えることも容易にできる。

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 $\epsilon = 10^{-14}$ とした本方法の計算時間を測定した。Pentium4 − 2.0GHz を使用したとき、ガウス型数値積分法は 0.00859msec、本方法は 0.00625msec 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よ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 積分区間の近くに特異点がある積分

例題として、積分区間の近くに特異点がある積分を倍精度（10 進数で約１５桁）で計算する。正確な値として、4 倍精度が扱える富士通製 Fortran を使用して、$\epsilon = 10^{-10}$ として、18 次から 20 次の級数を利用して計算した。これらの結果で数値が一致した部分を正確な値として使った。

表１、表２、表３の計算時間は、Pentuim ４ 2.0GHz を使ったときの測定時間である。コンパイラーとして Borland C++ Builder 6.0 を使用した。

4.2.1 例題 1

$$\begin{aligned} I_1 &= \int_{-1}^{2} \frac{5x-1}{x^3-3x-2.001}dx \\ &= 155.779816174584726130150\cdots \end{aligned}$$

この計算を計算精度 $\epsilon = 10^{-10}$ として、いろいろな次数でこの積分を計算した。その結果を表 1 に示す。3 次の級数を使って計算する場合が最も時間がかかり、級数の次数が上がると急激に計算時間が減少し、10 次以降ゆっくり減少する。プログラムは非常に簡単で、次のような関数を準備するだけで容易に計算できる。

```
power func( const power& x )
{
  return (5*x-1)/(x*(x*x-3)-2.001);
}
```

通常の関数との違いは、宣言部分だけであり、これまで、C 言語や C++言語を使ったことがある人ならば容易に作ることができる。

4.2.2 例題 2

$$\begin{aligned} I_2 &= \int_{0}^{1} \frac{-1}{x^5-x^4-0.75x^3+x^2-0.25x-10^{-6}}dx \\ &= 5195.2449734453507030173\cdots \end{aligned}$$

この計算を計算精度 $\epsilon = 10^{-10}$ として、いろいろな次数でこの積分を計算した。その結果を表 2 に示す。この積分は、通常の積分法でうまく計算できない例である。例題 1 と同じように、３次の級数を使って計算する場合が最も時間がかかり、級数の次数が上がると急激に計算時間が減少し、10 次以降ゆっくり減少する。この問題は、区間分割数がほかの例題より多くなり、数値積分の計算問題として難しいものであることが分かる。この積分をするために準備しなければならないプログラムも容易に作成でき、以下のようになる。

```
power func( const power& x )
{
   return -1.0/(((((x-1.0)*x-0.75)*x
       +1.0)*x-0.25)*x-1.0e-6) ;
}
```

前の例題と同じように、通常の関数との違いは、宣言部分だけである。

4.2.3 例題 3

$$\begin{aligned} I_3 &= \int_{0}^{1} \frac{e^{2x}(1.4e^x-10)^2}{e^x+2}\sqrt[3]{\frac{7.8e^x}{e^x-0.9}}dx \\ &= 115.07047409178540851986 87\cdots \end{aligned}$$

この計算を計算精度 $\epsilon = 10^{-10}$ として、いろいろな次数でこの積分を計算した。その結果を表 3 に示す。上の二つの例と同様に、3 次の級数を使って計算する場合が最も時間がかかり、級数の次数が上がると急激に計算時間が減少し、10 次以降ゆっくり減少する。この計算は、通

常の数値計算と同じように、e^x の計算は 1 回で済むので比較的簡単に計算できる。以下にそのプログラムを示す。この問題は、20 次程度の Taylor 級数を使うと、分割数が 8 とかなり小さな値になる。ここで扱った三つの例題の中では、最も容易な問題であることがわかる。この積分を計算するために準備しなければならないプログラムは、以下のようなものであり、簡単に記述できる。

```
power func( const power& x )
{
   power ex = exp(x) ;
   power e10 = 1.4*ex-10 ;
   return (ex*ex*e10*e10)/(ex+2)*
     pow(7.8*ex/(ex-0.9),1.0/3) ;
}
```

宮広、野田 [6] の論文では、例題 1 や 2 のように被積分関数が有理関数の積分法について論じている。そこでは、被積分関数の分母を数値的に因数分解し、その結果を利用して、数値的に部分分数に分解し、それを積分する方法を提案している。例題 1 や 2 の問題のように、通常の積分法では難しい積分も計算できるなどの利点もあるが被積分関数が有理関数に限られ、分母の次数が大きいとき、分母を因数分解するのが難しいなどの問題点がある。

本方法では、有理関数だけでなく、ほとんど制限なく計算可能である。例題 3 は変数変換によって有理関数に変換することができるが、本方法を使う場合、変数変換も不要であり、容易に適用可能である。

表 1：積分 I_1 の計算誤差

次数	分割表	相対誤差	時間 (msec)
3	39049	6.93e-12	121.125
4	4992	3.52e-12	16.359
5	1462	4.37e-11	4.941
6	657	5.74e-12	2.381
7	370	7.96e-13	1.374
8	242	9.39e-12	0.938
9	174	6.93e-12	0.702
10	133	7.28e-12	0.557
11	107	1.87e-11	0.454
12	90	2.62e-12	0.400
13	77	7.82e-12	0.351
14	67	8.61e-12	0.321
15	60	1.85e-12	0.294
16	55	2.38e-12	0.282
17	50	1.07e-12	0.263
18	46	7.42e-12	0.244
19	43	8.48e-12	0.236
20	40	2.65e-12	0.227

表 2：積分 I_2 の計算誤差

次数	分割表	相対誤差	時間 (msec)
3	180991	1.71e-12	734.500
4	18655	3.90e-13	82.000
5	4958	2.63e-13	23.188
6	2063	4.77e-13	10.008
7	1117	9.07e-13	5.617
8	705	4.49e-12	3.691
9	497	1.90e-13	2.686
10	375	4.10e-13	2.105
11	299	1.47e-12	1.693
12	247	2.91e-13	1.434
13	211	3.25e-13	1.267
14	183	2.08e-14	1.130
15	163	1.57e-12	1.038
16	147	7.72e-14	0.977
17	134	1.25e-12	0.900
18	124	2.09e-12	0.862
19	115	2.42e-12	0.824
20	108	1.08e-12	0.778

表 3：積分 I_3 の計算誤差

次数	分割表	相対誤差	時間 (msec)
3	6574	4.86e-10	43.469
4	870	8.17e-13	6.652
5	210	5.79e-12	1.831
6	97	2.87e-12	0.969
7	57	1.31e-12	0.641
8	37	2.69e-12	0.473
9	27	1.52e-12	0.382
10	21	1.67e-12	0.328
11	18	1.65e-12	0.313
12	15	1.62e-12	0.290
13	13	1.59e-12	0.275
14	12	1.62e-12	0.279
15	11	1.62e-12	0.275
16	10	1.58e-12	0.271
17	9	1.52e-12	0.267
18	9	1.60e-12	0.290
19	8	1.49e-12	0.279
20	8	1.54e-12	0.294

5　まとめ

本論文で示した Taylor 級数法は、手計算法としては昔から使われてた方法である。これを計算機に乗せることによって、非常に強力な数値積分法が得られることを示した。　積分区間の近くに特異点がある積分のような性質のよくない関数だけでなく、普通の問題も効率的に計算できる。計算時間は、ガウス型積分公式と同じ程度の時間である。

理論は非常に単純であり、わかりやすい方法なのでいろいろな問題にたいして容易に利用することができる。

Taylor 級数は、展開位置から正負両方向に同じ大きさの区間で収束する。本方法は、正方向の収束区間だけしか利用していない。展開位置を正方向に移動させ級数展開し、1 回の級数展開でより広い区間で積分することができるので、計算速度を上げることが期待できる。性質のよい関数の場合、収束区間の幅はあまり変化しないと推定できるので、展開位置を直前の収束区間幅だけ進めれば能率的にできる。

参考文献

[1] P.J.Davis P.Rabinwitz(森 正武訳)、計算機による数値積分法、日本コンピューター、1981

[2] Ellis.A.and Stroustrup B.,The Annotated C++ Reference Manual, Addision-Wesley,1990

[3] Henrici P.,Applied and Computational Complex Analysis, Vol.1, Chap.1, John Wiley & Sons, New York,1974

[4] 平山 弘、C++言語によるべき型特異点を持つ関数の数値積分、日本応用数理学会論文誌、vol.5, pp. 257-266 (1995)

[5] 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、vol.12, pp. 1-8(2002)

[6] 宮広、野田、新しい有理関数近似によるハイブリッド積分の拡張について、日本応用数理学会学会論文誌、Vol. 2, pp. 193-206(1992)

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