数値積分 (2)

桂田 祐史

2016 年 5 月 11 日, 2016 年 6 月 4 日

0　今日のお話は

前回 (脱線して少しのんびりし過ぎた) 紹介した**複合 Simpson 公式** (今日は面倒なので、「複合」を省略して単に Simpson 公式と呼ぶ) は、歴史上良く利用されたそうである [1]。(だからこそ、高校数学に登場したり、現代でも大学の微積分の授業で紹介すべきと言う人がいたりする。— 個人的にはこの意見にあまり賛成しないけれど。)

他に直交多項式に基づく **Gauss 型積分公式**というのがある。これは Simpson 公式よりもさらに優秀であるけれど、ここでは省略する [2]。

実は、1970 年頃、日本で大きなブレークスルーがあり、優秀な積分公式が発見された。今ではそれがスタンダードになりつつある (そうなるのに長い時間がかかったけれど)。その話を紹介したい。

1　クラシックな公式を試してみよう

$I = \displaystyle\int_a^b f(x)\,dx$ を計算することが目標である。

1.1　前回紹介した公式のまとめ

1.1.1　(複合) 中点公式

区間 $[a,b]$ を N 等分して、各小区間 $[a_j, b_j]$ $(j = 1, \ldots, N)$ で、区間の中点での関数の値 $f(a_j + h/2)$ を用いて 0 次関数補間して、それを積分する (長方形の面積の和)。

$$M_N := h \sum_{j=1}^{N} f\left(a + (j - 1/2)\, h\right), \quad h := \frac{b-a}{N}.$$

[1] 数表は、等間隔標本点における関数値が用意されているので、それを使って計算する場合は、Simpson 公式が最有力というのは納得できる。Gauss 型積分公式にしても、DE 公式にしても、等間隔でない標本点における関数値が必要なので、数表は利用しにくい。

[2] 余談になるけれど、直交多項式や、連立 1 次方程式に対する CG 法 (共役勾配法)　など、直交性が重要な役割を果たす話が多いので、何らかの形で講義をすべきであると考えるのだけど、一体どういう機会にやれば良いのだろう。

(複合) 中点公式

```
h = (b - a) / N;
M = 0;
for (j = 1; j <= N; j++)
  M += f(a + (j - 1.0 / 2) * h);
M *= h;
```

1.1.2 (複合) 台形公式

区間 $[a,b]$ を N 等分して、各小区間 $[a_j,b_j]$ $(j=1,\ldots,N)$ で、区間の端点での関数の値 $f(a_j)$, $f(b_j)$ で、1 次関数補間して、それを積分したもの (台形の面積) の和を作る。

$$T_N := h\left(\frac{1}{2}f(a)+\sum_{j=1}^{N-1} f\left(a+jh\right)+\frac{1}{2}f(b)\right), \quad h:=\frac{b-a}{N}.$$

(複合) 台形公式

```
h = (b - a) / N;
T = (f(a) + f(b)) / 2;
for (j = 1; j < N; j++)
  T += f(a + j * h);
T *= h;
```

1.1.3 (複合)Simpson 公式

区間 $[a,b]$ を m 等分して、各小区間 $[a_j,b_j]$ $(j=1,\ldots,m)$ をさらに 2 等分して、区間の 2 つの端点 a_j,b_j と中点 $\dfrac{a_j+b_j}{2}$ の 3 点での関数値を使って、2 次関数補間して、それを積分したもの $(\dfrac{h}{3}\left(f(a_j)+4f\left(\frac{a_j+b_j}{2}\right)+f(b_j)\right))$ の和を作る。

$$S_{2m} := \frac{h}{3}\left(f(a)+2\sum_{j=1}^{m-1} f\left(a+2jh\right)+4\sum_{j=1}^{m} f\left(a+(2j-1)h\right)+f(b)\right), \quad h:=\frac{b-a}{2m}.$$

実は $S_{2m}=(T_m+2M_m)/3$ という関係がある。

1.1.4 C 言語によるプログラム

本日 (2016/5/11) の多くのプログラムは、次のコードを用いている。

```
/*
 * nc.c --- Newton-Cotes の積分公式: 複合台形公式, 複合中点公式, 複合 Simpson 公式
 */

typedef double ddfunction(double);

double midpoint(ddfunction, double, double, int);
double trapezoidal(ddfunction, double, double, int);
double simpson(ddfunction, double, double, int);
```

```
/* 関数 f の [a,b] における積分の複合中点公式による数値積分 M_N */
double midpoint(ddfunction f, double a, double b, int N)
{
  int j;
  double h, M;
  h = (b - a) / N;
  M = 0.0;
  for (j = 1; j <= N; j++) M += f(a + (j - 0.5) * h);
  M *= h;
  return M;
}

/* 関数 f の [a,b] における積分の複合台形公式による数値積分 T_N */
double trapezoidal(ddfunction f, double a, double b, int N)
{
  int j;
  double h, T;
  h = (b - a)/ N;
  T = (f(a) + f(b)) / 2;
  for (j = 1; j < N; j++) T += f(a + j * h);
  T *= h;
  return T;
}

/* 関数 f の [a,b] における積分の複合 Simpson 公式による数値積分 S_{N} */
double simpson(ddfunction f, double a, double b, int N)
{
  int m = N / 2;
  return (trapezoidal(f, a, b, m) + 2 * midpoint(f, a, b, m)) / 3;
}
```

1.2 公式の良し悪しを試す

I_N をどれかの積分公式とする ($I_N = M_N, T_N, S_N$ 等)。

$$E_N := I - I_N$$

を誤差と呼ぶ。

上で紹介した M_N, T_N, S_N は、多項式での補間を基礎としているので、多項式関数に対しては $E_N = 0$ となることがありうる (後で確認する)。しかし一般の関数に対しては、$E_N = 0$ が成り立つとは期待できない。

M_N, T_N, S_N はそれぞれ、0 次補間, 1 次補間, 2 次補間であるから、0 次関数, 1 次関数, 2 次関数までは誤差が 0 になるはずである。

例 1.1 (f が多項式関数の場合)

$$I = \int_0^1 f(x)dx.$$

$f(x) = 1$, $f(x) = 1 + 2x$, $f(x) = 1 + 2x + 3x^2$, $f(x) = 1 + 2x + 3x^2 + 4x^3$, $f(x) = 1 + 2x + 3x^2 + 4x^3 + 5x^4$ という 5 つの場合 (それぞれ $I = 1, 2, 3, 4, 5$ となる) に、I を中点公式、台形公式、Simpson 公式で計算して、誤差を求めてみる。

```
$ ls
```

example1.c, nc.c があることを確認する。以下、コンパイルして実行する。

```
$ cc -o example1 example1.c

$ ./example1
次数=0
N=2
中点公式     M=   1.000000000000000, 誤差=0.000000e+00
台形公式     T=   1.000000000000000, 誤差=0.000000e+00
Simpson 公式 S=   1.000000000000000, 誤差=0.000000e+00

$ ./example1
次数=1
N=2
中点公式     M=   2.000000000000000, 誤差=0.000000e+00
台形公式     T=   2.000000000000000, 誤差=0.000000e+00
Simpson 公式 S=   2.000000000000000, 誤差=0.000000e+00

$ ./example1
次数=2
N=2
中点公式     M=   2.937500000000000, 誤差=6.250000e-02
台形公式     T=   3.125000000000000, 誤差=-1.250000e-01
Simpson 公式 S=   3.000000000000000, 誤差=0.000000e+00

$ ./example1
次数=3
N=2
中点公式     M=   3.812500000000000, 誤差=1.875000e-01
台形公式     T=   4.375000000000000, 誤差=-3.750000e-01
Simpson 公式 S=   4.000000000000000, 誤差=0.000000e+00

$ ./example1
次数=4
N=2
中点公式     M=   4.613281250000000, 誤差=3.867188e-01
台形公式     T=   5.781250000000000, 誤差=-7.812500e-01
Simpson 公式 S=   5.041666666666667, 誤差=-4.166667e-02
```

予想通り、中点公式は 0 次関数に対して、台形公式は 1 次以下の関数に対して、Simpson 公式は 2 次以下の関数に対して、誤差が 0 になる。

しかし、その予想を超えて、中点公式は 1 次以下の関数に対して、Simpson 公式は 3 次以下の関数に対して、誤差が 0 になる。■

事実 1

M_N, T_N は 1 次以下の多項式関数に対して、S_N は 3 次以下の多項式関数に対して、$E_N = 0$ となる。

M_N, S_N が予想よりも 1 次分良くなっている。

多項式関数でない場合、$E_N = 0$ は期待できないが、実は、$\lim_{N\to\infty} E_N = 0$ は成り立つ。

E_N がどの程度の大きさになるかを比較してみよう。多くの場合に、ある r が存在して、

$$E_N = O\left(\frac{1}{N^r}\right) \quad (N \to \infty)$$

が成り立つ。r が大きいと、収束が速く、優れた公式である、ということになる。これを確認するには、N を変化させたときの E_N を求め、両側対数目盛りでプロットすると良い。

例 1.2 (多項式関数ではないが、滑らかな関数の場合) $\int_0^1 e^x\,dx$, $\int_1^2 \frac{1}{x}\,dx$ など。積分の計算部分は上のプログラムと同じで済む。数表を出力して、gnuplot でグラフ化するために、N に関するループを作る。

まず、プログラムの確認をしてみる。

```
$ cat example2.c
```

(中身はこの文書では省略する。)

続いてコンパイルして実行する。

```
$ cc -o example2 example2.c
$ ./example2 > ex2.data
```

計算結果と、gnuplot のプログラム example2.gp の確認をする。

```
$ cat ex2.data
#   N       I-M_N           I-T_N         I-S_N
    2    1.776911e-02  -3.564926e-02  -5.793234e-04
    4    4.466549e-03  -8.940076e-03  -3.701346e-05
    8    1.118163e-03  -2.236764e-03  -2.326241e-06
   16    2.796364e-04  -5.593001e-04  -1.455928e-07
   32    6.991508e-05  -1.398319e-04  -9.102727e-09
   64    1.747914e-05  -3.495839e-05  -5.689695e-10
  128    4.369809e-06  -8.739624e-06  -3.556155e-11
  256    1.092454e-06  -2.184908e-06  -2.222444e-12
  512    2.731135e-07  -5.462270e-07  -1.383338e-13
 1024    6.827838e-08  -1.365568e-07  -9.547918e-15
 2048    1.706960e-08  -3.413919e-08  -4.440892e-16
 4096    4.267396e-09  -8.534800e-09  -6.661338e-16
 8192    1.066844e-09  -2.133694e-09  -2.442491e-15
16384    2.667191e-10  -5.334184e-10  -2.220446e-15
32768    6.667733e-11  -1.333611e-10   6.661338e-15
65536    1.668576e-11  -3.332978e-11  -2.220446e-15
$ cat example2.gp
set logscale xy
set format y "10^{%L}"
set format x "10^{%L}"
plot "ex2.data" using ($1):(abs($2)) with linespoints, \
     "ex2.data" using ($1):(abs($3)) with linespoints, \
     "ex2.data" using ($1):(abs($4)) with linespoints
set term postscript eps color
set output "ex2.eps"
replot
```

gnuplot のプログラムを実行する。

```
$ gnuplot example2.gp
```

図 1 を見ると、プロットした点が直線上に乗り、それら直線の傾きがそれぞれ $-2, -2, -4$ であることが分かる。 ■

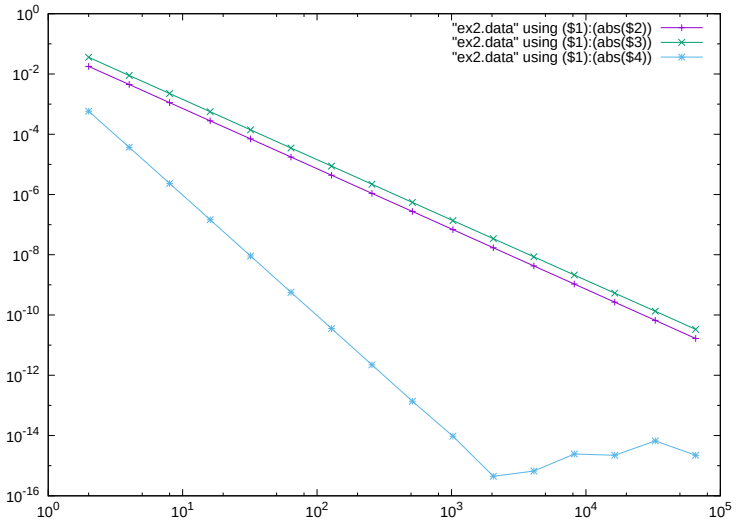

図 1: $I = \int_0^1 e^x\,dx$ を中点公式、台形公式、Simpson 公式で計算したときの誤差

事実 2 (1,2,3 でなく、2,2,4)

滑らかな関数に対して、

$$I - M_N = O\left(\frac{1}{N^2}\right), \quad I - T_N = O\left(\frac{1}{N^2}\right), \quad I - S_N = O\left(\frac{1}{N^4}\right).$$

高次の公式の方が収束は速い or 等しい。必ず速いわけでなく、等しい場合もある (台形公式が中点公式より優れているわけではない)。

試してみよう　自分で $[a,b]$, f を選び、プログラムに必要な修正を施して計算してみよう。(誤差を知りたいので、$I = \int_a^b f(x)\,dx$ の値が分かるものを試すのが良い。) ■

余談 1.1 関数の 3 次関数補間に基づく、Simpson $\frac{3}{8}$ 公式というものがある。その場合は、3 次以下の多項式関数に対して、$E_N = 0$, 一般の滑らかな関数に対して $E_N = O\left(\frac{1}{N^4}\right)$ である。中点公式、台形公式、Simpson 公式、Simpson $\frac{3}{8}$ 公式の順に $r = 2, 2, 4, 4$. (1, 2, 3, 4 ではない！)■

1.3　一般論の守備範囲外の話

1.3.1　うまく行かない話

滑らかでない関数の場合はどうなるか、見てみよう。

例 1.3 (滑らかでない関数の場合) 2 つの問題 (a), (b) を考えよう。

(a) まず区間の内部では滑らかであるが、区間の端点 ($x = 1$) で連続ではあるが、微分可能でない

$$I = \int_0^1 \sqrt{1 - x^2}\,dx \quad \left(= \frac{\pi}{4}\right).$$

プログラムは `example2.c` をコピーして少し修正するだけなので、`example3a.c` を作るのは各自の演習とする (分からなければ質問して下さい)。

```
$ cc -o example3a example3a.c
$ ./example3a > ex3a.data
$ cat ex3a.data
#    N       I-M_N           I-T_N          I-S_N
     2   -2.944367e-02   1.023855e-01   4.138123e-02
     4   -1.058414e-02   3.647090e-02   1.449937e-02
     8   -3.773569e-03   1.294338e-02   5.100871e-03
    16   -1.339789e-03   4.584904e-03   1.798746e-03
    32   -4.746873e-04   1.622558e-03   6.351089e-04
    64   -1.680046e-04   5.739352e-04   2.243944e-04
   128   -5.942998e-05   2.029653e-04   7.930866e-05
   256   -2.101722e-05   7.176766e-05   2.803511e-05
   512   -7.431692e-06   2.537522e-05   9.911071e-06
  1024   -2.627674e-06   8.971763e-06   3.503944e-06
  2048   -9.290536e-07   3.172045e-06   1.238805e-06
  4096   -3.284755e-07   1.121496e-06   4.379792e-07
  8192   -1.161346e-07   3.965100e-07   1.548482e-07
 16384   -4.105994e-08   1.401877e-07   5.474696e-08
 32768   -1.451691e-08   4.956389e-08   1.935595e-08
 65536   -5.132508e-09   1.752349e-08   6.843354e-09
$
```

図 2 を見ると、一応誤差は減少するが、その速さがこれまでより遅く、また Simpson 公式の速

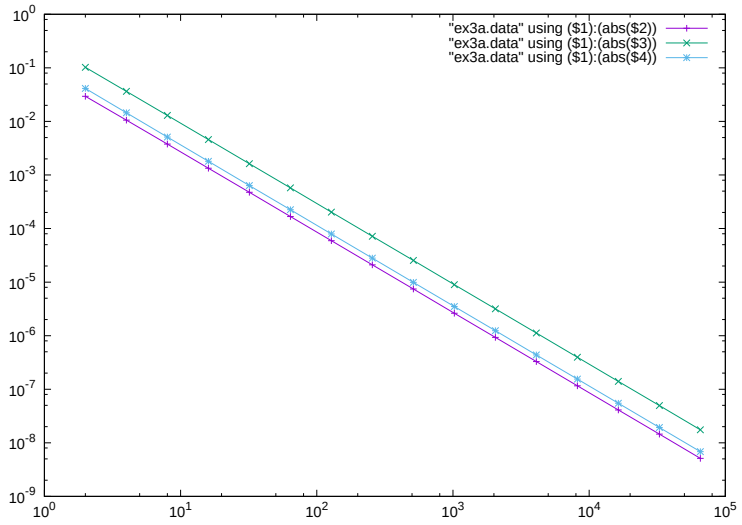

図 2: $I = \displaystyle\int_0^1 \sqrt{1-x^2}\,dx$ を中点公式、台形公式、Simpson 公式で計算したときの誤差

さが中点公式、台形公式と変わらないことに気付く。

(b) 次に

$$I = \int_0^1 \frac{1}{\sqrt{1-x^2}}\,dx \quad \left(= \frac{\pi}{2}\right)$$

を考えよう。これも `example3a.c` をコピーして少し修正するだけでプログラムが作れる。実行すると

```
$ cc -o example3b example3b.c
$ ./example3b
#   N       I-M_N           I-T_N          I-S_N
    2    2.984696e-01          -inf           -inf
    4    2.124860e-01          -inf           -inf
    8    1.507431e-01          -inf           -inf
   16    1.067627e-01          -inf           -inf
   32    7.555268e-02          -inf           -inf
   64    5.344494e-02          -inf           -inf
  128    3.779873e-02          -inf           -inf
  256    2.673037e-02          -inf           -inf
  512    1.890215e-02          -inf           -inf
 1024    1.336617e-02          -inf           -inf
 2048    9.451425e-03          -inf           -inf
 4096    6.683208e-03          -inf           -inf
 8192    4.725756e-03          -inf           -inf
16384    3.341619e-03          -inf           -inf
32768    2.362884e-03          -inf           -inf
65536    1.670812e-03          -inf           -inf
$
```

結果がおかしい？実はこの積分の被積分関数 $\dfrac{1}{\sqrt{1-x^2}}$ は、区間の内部では滑らかであるが、区間の端点 $x=1$ では連続でない (というか定義域に入っていない)。台形公式、Simpson 公式では、$f(a), f(b)$ が必要なので、計算が出来ない。中点公式では端点での値 $f(a), f(b)$ が必要ないので、計算は出来るが、図 4 を見ても、誤差の減少が非常にゆっくりであることが分かる。 ■

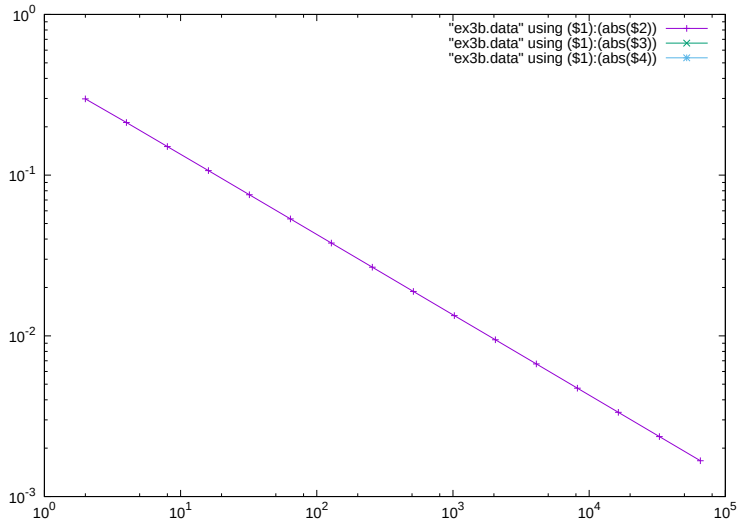

図 3: $I=\displaystyle\int_0^1 \frac{1}{\sqrt{1-x^2}}\,dx$ を中点公式で計算したときの誤差

事実 3

連続であるが、滑らかなでない関数に対しては、収束はしても収束は遅い。高次の公式の優位性はない。
連続でない関数に対しては、そもそも積分公式が適用不可能なこともありうる。

以前、プログラミング入門のような本で、$I=\displaystyle\int_0^1 \sqrt{1-x^2}\,dx$ に対して Simpson 公式で計算して、ちゃんと $\pi/4$ の近似値が得られたと説明しているものに遭遇したことがある。ちょっとがっかりした (その著者はプログラミングは得意なのかもしれないけれど、数値計算には詳しくないということだ)。

以上、何を無茶なことをしているのかと思うかもしれないけれど、後でサプライズがある。

試してみよう　自分で適当な問題を選び ($\sqrt{\ }$ でもっとシンプルなものとか)、プログラムに必要な修正を施して計算してみよう。 ■

1.3.2　うまく行く話

一方で、不思議なくらい、非常にうまく行く場合があることが知られている。それを 2 つ紹介する。

まず、滑らかな周期関数の 1 周期の積分について。つまり $f\colon \mathbb{R} \to \mathbb{C}$ は滑らかで、ある正数 T に対して、$f(x+T) = f(x)$ $(x \in \mathbb{R})$ が成り立つとき、

$$I = \int_a^b f(x)\,dx, \quad b - a = T$$

を計算する場合。$f(a) = f(b)$ であるので、

$$T_N = h\left(\frac{1}{2}f(a) + \sum_{j=1}^{N-1} f(a+jh) + \frac{1}{2}f(b)\right) = h\sum_{j=1}^{N} f(a+jh) = h\sum_{j=0}^{N-1} f(a+jh)$$

であることに注意する。

例 1.4 (滑らかな周期関数の 1 周期の積分) $I = \displaystyle\int_0^{2\pi} \frac{dx}{2+\cos x} = \frac{2\pi}{\sqrt{3}}$. もう少し複雑なものが欲しければ、Bessel 関数の積分表示[3]、振り子の周期の計算など。

```
$ cat example4.c
$ cc -o example4 example4.c
$ ./example4 > ex4.data
$ cat ex4.data
  (結果はこの文書では省略)
$ gnuplot ex4.data
```

$N = 16$ の段階で、台形公式の誤差 $|I - T_N| \simeq 5.1 \times 10^{-16}$. ほぼ使用している処理系の最高精度に到達している[4]。 ■

ここまで順番に例を見ておいてもらえると、この例の結果が異常に思えるくらい良いことが分かるであろう。

事実 4

滑らかな周期関数の 1 周期の積分を台形公式で計算すると、小さい N でも高精度な値が得られる。

余談 1.2 (某月某日の質問対応) 某授業で単振り子の周期の計算をする必要が生じて、台形公式で、N も小さくしたにもかかわらず、高精度の値が得られたけれど、一体なぜ？という質問をしに来た学生がいた。答はピンポイントでこの事実 4 である。 ■

(なお、周期関数の 1 周期積分については、中点公式は本質的に台形公式と同じであるから、やはり小さい N でも高精度な値が得られる。一方、S_N は T_N, M_N より一歩遅れた結果を与える。)

[3] $J_n(x) = \dfrac{1}{2\pi}\displaystyle\int_0^{2\pi} \cos(nt - x\sin t)\,dt.$

[4] 最近のパソコンの C 言語処理系では、浮動小数点数は、IEEE 754 という規格に従っている。double 型のデータの内部表現は、仮数部が 2 進法で 53 桁で、10 進法に換算すると、16 桁弱に相当する。

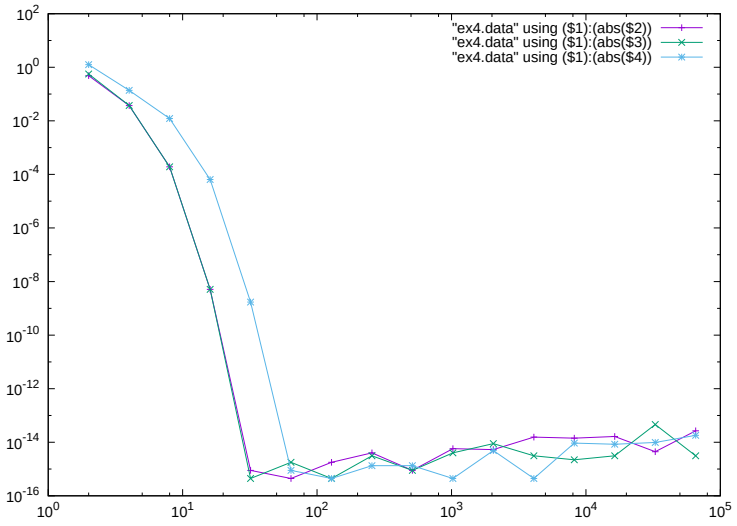

図 4: $I = \displaystyle\int_0^{2\pi} \frac{1}{2+\cos x}\,dx$ を中点公式, 台形公式, Simpson 公式で計算したときの誤差

試してみよう　自分で適当な問題を選び、プログラムに必要な修正を施して計算してみよう。■

もう 1 つ台形公式が良好な結果を生む例を紹介する。

例 1.5 (遠方で速く減衰する関数の $\mathbb{R}$ 全体での積分) 有名な

$$I = \int_{-\infty}^{\infty} e^{-x^2}\,dx \quad \left(=\sqrt{\pi}\right)$$

を、適当な $h > 0$, $N \in \mathbb{N}$ を選んで、

$$I_N := h \sum_{j=-N}^{N} f(jh)$$

で近似してみる。無限区間であるからこれまでとは異なるが、これも台形公式と呼ばれる。

$h = 1$, $N = 6$; $h = 1/2$, $N = 12$; $h = 1/4$, $N = 24$ として実行した (N は $-6 \le x \le 6$ の範囲までと考えて定めた。)。

```
$ cc -o example5 example5.c
$ ./example5
 N      h                   I                 I-T
 6      1          1.772453850905516  -1.833539e-04
12      0.5        1.772453850905516  -2.220446e-16
24      0.25       1.772453850905516  -4.440892e-16
```

$h = 1/2$ という粗い分割で ($2N + 1 = 25$ 点での値を用いて)、ほぼ使用している処理系の最高精度に到達している。■

2　1970 年前後 (歴史メモ)

まだ「歴史」というほど古くはないかもしれないけれど、自分が見て来たわけではないので (さすがに私もその頃は小学生) …

§§1.3.2 のような例を知って、面白いと感じても、でも特殊例に過ぎない、と切り捨ててしまう人が多いと思われるが、非凡な人は見逃さない。

1970 年、伊理正夫、森口繁一、高澤嘉光により、後年 **IMT 公式** と呼ばれる積分公式が提唱された ([1], [2])。これは積分を変数変換によって、滑らかな周期関数の 1 周期積分に変換して、それに対して台形公式を適用する、という考えに基づく。

(一松 [3] によると、「日本において最初に発見された他の多くの重要な業績と同様に，日本において順調に発展したとはいい難く，外国で有名になり，後年になって結果的に日本に逆輸入されるという経過をたどっているのは、残念なことと思う.」)

IMT 公式は非常に高性能であるが、それをヒントにして、さらに高性能な **DE 公式** (double exponential formula, 二重指数関数型数値積分公式) が高橋秀俊と森正武により提唱された ([4], [5] が報告され、[9], [6] が出版された)。

DE 公式は、ほぼ最適の公式であると考えられている (それを裏付ける数学的証拠がある)。

IMT 公式も、DE 公式も、Mathematica の `NIntegrate[]` で利用されている。

以下は雑談。

比較的近年であることから、発表当時の情報が得やすい。京都大学数理解析研究所で行われた研究集会での発表は、京都大学数理解析研究所講究録[5] の形で残っていて、フリーにアクセス出来る (正式な論文でないが、日本語で書かれているのも、まだ英語に不慣れな学生には嬉しいかも？もしかすると書く方にとっても、細かいニュアンスが書きやすいかもしれない…)。

伊理・森口・高澤 [1] の発表があった研究集会 (「科学計算基本ライブラリーのアルゴリズムの研究会」, 1969/11/5〜1969/11/07, 於 京都大学数理解析研究所[6]) で、直後の発表が高橋・森 [7] であった (後に [8] が出版される)。これは数値積分を複素関数論の手法で誤差解析する手法に関するものであった。

参考文献

[1] 伊理正夫, 森口繁一, 高澤嘉光：ある数値積分公式について, 京都大学数理解析研究所講究録, Vol. 91, pp. 82–119 (1970), http://www.kurims.kyoto-u.ac.jp/~kyodo/kokyuroku/contents/pdf/0091-03.pdf.

[2] Iri, M., Moriguti, S. and Takasawa, Y.: On a certain quadrature formula, *J. Comput. Appl. Math*, Vol. 17, pp. 3–20 (1987).

[3] 一松信（ひとつまつしん）：留数解析 — 留数による定積分と級数の計算, 共立出版 (1979), 第 5 章は高橋-森理論の解説。

[4] 高橋秀俊, 森正武：変数変換によって得られる積分公式, 数理解析研究所講究録, Vol. 149, pp. 93–110 (1972), http://www.kurims.kyoto-u.ac.jp/~kyodo/kokyuroku/contents/pdf/0149-07.pdf.

[5] 高橋秀俊, 森正武:変数変換によって得られる積分公式 (2), 数理解析研究所講究録, Vol. 172, pp. 88–104 (1973), http://www.kurims.kyoto-u.ac.jp/~kyodo/kokyuroku/contents/pdf/0172-06.pdf.

[6] Takahashi, H. and Mori, M.: Double exponential formulas for numerical integration, *Publ. RIMS Kyoto Univ.*, Vol. 9, pp. 721–741 (1974).

[5] http://www.kurims.kyoto-u.ac.jp/~kyodo/kokyuroku/kokyuroku.html
[6] http://www.kurims.kyoto-u.ac.jp/~kyodo/kokyuroku/contents/91.html

[7] 高橋秀俊, 森正武：解析関数の数値積分の誤差の新しい評価法, 数理解析研究所講究録, Vol. 91, pp. 119–141 (1970), http://www.kurims.kyoto-u.ac.jp/~kyodo/kokyuroku/contents/pdf/0091-04.pdf.

[8] Takahasi, H. and Mori, M.: Error estimation in the numerical integration of analytic functions, *Rep. Comput. Centre Univ. Tokyo*, Vol. 3, pp. 41–108 (1970).

[9] Takahasi, H. and Mori, M.: Quadrature Formulas Obtained by Variable Transformation, *Numerische Mathematik*, Vol. 21, pp. 206–219 (1973).