数値積分解説 (2017年度)

桂田 祐史

2017年7月12日, 2017 年7 月24 日

数値積分については、2016 年度に応用複素関数で講義した時のノート (http://nalab.mind.meiji.ac.jp/~mk/lecture/applied-complex-function-2016/numerical-integration.pdf) があるが、整理不十分で分量も多すぎるので (4 回分の錯綜した講義のノート)、講義用に短い文書を用意し直すことにした (古いものも整理し直していつか復活させたい。こちらの付録にマージしていく形でそうするかもしれない。)。

目 次

1 はじめに

定積分の値を数値計算で求めることを**数値積分**という。

微分の計算はある意味で簡単 (導関数が分かっている関数を組み合わせて作った関数の導関数は求められる) であるが、積分の計算は難しいことが多い。

そこで定積分の値を数値計算で求めることが必要になる。この文書では 1 変数関数の積分

$$I(f) = \int_a^b f(x)\,dx \tag{1}$$

の値を求める方法について論じる。

特別の f に対して $I(f)$ を計算するのではなく、ある程度広い範囲の f について、共通のやり方で $I(f)$ を計算する方法を考察する。

応用上現れる近似公式 (数値積分公式) は 、ほとんどが次の形をしている。

$$I_n(f) = \sum_{k=1}^{n} A_k f(x_k). \tag{2}$$

ここで x_k は $[a,b]$ 内から選んだ相異なる点で、**標本点** (sample point) と呼ばれる。また A_k は**重み** (weight) と呼ばれる。

この文書で取り上げる

(a) **補間型数値積分公式**

(b) **二重指数関数型数値積分公式** (double exponential formula)

はいずれも (2) の形をしている。

2 補間型数値積分公式

被積分関数 f の補間多項式 $f_n(x)$ を求めて、その積分 $I(f_n)$ の値を $I(f)$ の近似値に採用するが補間型数値積分公式である。

2.1 補間多項式

2.1.1 定義と一意存在

命題 2.1 (補間多項式の一意存在) $[a,b]$ は $\mathbb{R}$ の有界閉区間、$x_1,\dots,x_n$ は $[a,b]$ 内の相異なる点、$f\colon [a,b]\to\mathbb{R}$ とするとき、

$$\deg f_n(x)\le n-1,\quad f_n(x_k)=f(x_k)\quad (k=1,\dots,n \tag{3}$$

を満たす実係数多項式 $f_n(x)$ が一意的に存在する。

証明　学ぶ価値のある証明が色々あるが、ここでは構成的な証明を採用する。$k=1,\cdots,n$ に対して、

$$L_k^{(n-1)}(x):=\frac{\prod_{\substack{1\le j\le n\\ j\neq k}}(x-x_j)}{\prod_{\substack{1\le j\le n\\ j\neq k}}(x_k-x_j)}=\frac{(x-x_1)\cdots(x-x_{k-1})(x-x_{k+1})\cdots(x-x_n)}{(x_k-x_1)\cdots(x_k-x_{k-1})(x_k-x_{k+1})\cdots(x_k-x_n)} \tag{4}$$

で $L_k^{(n-1)}(x)$ を定める。

$$L_k^{(n-1)}(x)\in\mathbb{R}[x],\quad \deg L_k^{(n-1)}(x)=n-1,\quad L_k^{(n-1)}(x_j)=\delta_{jk}\quad (j=1,\dots,n) \tag{5}$$

が成り立つ。また、証明としては余談になるが、

$$F_n(x):=\prod_{j=1}^{n}(x-x_j) \tag{6}$$

とおくと、

$$L_k^{(n-1)}(x)=\frac{F_n(x)}{(x-x_k)\,F_n'(x_k)} \tag{7}$$

が成り立つ。

$$f_n(x):=\sum_{k=1}^{n}f(x_k)L_k^{(n-1)}(x) \tag{8}$$

とおくと、$f_n(x)$ は (3) を満たす。

以上で存在が示せた。$n-1$ 次多項式の係数 $(a_0,a_1,\dots,a_{n-1})$ (つまり $f_n(x)=\sum_{k=0}^{n-1}a_kx^k$ ということ) を x_1, x_2, …, x_n での値 $f_n(x_1)$, $f_n(x_2)$, …, $f_n(x_n)$ を対応させる写像は、$\mathbb{R}^n$ から $\mathbb{R}^n$ への線型写像である。上でそれが全射であることが分かった。線形代数で学ぶ定理によって、それは単射である。これは $f_n(x)$ が一意的に定まることを意味している。 ■

$L_1^{(n-1)}(x)$, …, $L_n^{(n-1)}(x)$ を、x_1, …, x_n を標本点とする **Lagrange 補間係数**と呼ぶ。

上の定理の $f_n(x)$ を f の**補間多項式** (interpolating polynomial) と呼ぶ。上の議論から分かるように

$$f_n(x)=\sum_{k=1}^{n}\frac{F_n(x)}{(x-x_k)\,F_k'(x_k)}f(x_k) \tag{9}$$

と表せる。これを **Lagrange 補間公式** (Lagrange 補間多項式, Lagrange interpolating polynomial) と呼ぶ。

(**Newton の補間公式** (Newton 補間多項式, Newton polynomial) というものもあるが、補間多項式であることには変わりがない。)

2.1.2 Runge の現象

n を大きくすると「良い」補間多項式が得られそうに思えるかもしれないが、それは誤解である。

標本点を等間隔に取って n を大きくすると、$f_n(x)$ は $f(x)$ と似ても似つかないものになることがある (**Runge の現象**と呼ばれている)。

$$a = -1, \quad b = 1, \quad N \in \mathbb{N}, \quad h = \frac{b-a}{2N} = \frac{1}{N}, \quad x_j = a + jh \quad (j = 0, 1, \ldots, 2N)$$

とする。実際に関数

$$f(x) = \frac{1}{1 + 25x^2}$$

の補間多項式を求めてグラフを描いてみよう。

```
/*
 * runge.c --- 等間隔標本点の補間多項式はポシャるという Runge の現象
 * 参考: 森正武, 数値解析, 共立出版 (1973, 第 2 版 2002).
 * gcc runge.c ; ./a.out > runge.data
 * gnuplot> f(x)=1/(1+25*x*x)
 * gnuplot> plot [-1:1] [-1:1] "runge.data" with linespoints, f(x)
 * gnuplot> plot [-1:1] [-1:10] "runge.data" with linespoints, f(x)
 *
 * ここでは Lagrange 補間多項式として計算している。
 */

#include <stdio.h>
#include <stdlib.h>

/* [-1,1] での等間隔標本点がうまく行かないことで有名な関数 */
double f(double x)
{
  return 1.0 / (1.0 + 25.0 * x * x);
}

/* Lagrange 補間係数 */
double L(double x, int k, int N, double xv[])
{
  int j;
  double t = 1;
  for (j = -N; j <= N; j++)
    if (j != k)
      t *= (x - xv[j+N]) / (xv[k+N] - xv[j+N]);
  return t;
}

/* Lagrange 補間公式 */
double fn(double x, int N, double xv[], double fv[])
{
  int k;
  double s = 0;
  for (k = -N; k <= N; k++)
    s += fv[k+N] * L(x, k, N, xv);
  return s;
}

int main(void)
{
  int j, N, nn;
  double h;
  double *xv, *fv;
  N = 10;
  xv = malloc(sizeof(double) * (2 * N + 1)); // エラーチェックさぼり
  fv = malloc(sizeof(double) * (2 * N + 1)); // 同上
  h = 1.0 / N;
  for (j = -N; j <= N; j++) {
    xv[j + N] = j * h;
    fv[j + N] = f(xv[j + N]);
  }
  nn = 200;
  h = 2.0 / nn;
  printf("%g %g\n", -1.0, fn(-1.0, N, xv, fv));
  for (j = 1; j <= nn; j++)
    printf("%g %g\n", -1.0 + j * h, fn(-1.0 + j * h, N, xv, fv));
}
```

```
$ cc -o runge runge.c
$ ./runge > runge.dat
$ gnuplot
gnuplot> f(x)=1/(1+25*x*x)
gnuplot> plot [-1:1] [-1:1] "runge.dat" with linespoints,f(x)
gnuplot> plot [-1:1] [-1:10] "runge.dat" with linespoints,f(x)
```

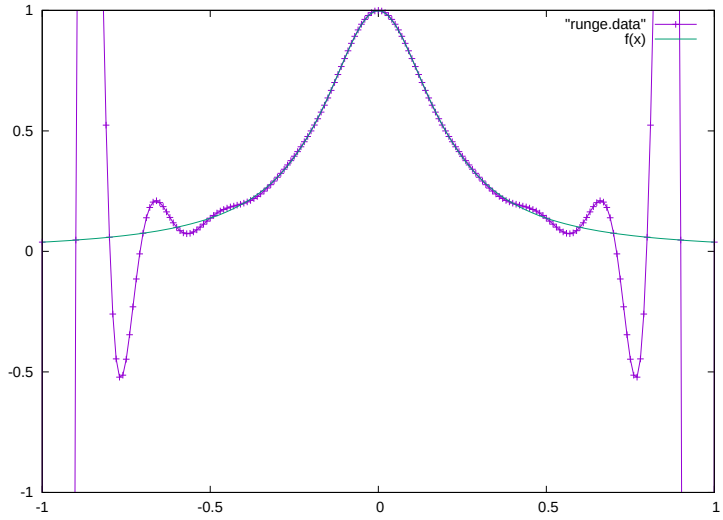

図 1: Runge の現象, $f(x) = \dfrac{1}{1+25x^2}$, $N = 10$ $(n = 21)$

グラフを見ると、区間の中央部分では、そこそこ近似できているが、中央から離れるとずれが大きくなり、端の近くでははなはだしい差が生じている。これは標本点の個数を増やしても改善されず、むしろ悪化する。 ■

上の f は多くのテキストの例に採用されている。大人しい関数と言って良いと思われる。そういう関数の等間隔標本点による補間多項式の近似がうまく行かないのは、不思議な感じがするかもしれない (私は最初知ったとき、とても驚きましたし、色々と理解した今でも不思議に感じます)。

2.1.3 Runge の現象があるので

Runge の現象を避けるために、次の 2 つの対策が良く使われる。

(a) 区間 $[a, b]$ 全体で 1 つの補間多項式を使うことをあきらめ、区間を小区間に分割して、それら各小区間で、小さい n に対して補間多項式 $f_n(x)$ を用いる。

- **スプライン近似** (spline approximation)
- 有限要素法の区分多項式
- 数値積分の**複合数値積分公式** (後述)

(b) **直交多項式**の根 (零点) を標本点とする補間多項式を利用する (直交多項式の根は、区間の端点の近くに密集している)。Gauss 型数値積分公式は、n 次の公式で、$2n-1$ 次多項式の積分を正確に計算できる。

2.2 補間型数値積分公式

区間 $[a,b]$ から標本点 $x_1,\dots,x_n$ を選び出したとき、関数 f の補間多項式 $f_n(x)$ が定まるが、それは多項式であるから、

$$I_n(f) := I(f_n) \tag{10}$$

は容易に計算でき、整理すると既に紹介した

$$I_n(f) = \sum_{k=1}^{n} A_k f(x_k) \tag{再掲 (2)}$$

の形になる。これを $I(f)$ の近似に採用したものを**補間型数値積分公式**と呼ぶ。

小さい n に対して名前がついている。それを紹介しよう。(授業では、図を板書すること。)

中点公式 $[a,b]$ の中点を標本点に採用する。$f_1(x) = f\left(\frac{a+b}{2}\right)$ は 0 次多項式 (定数) である。

$$I_1(f) = hf\left(\frac{a+b}{2}\right), \quad h := b-a. \tag{11}$$

台形公式 $[a,b]$ の端点 a, b を標本点に採用する。$f_2(x)$ は 1 次関数であり、$I_2(f)$ は台形の面積を表す。

$$I_2(f) = \frac{h}{2}\left(f(a)+f(b)\right), \quad h := b-a. \tag{12}$$

Simpson 公式 $[a,b]$ の端点 a, b と中点 $\frac{a+b}{2}$ を標本点に採用する。$f_3(x)$ は 2 次関数である。

$$I_3(f) = \frac{h}{3}\left(f(a)+4f\left(\frac{a+b}{2}\right)+f(b)\right), \quad h := \frac{b-a}{2}. \tag{13}$$

Simpson $\frac{3}{8}$ 公式 $[a,b]$ を 3 等分したときの 4 点を標本点に採用する。$f_3(x)$ は 3 次関数である。— この公式は実は使われない。

$$I_4(f) = \frac{3h}{8}\left(f(a)+3f\left(\frac{2a+b}{3}\right)+3f\left(\frac{a+2b}{3}\right)+f(b)\right), \quad h := \frac{b-a}{3}. \tag{14}$$

これらの公式の導出は、一般的に行うことも出来るが、実際に使われるのは、$n=1,2,3$ までなので、気張らないことにして省略する (やれば出来る)。$n \geq 4$ の場合はほとんんど使われない (というか、実は $n=3$ の場合もあまり使われない)。

数値積分公式が m 位の公式 (m 次の精度) であるとは、関数 f の数値積分公式の誤差を $E(f)$ と書くとき、

$$E\left(x^k\right) = 0 \quad (k=0,1,\dots,m), \quad E\left(x^{m+1}\right) \neq 0 \tag{15}$$

が成り立つことをいう。

補間型数値積分公式 $I_n(f)$ は作り方から、少なくとも $n-1$ 位の公式であるが、実は n が奇数のとき、n 位の公式である。例えば、中点公式 $I_1(f)$ と台形公式 $I_2(f)$ はともに 1 位の公式で、Simpson 公式 $I_3(f)$ と Simpson$\frac{3}{8}$ 公式 $I_4(f)$ はともに 3 位の公式である。

2.2.1 複合中点公式

$[a,b]$ を N 等分して、各小区間 $[a_j,b_j]$ $(j=1,\dots,N)$ で中点公式を用いて、それらの和を取る。

$$M_N := h\sum_{j=1}^{N} f\left(a+(j-1/2)h\right), \quad h := \frac{b-a}{N}. \tag{16}$$

複合中点公式, 複合中点則、あるいは単に中点公式とよぶ。

2.2.2 複合台形公式

$[a,b]$ を N 等分して、各小区間 $[a_j,b_j]$ $(j=1,\dots,N)$ で台形公式を用いて、それらの和を取る。

$$T_N := h\left(\frac{1}{2}f(a)+\sum_{j=1}^{N-1} f\left(a+jh\right)+\frac{1}{2}f(b)\right), \quad h := \frac{b-a}{N}. \tag{17}$$

複合台形公式, 複合台形則、あるいは単に台形公式とよぶ。

2.2.3 複合 Simpson 公式

$[a,b]$ を m 等分して、$[a_j,b_j]$ $(j=1,\dots,m)$ で Simpson 公式 ($[a_j,b_j]$ の中点も使うことになる) を用いて、それらの和を取る。

$$S_{2m} = \frac{h}{3}\left(f(a)+2\sum_{j=1}^{m-1} f(a+2jh)+4\sum_{j=1}^{m} f(a+(2j-1)h)+f(b)\right), \quad h := \frac{b-a}{2m}. \tag{18}$$

複合 Simpson 公式, 複合 Simpson 則、あるいは単に Simpson 公式とよぶ。
じつは

$$S_{2m} = \frac{T_m+2M_m}{3}, \quad T_{2m} = T_m + M_m \tag{19}$$

という関係がある。これはときどき使うことがある。
(**つぶやき**: 中点公式の誤差は、台形公式の誤差と符号が逆で、絶対値はほぼ 1/2 となっていることが多い。そこで、中点公式と台形公式を 2 : 1 に内分して作った公式の精度が高いことが期待できる。それが実は Simpson 公式である、ということになる。)

2.3 数値例

以下にあげる例は、サンプル・プログラム (C 言語で記述) を用意してある。現象数理学科 Mac であれば、ターミナルで以下のコマンドを実行すれば動くはず。

```
curl -O http://nalab.mind.meiji.ac.jp/~mk/complex2/prog20170712.tar.gz
tar xzf prog20170712.tar.gz
cd prog20170712
make
```

中点公式、台形公式、Simpson 公式は次のコードで計算できる (以下の数値実験例のプログラムの多くで、この `nc.c` をインクルードして使っている)。

nc.c

```
/*
 * 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;
}
```

2.3.1 公式の位数

(これは授業ではカットするかな。そういうわけで工事中。)

```c
/*
 * example1.c -- 多項式の積分
 */

#include <stdio.h>
#include <math.h>
#include "nc.c"

int degree = 0; // 0 or 1 or 2 or 3 or 4

double f(double x)
{
  switch (degree) {
    case 0: return 1;
    case 1: return 1 + 2 * x;
    case 2: return 1 + 2 * x + 3 * x * x;
    case 3: return 1 + 2 * x + 3 * x * x + 4 * x * x * x;
    case 4: return 1 + 2 * x + 3 * x * x + 4 * x * x * x + 5 * x * x * x * x;
    default:
      return 1;
    }

  return 1;
  // return 1 + 2 * x;
  //return 1 + 2 * x + 3 * x * x;
}

int main(void)
{
  int N;
  double a, b, I[5] = {1.0, 2.0, 3.0, 4.0, 5.0};
  double  M, T, S;
  a = 0.0; b = 1.0;
  printf("次数="); scanf("%d", &degree);
  printf("N="); scanf("%d", &N);
  M = midpoint(f, a, b, N);
  T = trapezoidal(f, a, b, N);
  S = simpson(f, a, b, N);
  printf("中点公式    M=%20.15f, 誤差=%e\n", M, I[degree] - M);
  printf("台形公式    T=%20.15f, 誤差=%e\n", T, I[degree] - T);
  printf("Simpson 公式 S=%20.15f, 誤差=%e\n", S, I[degree] - S);
  return 0;
}
```

```
$ cc -o example1 example1.c
$ ./example1
次数=2
N=10
中点公式    M=   2.997500000000000, 誤差=2.500000e-03
台形公式    T=   3.005000000000001, 誤差=-5.000000e-03
Simpson 公式 S=   3.000000000000000, 誤差=0.000000e+00
$
```

2.3.2 滑らかな関数の場合

$$I = \int_0^1 e^x \, dx$$

```
/*
 * example2.c -- \int_0^1 e^x dx
 *   cc example2.c
 *   ./a.out > ex2.data
 */

#include <stdio.h>
#include <math.h>
#include "nc.c"

double f(double x)
{
  return exp(x);
}

int main(void)
{
  int N;
  double a, b, If;
  double  M, T, S;
  a = 0.0; b = 1.0; If = exp(1.0) - 1;
  printf("#   N       I-M_N           I-T_N         I-S_N\n");
  for (N = 2; N <= 65536; N *= 2) {
    M = midpoint(f, a, b, N);
    T = trapezoidal(f, a, b, N);
    S = simpson(f, a, b, N);
    printf("%5d  %14e %14e %14e\n", N, If - M, If - T, If - S);
  }
  return 0;
}
```

```
$ cc -o example2 example2.c
$ ./example2
```

(結果は自分でやってみよう)

```
$ ./example2 > ex2.data
$ gnuplot example2.gp -
```

(結果は図 2)

グラフの横軸は N (標本点数 -1)、縦軸は誤差 (いずれも対数目盛) である。 ■

事実 2

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)。実は、これは公式の位数とも関係する。

2.3.3 滑らかでない関数の場合

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

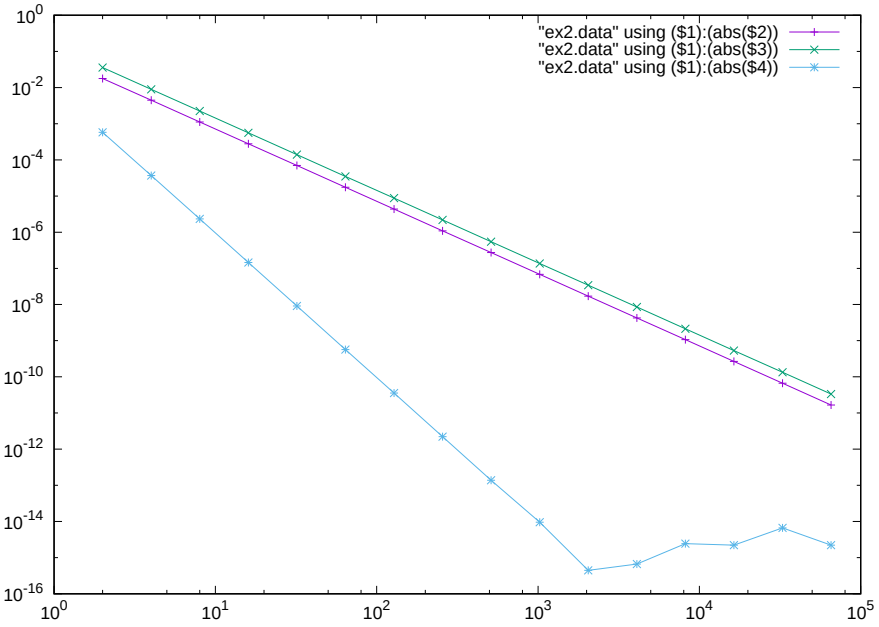

図 2: $I = \int_0^1 e^x \, dx$ を中点公式、台形公式、Simpson 公式で計算したときの誤差

```
$ cc -o example3 example3.c
$ ./example3 > ex3.data
$ cat ex3.data
  (結果はこの文書では省略)
$ gnuplot example3.gp -
```

(結果は図 3)

図 3 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

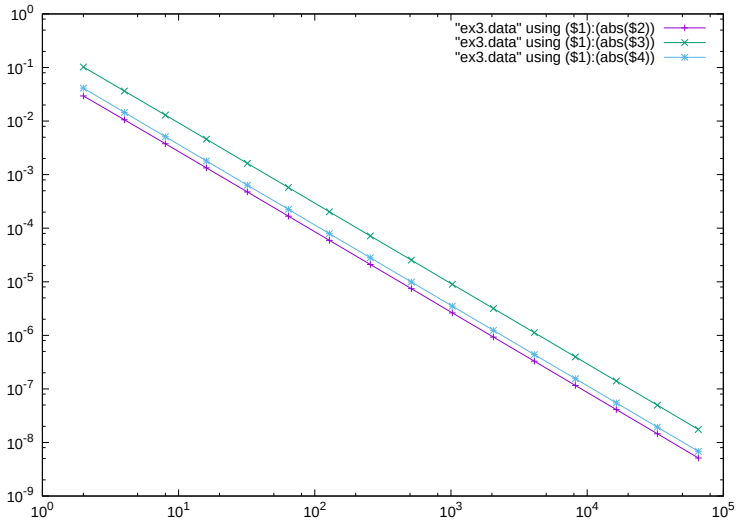

図 3: $I = \int_0^1 \sqrt{1-x^2} \, dx$ を中点公式、台形公式、Simpson 公式で計算したときの誤差

さが中点公式、台形公式と変わらないことに気付く。

事実 3

連続であるが、滑らかでない関数 (実際、$x = 1$ で微分可能でない) 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。

3　台形公式 — うまく行くのを見る

不思議なくらい、非常にうまく行く例を 2 つ紹介する。どちらも「台形公式」に関係する。

3.1　滑らかな周期関数の 1 周期の積分

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、

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

を計算する場合である。$f(a) = f(b)$ であるので、

$$T_N = h\left(\frac{1}{2}f(a) + \sum_{j=1}^{N-1} f(a+jh) + \frac{1}{2}f(b)\right) = h\sum_{j=1}^{N} f(a+jh) = h\sum_{j=0}^{N-1} f(a+jh) \tag{20}$$

であることに注意する。

3.1.1　数値例

$I = \displaystyle\int_0^{2\pi} \frac{dx}{2+\cos x} = \frac{2\pi}{\sqrt{3}}$. もう少し複雑なものが欲しければ、Bessel 関数の積分表示[1]、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
#   N       I-M_N           I-T_N          I-S_N
    2    4.860061e-01  -5.611915e-01  -1.259323e+00
    4    3.720712e-02  -3.759270e-02   1.369402e-01
    8    1.927779e-04  -1.927882e-04   1.227385e-02
   16    5.122576e-09  -5.122576e-09   6.425590e-05
   32    8.881784e-16   4.440892e-16   1.707525e-09
   64    4.440892e-16  -1.776357e-15   8.881784e-16
  128   -1.776357e-15  -4.440892e-16  -4.440892e-16
  (以下略)
$ gnuplot example4.gp -
```

$N = 32$ の段階で、台形公式の誤差 $|I - T_N| \simeq 4.4 \times 10^{-16}$. ほぼ使用している処理系の最高精度に到達している[2]。 ■

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

[2] 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。$2\pi/\sqrt{3} = 3.6275\cdots$ なので、誤差が 5.1×10^{-16} ということは、ほぼ 16 桁正しいことになる。

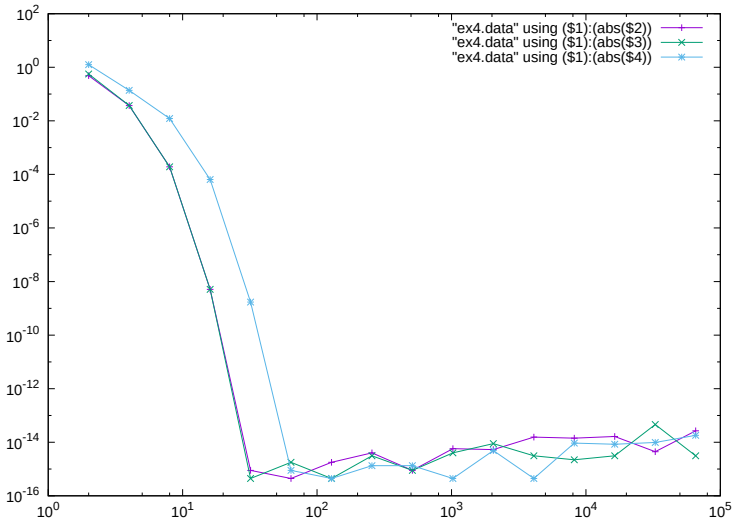

図 4: $I = \displaystyle\int_0^{2\pi} \frac{1}{2+\cos x}\,dx$ を中点公式, 台形公式, Simpson 公式で計算したときの誤差

前節の例を知った後では、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

(実は) 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。

3.1.2 Euler-Maclaurin の定理

周期関数の一周期積分が台形公式でうまく計算できることは、次の定理を用いることでひとまず説明できる。

命題 3.1 (Euler-Maclaurin 展開, Euler (1736), MacLaurin (1742), Jacobi (1834)) $f\colon [a,b] \to \mathbb{R}$ が C^{2m} 級であれば、任意の $n \in \mathbb{N}$ に対して $h = (b-a)/n$ とおくとき、

$$\int_a^b f(x)\,dx = h\left(\frac{1}{2}f(a) + \sum_{j=1}^{n-1} f(a+jh) + \frac{1}{2}f(b)\right) - \sum_{r=1}^{m} \frac{h^{2r}B_{2r}}{(2r)!}\left(f^{(2r-1)}(b) - f^{(2r-1)}(a)\right) + R_m,$$

$$R_m = \frac{h^{2m+1}}{(2m)!}\int_0^1 B_{2m}(t)\left(\sum_{k=0}^{n-1} f^{(2m)}(a+kh+th)\right)\,dt$$

が成り立つ。ただし B_m, $B_m(t)$ は、それぞれ次式で定義される **Bernoulli 数**、**Bernoulli 多項式**である:

$$\frac{s}{e^s-1} = \sum_{n=0}^{\infty} \frac{B_n}{n!}s^n, \quad \frac{se^{ts}}{e^s-1} = \sum_{n=0}^{\infty} \frac{B_n(t)}{n!}s^n \quad (|s| < 2\pi).$$

Euler や MacLaurin は、$\displaystyle\sum_{k=1}^{n} k^r$ $(r = \pm1, \pm2, \dots)$ や $\displaystyle\sum_{k=1}^{n} \log k$ を評価するためにこの公式を導出したが、この文書の立場では、台形公式の誤差を表す公式と見ることが出来る。$b-a$ が f の周期であれば、$\sum$ が 0 になり、

$$\text{台形公式の誤差} = I(f) - T_n = R_m = O\left(h^{2m+1}\right)$$

であることが期待される。

もっとも、具体的な問題に対して、この公式で誤差がどの程度になるかを調べるのは難しい。

3.2 無限区間の積分に対する台形公式

$$I = \int_{-\infty}^{\infty} f(x)\,dx \tag{21}$$

に対して、$h > 0$ を取り、

$$I_h := h \sum_{n=-\infty}^{\infty} f(nh) \tag{22}$$

とおく。実は多くの場合に I_h は I の良い近似になることが知られているが、無限和を計算することは難しいので、$N \in \mathbb{N}$ を取って、

$$I_{h,N} := h \sum_{n=-N}^{N} f(nh) \tag{23}$$

で代用することが多い。

I_h, $I_{h,N}$ も**台形公式**と呼ばれる。

問題によっては、非対称に和を取る

$$I_{h,N_1,N_2} := h \sum_{n=-N_1}^{N_2} f(nh)$$

が望ましいことがあるが、ここでは簡単のため、主に $I_{h,N}$ を用いる。

3.2.1 数値例

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、$I_{h,N}$ 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。$|x| > 6$ のとき、$0 < f(x) \le 2.4 \times 10^{-16}$ であり、もう $hf(jh)$ を加えても値が変わらない)。

```
/*
 * example5.c --- 確率積分
 *          ∞
 *    I =∫   exp(-x^2)dx = √π
 *         -∞
 * は R 上の解析関数の積分だから、台形則
 *             ∞
 *    T_h = h Σ    f(n h)              (ただし f は被積分関数)
 *           n=-∞
 * あるいは、その打ち切り
 *                N
 *    T_{h,N} = h Σ    f(n h)
 *               n=-N
 * で非常に精密に計算できるはずである。
 *
 *  コンパイル: cc -o example5 example5.c
 */

#include <stdio.h>
#include <math.h>

typedef double ddfunction(double);
ddfunction f;
double trapezoidal2(ddfunction, double, int);

/* 被積分関数 */
double f(double x)
{
  return exp(- x * x);
}

int main()
{
  int m, N;
  double pi, I, h, T;

  /* 円周率, 確率積分の真値 */
  pi = 4.0 * atan(1.0); I = sqrt(pi);

  printf(" N       h                   I               I-T\n");
  h = 1.0;
  for (m = 0; m < 3; m++) {
    /* [-6,6] で打ち切る */
    N = rint(6.0 / h);
    T = trapezoidal2(f, h, N);
    printf("%2d\t%g\t%20.15f %14e\n", N, h, T, I - T);
    h /= 2;
  }
  return 0;
}

double trapezoidal2(ddfunction f, double h, int N)
{
  int j;
  double T = 0.0;
  for (j = - N; j <= N; j++) T += f(j * h);
  T *= h;
  return T;
}
```

```
$ 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。 ■

3.3 時間の埋め草: 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) …今の学生にとっては、十分に歴史かもしれない。

上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、[6], [7] 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。

以下は完全な雑談である。

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[3] 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[4]) で、直後の発表が高橋・森 [8] であった (後に [9] 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。

4 DE 公式 速習

授業でゆっくり説明する時間は取れないだろうが、(便利のために) 大まかな筋を書いておく。

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

4.1 考え方

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

を計算したい。

$$\varphi\colon \mathbb{R} \to (a,b), \quad \lim_{t\to-\infty}\varphi(t) = a, \quad \lim_{t\to\infty}\varphi(t) = b \tag{24}$$

を満たす φ を取って、$x = \varphi(t)$ と変数変換すると

$$I = \int_{-\infty}^{\infty} f\left(\varphi(t)\right)\varphi'(t)\,dt. \tag{25}$$

$h > 0$, $N \in \mathbb{N}$ をとって、(25) に対する台形公式

$$I_h = h\sum_{n=-\infty}^{\infty} f\left(\varphi(nh)\right)\varphi'(nh), \tag{26}$$

$$I_{h,N} = h\sum_{n=-N}^{N} f\left(\varphi(nh)\right)\varphi'(nh) \tag{27}$$

を考える。

どのような φ を選べば良いか。高橋・森は

$$f\left(\varphi(t)\right)\varphi'(t) \sim C\exp\left(-C'e^{|t|}\right) \tag{28}$$

のようになる φ が良いと論じ、φ の具体例を与えた ([4], [5], [6], [7])。また、(28) を満たす変数変換を**二重指数関数型変数変換**と呼んだ。

4.2 具体的な変数変換

色々な場合に応じて、どのような変数変換 $x = \varphi(t)$ を使うのが良いかを述べる。これらの数値例は後で与えるが、時間に余裕があれば、φ や $(f\circ\varphi)\cdot\varphi'$ のグラフを描いて、後者 (被積分関数) が急速に減衰する関数であることを目で見ることを勧める (そのうち付録に収録する予定)。

4.2.1 有界区間の場合

(a,b) が有界区間の場合は、

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

は

$$x = a + \frac{b-a}{2}(u+1), \quad u \in (-1,1)$$

という 1 次関数により、

$$I = \int_{-1}^{1} F(u)\,du, \quad F(u) := f\left(a + \frac{b-a}{2}(u+1)\right)\frac{b-a}{2}$$

と変数変換されるので、以下は $\displaystyle\int_{-1}^{1} f(x)\,dx$ の場合のみ考える。

$$\varphi_1(t) := \tanh\left(\frac{\pi}{2}\sinh t\right) \quad (t \in \mathbb{R}) \tag{29}$$

が条件を満たす。

4.2.2 $\mathbb{R}$ 上の減衰の遅い関数の数値積分

$(a,b)=(-\infty,\infty)$ であるが、f が $f(x)=\frac{1}{1+x^2}$ のように

$$f(x)\sim\frac{C}{|x|^r},\quad r>1$$

程度の緩い減衰しかしない場合は、

$$x=\varphi_2(t)=\sinh\left(\frac{\pi}{2}\sinh t\right)\quad(t\in\mathbb{R}) \tag{30}$$

で変数変換すると、被積分関数は二重指数関数的に減衰するようになる。

4.2.3 $(0,\infty)$ 上の減衰の遅い関数の数値積分

$(a,b)=(0,\infty)$ であるが、f が f が $f(x)=\frac{1}{1+x^2}$ のように

$$f(x)\sim\frac{C}{|x|^r},\quad r>1$$

程度の緩い減衰しかしない場合は、

$$x=\varphi_3(t)=\exp\left(\pi\sinh t\right)\quad(t\in\mathbb{R}) \tag{31}$$

で変数変換すると、被積分関数は二重指数関数的に減衰するようになる。

4.2.4 $\mathbb{R}$ 上の積分

$(a,b)=(-\infty,\infty)$ であるが、f が $f(x)=$ のようにの場合は、

$$x=\varphi_4(t)=\exp\left(t-e^{-t}\right)\quad(t\in\mathbb{R}) \tag{32}$$

で変数変換するのが良い。この場合は、$I_{h,N}$ よりも

$$I_{h,N}=h\sum_{n=-N}^{N}f\left(\varphi(nh)\right)\varphi'(nh)$$

を採用する方が良いことが多い。
(工事中)

4.3 数値例

前節で中点公式、台形公式、Simpson 公式でまともに解けなかった

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

を φ_1 を用いる DE 公式 $I_{h,N}$ で近似してみる。
$h=1$, $N=4$ から始め、h を半分、N を 2 倍にしていく。

example6 の結果 (前半)

```
% cc -o example6 example6.c
% ./example6
test1 (sqrt(1-x^2) の積分)
h=1.000000, N=   4, I_hN=       1.7125198292703636, I_hN-I=1.417235e-01
h=0.500000, N=   8, I_hN=       1.5709101233831166, I_hN-I=1.137966e-04
h=0.250000, N=  16, I_hN=       1.5707963267997540, I_hN-I=4.857448e-12
h=0.125000, N=  32, I_hN=       1.5707963267948970, I_hN-I=4.440892e-16
h=0.062500, N=  64, I_hN=       1.5707963267948968, I_hN-I=2.220446e-16

(後略)
```

驚くべきことに、$h = \frac{1}{8}$, $N = 24$ で誤差がほぼ 10^{-16} 程度になっている。$x = \pm 1$ にあった特異性は、変数変換により消えてしまった。

それどころか、もっと特異性の強い ($x = \pm 1$ で分母が 0 !!)

$$I = \int_{-1}^{1} \frac{dx}{\sqrt{1-x^2}} = \pi$$

に対しても、計算が可能である。

```
test2 (1/sqrt(1-x^2) の積分)
h=1.000000, N=   4, I_hN=       3.1435079763395439, I_hN-I=1.915323e-03
h=0.500000, N=   8, I_hN=       3.1415926717394895, I_hN-I=1.814970e-08
h=0.250000, N=  16, I_hN=       3.1415926194518016, I_hN-I=-3.413799e-08
h=0.125000, N=  32, I_hN=       3.1415926318228000, I_hN-I=-2.176699e-08
h=0.062500, N=  64, I_hN=       3.1415926343278699, I_hN-I=-1.926192e-08
h=0.031250, N= 128, I_hN=       3.1415926326210668, I_hN-I=-2.096873e-08
h=0.015625, N= 256, I_hN=       3.1415926323669527, I_hN-I=-2.122284e-08
h=0.007812, N= 512, I_hN=       3.1415926327540080, I_hN-I=-2.083579e-08
h=0.003906, N=1024, I_hN=       3.1415926312582507, I_hN-I=-2.233154e-08
h=0.001953, N=2048, I_hN=       3.1415926319069589, I_hN-I=-2.168283e-08
%
```

$h = \frac{1}{2}$, $N = 8$ で誤差が 10^{-8} 程度になっている。その後は、分割を細かくしても精度は上がらないが、これは**桁落ち** (cancellation of siginificant digits) のためで、適切に対策すれば解決できる。対策の詳細は省略するが (後で付録にでも入れる)、次のような結果が得られる。

example6kai の結果 (後半)

```
% cc -o example6kai example6kai.c
% ./example6kai

(中略)

test2 (1/sqrt(1-x^2) の (-1,1) での積分)
h=1.000000, N=   4, I_hN=       3.1435079789309328, I_hN-I=1.915325e-03
h=0.500000, N=   8, I_hN=       3.1415926733057051, I_hN-I=1.971591e-08
h=0.250000, N=  16, I_hN=       3.1415926535897940, I_hN-I=8.881784e-16
h=0.125000, N=  32, I_hN=       3.1415926535897940, I_hN-I=8.881784e-16

(以下略)
```

$h = \frac{1}{4}$, $N = 16$ で誤差が 10^{-16} 程度になっている。

example6.c

```
/*
```

```
 * example6.c --- DE 公式
 *
 *          1      2 (1/2)
 *  I1 = ∫    (1-x )     dx = π/2
 *        -1
 *
 *          1      2 (-1/2)
 *  I2 = ∫    (1-x )      dx = π
 *        -1
 *
 *  いずれも端点に特異性があって、古典的な数値積分公式はうまく行かない。
 *  double exponential formula (DE 公式) ならばうまく計算できる。
 *
 *  コンパイル: cc -o example6 example6.c
 *
 */

#include <stdio.h>
#include <math.h>
#include <string.h>

typedef double ddfunction(double);

double pi, halfpi;

/* φ */
double phi1(double t)
{
  return tanh(halfpi * sinh(t));
}

/* 2 乗 */
double sqr(double x) { return x * x; }

/* φ' */
double dphi1(double t)
{
  return halfpi * cosh(t) / sqr(cosh(halfpi * sinh(t)));
}

/* DE 公式による (-1,1) における定積分の計算 */
double de(ddfunction f, double h, double N)
{
  int n;
  double t, S, dS;
  S = f(phi1(0.0)) * dphi1(0.0);
  for (n = 1; n <= N; n++) {
    t = n * h;
    dS = f(phi1(t)) * dphi1(t) + f(phi1(-t)) * dphi1(- t);
    S += dS;
  }
  return h * S;
}

/* テスト用の被積分関数 その 1 */
double f1(double x)
{
  return sqrt(1 - x * x);
}

/* テスト用の被積分関数 その 2 */
double f2(double x)
{
```

```
  if (x >= 1.0 || x <= -1.0) // φ(t) の計算で情報落ちで 1 になる場合の安全網
    return 0;
  else
      return 1 / sqrt(1 - x * x);
}

void test(ddfunction f, double I)
{
  int m, N;
  double h, IhN;

  /* |t|≦ 4 まで計算することにする */
  h = 1.0; N = 4;
  /* h を半分, N を倍にして double exponential formula で計算してゆく */
  for (m = 1; m <= 10; m++) {
    IhN = de(f, h, N);
    printf("h=%f, N=%4d, I_hN=%25.16f, I_hN-I=%e\n", h, N, IhN, IhN - I);
    h /= 2; N *= 2;
  }
}

int main(void)
{
  pi = 4.0 * atan(1.0); halfpi = pi / 2;

  printf("test1 (sqrt(1-x^2) の積分)\n");
  test(f1, halfpi);

  printf("test2 (1/sqrt(1-x^2) の積分)\n");
  test(f2, pi);

  return 0;
}
```

4.4 DE 公式の性質

- (名前の由来) 要するに変数変換後に得られる被積分関数 $f(\varphi(t))\,\varphi'(t)$ $(t \in \mathbb{R})$ が二重指数関数的に減衰する:

$$|f(\varphi(t))\,\varphi'(t)| \le C \exp(-C' \exp|t|) \quad (|t| \to \infty).$$

- 端点における特異性に強い。例えば次のような積分でも大丈夫。

$$I = \int_{-1}^{1} \frac{1}{\sqrt{1-x^2}}\,dx$$

- (誤差の性質)

$$\Delta I_h \sim \exp\left(-\frac{C}{h}\right).$$

  これから

$$\Delta I_{h/2} \sim \exp\left(-\frac{2C}{h}\right) \sim (\Delta I_h)^2.$$

  つまり刻み幅を半分にすると、結果の有効桁数が 2 倍になる。($I_{h,N}$ についても、N を十分大きくとって、h を半分、N を 2 倍にすると、同様の挙動を示すと期待出来る。)

- Simpson 則などと比べて桁違いに高性能

  - 同じ手間で精度が何桁も良い
  - 同じ精度を得るのに手間が桁違いに少ない
- Gauss 型公式 (DE 公式発見以前は究極の公式だった) と比べても
  - やはり桁違いに高性能 (Simpson 則との比較と同様)
  - 自動積分が出来るのは有利 (これが出来ないのは Gauss 型公式の弱点)
  - 分点や重みが計算しやすい (Gauss 型の場合は手間がかかり注意が必要 — 面倒)

一方、以下のことは注意すべきである。

- 低次の多項式に対しても誤差が 0 とはならない (**固有誤差**)。
- アンダーフロー、オーバーフローが起こりやすく、プログラムを書くときに注意が必要である。

DE 公式のプログラミング上の注意については、森 [10] が詳しい。

5　数値積分の高橋・森による誤差解析理論

ここが応用関数論としてぜひ紹介したい内容で…あったはずなのだけど、時間がなくなりましたね。

5.1　誤差の特性関数

(準備中)

5.2　有理関数の積分への応用

(準備中)

5.3　誤差の特性関数の例 (1) 有限区間の場合の古典的公式

$[a,b]=[-1,1]$, $p(x)=1$ の場合の 21 点複合シンプソン公式 S_{20}

$$a=-1,\quad b=1,\quad m=10,\quad n=2m+1,\quad h=\frac{b-a}{2m},$$

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

これは

$$x_k=a+kh\quad(k=0,1,\ldots,2m),$$

$$w_k=\begin{cases}h/3 & (k=0,2m)\\ 2h/3 & (k=2j,\ j=1,2,\ldots,m-1)\\ 4h/3 & (k=2j-1,\ j=1,2,\ldots,m)\end{cases}$$

とおくと、

$$\Psi_{2m+1}(z) = \sum_{k=0}^{2m} \frac{w_k}{z - x_k}$$

と書ける。

$$\Psi(z) = \operatorname{Log} \frac{z+1}{z-1}, \quad \Phi_{2m+1}(z) = \Psi(z) - \Psi_{2m+1}(z)$$

として、$|\Phi_{2m+1}(z)|$ $(-4 \le \operatorname{Re} z \le 4,\ -4 \le \operatorname{Im} z \le 4)$ の等高線を描いてみる。
(これは森 [11] にある図と見比べるためである。)

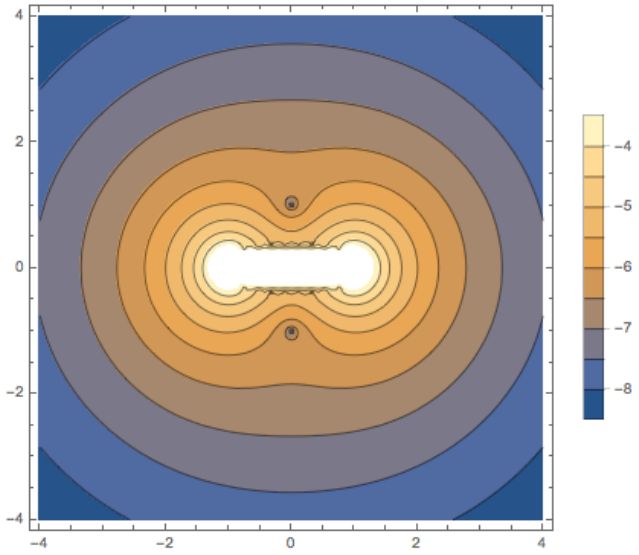

図 5: 21 点複合シンプソン公式の誤差の特性関数 (絶対値の常用対数)

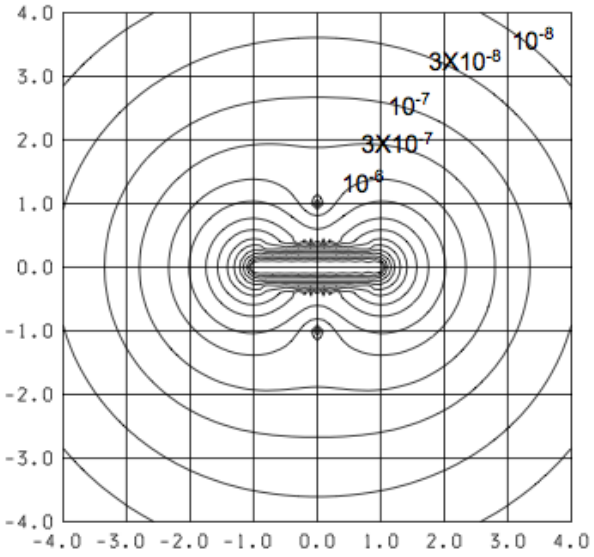

図 6: 森 [12] から $|\Phi_n(z)|$ for Simpson's formula ($h = 0.1$)

プログラムは、付録 **??** を見よ。

z 平面において、積分区間 $[a, b]$ から遠ざかると、$|\Phi_{2m+1}(z)|$ が急速に減少することが分かる。このような挙動が多くの数値積分公式に共通して見られることは、**??** で述べたことから理解できる。

この図は実際的な誤差評価に使うことが出来る。

例 5.1 (森 [12] から引用)

$$I = \int_{-1}^{1} \frac{1}{x-2}\, dx = -\log 3 = -1.0986\ 12288\ \cdots$$

被積分関数は有理関数で、2 を唯一の極に持つ。留数は

$$\operatorname{Res}\left(\frac{1}{z-2}; 2\right) = 1,$$

$n = 21$ の複合 Simpson 公式では、等高線図から

$$|\Phi_n(2)| \fallingdotseq 3 \times 10^{-6}.$$

系**??** を用いて

$$|\Delta I_n| = \left|\Phi_n(2) \operatorname{Res}\left(\frac{1}{z-2}; 2\right)\right| \fallingdotseq 3 \times 10^{-6}.$$

実際に 21 点 Simpson 公式で計算すると、$I_n = -1.0986\ 15504\ \cdots$ となり、誤差は -3×10^{-6} 程度。被積分関数が有理関数でない場合は、これほど簡単にはいかないが、代替手段として、**鞍点法**(あんてんほう) (saddle point method) と呼ばれる方法が使える場合がある (森 [13] を見よ)。 ■

もう 1 つ、有限区間上の数値積分公式の誤差の特性関数の例をあげておく。この講義では解説していないが、有名な Gauss-Legendre 公式の場合を紹介する。

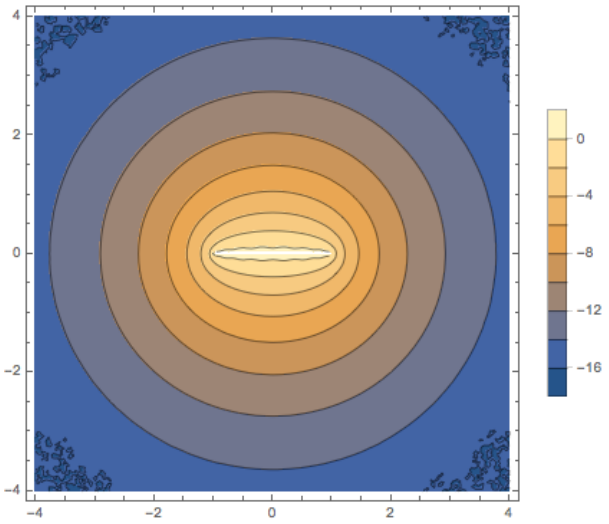

図 7: $(-1,1)$ における 8 次 Gauss-Legendre 公式の誤差の特性関数 (絶対値の常用対数)

8 次の Gauss-Legendre 公式は、$n=8$ 次の直交多項式の 8 個の零点を標本点に使い、$2n-1=15$ 次までの多項式について正確な積分を計算できる。つまり 15 位の公式である。実際 $|\Psi_8(z)| = 10^{-16}$ の曲線が見え、21 点 Simpson 公式よりも格段に誤差の特性関数の値が小さい (8 桁下、つまり 1 億分の 1) ことが分かる。

5.4 誤差の特性関数の例 (2) 無限区間の台形公式

$$I = \int_{-\infty}^{\infty} f(x)\,dx$$

に対する台形公式

$$I_h = h \sum_{n=-\infty}^{\infty} f(nh)$$

の場合、$x_n = nh$, $w_n = h$ $(n \in \mathbb{Z})$ だから、

$$\Psi_h(z) \stackrel{?}{=} \sum_{n=-\infty}^{\infty} \frac{h}{z-nh}.$$

実はこれは残念ながら収束しない。しかし次のように小修正すれば良い。

$$\Psi_h(z) = \lim_{N\to\infty} \sum_{n=-N}^{N} \frac{h}{z-nh} = \pi \cot \frac{\pi z}{h}.$$

一方、$\Psi(z)$ はどうすべきか？有限の (a,b) の場合の $\operatorname{Log} \dfrac{z-a}{z-b}$ の適当な極限として

$$\Psi(z) = \lim_{R\to\infty} \operatorname{Log} \frac{z+R}{z-R} = \begin{cases} -i\pi & (\operatorname{Im} z > 0) \\ i\pi & (\operatorname{Im} z < 0). \end{cases}$$

(最後の等式の証明も手頃な問題かな？と思ったので残す。余談?? と見比べると良い。) 実際にこの Φ について

$$I = \frac{1}{2\pi i} \int_\Gamma \Psi(z) f(z)\, dz$$

が成り立つことは直接簡単に証明できる。

これから

$$\Delta I_h := I - I_h = \frac{1}{2\pi i} \int_\Gamma \Phi_h(z) f(z)\, dz, \quad \Phi_h(z) := \Psi(z) - \Psi_h(z).$$

$|\Phi_h(z)|$ の等高線を見よう。

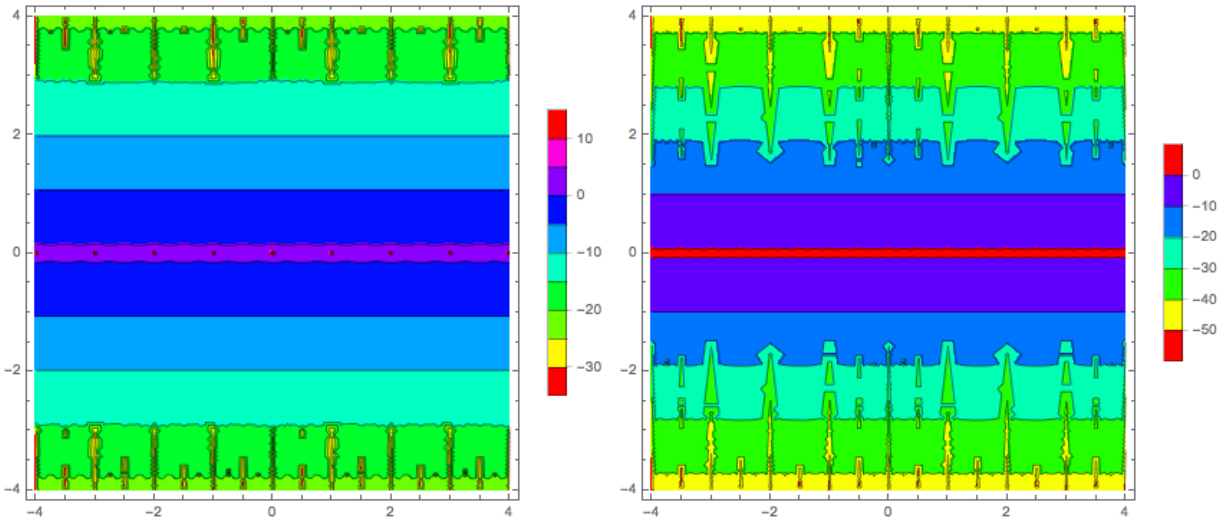

図 8: $\mathbb{R}$ 上の積分に対する台形公式 I_h $(h = 1/2, 1/4)$ の誤差の特性関数の絶対値

ぞっとするほど小さいことが見て取れる。

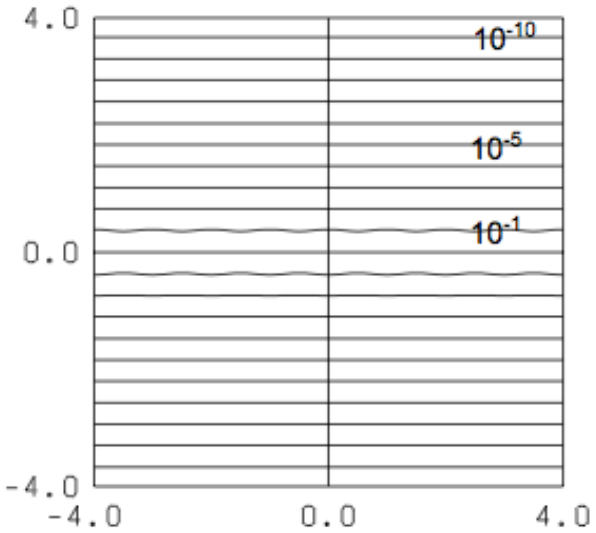

図 9: 森 [12], $\mathbb{R}$ 上の積分に対する台形公式 I_h $(h = 0.1)$ の誤差の特性関数の絶対値… 多分誤植で、実際は $h = 1$ の場合の図だと思われる。

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 章は数値積分の高橋-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] Takahasi, H. and Mori, M.: Quadrature Formulas Obtained by Variable Transformation, *Numerische Mathematik*, Vol. 21, pp. 206–219 (1973).

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

[8] 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.

[9] 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).

[10] 森正武：FORTRAN 77 数値計算プログラミング, 岩波書店 (1986, 1987).

[11] 森正武（まさたけ）：数値解析, 共立出版 (1973), 第 2 版が 2002 年に出版された。

[12] Mori, M.: Discovery of the Double Exponential Transformation and Its Developments, *Publ. RIMS, Kyoto Univ.*, Vol. 41, pp. 897–935 (2005), http://www.kurims.kyoto-u.ac.jp/~okamoto/paper/Publ_RIMS_DE/41-4-38.pdf で入手可能.

[13] 森正武：数値解析と複素関数論, 筑摩書房 (1975), 入手しづらくて困る。あ！ちくま学芸文庫に入れたらどうかなあ。筑摩書店の方、ぜひ考えてみて下さい。