数値積分ノート

桂田 祐史

2016 年 4 月 26 日, 2018 年 7 月 23 日

森 [1], 杉原・室田 [2], 山本 [3], 一松 [4], 篠原 [5] 等のテキストや、高橋、森、杉原の論文を参考にした。

「応用複素関数」としては、6 節、F 節に重点をおきたいわけだが、時間に追われて、F 節は説明できなかった (2016/7/6 現在, もしかしたら最後に説明する)。

来年度に向けて、今年度やったことをこの文書に詰め込んでいる。以前から気にかかっていたことの多くを実行できたが、全体をうまくまとめたとは言い難い。来年度はきちんとやりたい。

目 次

TO DO LIST

- DE 公式の説明の記号を一松流から森・杉原流に直す (これは来年度版のファイルに乗り換えてから)。
- DE 公式の杉原流の解析の勉強
- Runge の現象を効率的な補間の計算で。
- 周期関数に対する台形公式の解析
- 桁落ち対策プログラム `de3(f,g, H, a, b, h, N)`
  f 以外に f を原点移動した $g(y)=f(y+b)$, $H(y)=f(y+a)$ が必要になる。
  これで色々な計算をしよう。
- 数値積分の平山法

1 数値積分入門 (2016/4/27)

2016/4/27 の講義では、この節以外に、付録 A.1, A.2 の内容を講義した。

1.1 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。

積分の計算が微分の計算よりも難しい場合が多いということは知っているであろう。与えられた関数 f が、導関数が分かっている関数を組み合わせたものであるとき、f の導関数は容易に計算出来るが、同様のことが積分については成立しない (商の微分法、合成関数の微分法があるが、商の積分法、合成関数の積分法はないことが理由としてあげられる。$\sin x$, x の原始関数を知っていても、$\dfrac{\sin x}{x}$ の原始関数は分からない。)。

(原始関数が初等関数にならないということについて、一松 [6] に解説がある。)

そこで、定積分の値を数値計算で求めることを考える。

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

を考える。

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**と呼ばれる。また A_k は**重み**と呼ばれる。$I_n(f)$ は、n 個の標本点での f の関数値に重みをかけて和を取ったものである。

簡単な例をあげよう (図を描いて、各 $I_n(f)$ がどういう図形の面積を表しているか説明すること)。

例 1.1 (中点公式)

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

例 1.2 (台形公式)

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

例 1.3 (Simpson 公式)

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

例 1.4 (Simpson $\dfrac{3}{8}$ 公式)

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

注意 1.5 Simpson $\dfrac{3}{8}$ 公式には利点がほとんどないことを、後で紹介する予定である。使う価値のない公式であるので、間違えて使わないように。 ■

1.2 補間型数値積分公式

ここを説明する前に補間多項式 (付録 A.1, A.2) の説明をすること。

定積分 $I = \int_a^b f(x)\,dx$ の f を、区間 $[a,b]$ から選んだ相異なる n 個の点 (標本点と呼ばれる) $x_1, \cdots, x_n$ に関する f の Lagrange 補間多項式 $f_n(x)$ で置き換えて得られる

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

を**補間型数値積分公式**と呼ぶ。

標本点 $x_1, \cdots, x_n$ に関する補間多項式 $f_n(x)$ は、

$$f_n(x) \in \mathbb{R}[x], \quad \deg f_n(x) \le n-1, \quad f_n(x_j) = f(x_j) \quad (j = 1, 2, \cdots, n)$$

を満たすものとして定義されるが、次のように具体的に表せる。

$$f_n(x) := \sum_{k=1}^{n} f(x_k) L_k^{(n-1)}(x), \quad L_k^{(n-1)}(x) := \prod_{\substack{1 \le j \le n \\ j \ne k}} \frac{x - x_j}{x_k - x_j}.$$

$L_k^{(n-1)}(x)$ は、標本点 $x_1, \cdots, x_n$ に関する Lagrange 補間係数と呼ばれる。次の条件で特徴づけることも出来る:

$$L_k^{(n-1)}(x) \in \mathbb{R}[x], \quad \deg L_k^{(n-1)}(x) = n-1, \quad L_k^{(n-1)}(x_j) = \delta_{jk} \quad (j = 1, 2, \ldots, n).$$

$$F_n(x) := \prod_{j=1}^{n} (x - x_j) = (x - x_1) \cdots (x - x_n)$$

とおくと、

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

と表すことも出来る。

$$I_n = \int_a^b f_n(x)\,dx = \int_a^b \sum_{k=1}^{n} f(x_k) L_k^{(n-1)}(x)\,dx = \sum_{k=1}^{n} f(x_k) \int_a^b L_k^{(n-1)}(x)\,dx$$

であるから、

$$A_k := \int_a^b L_k^{(n-1)}(x)\,dx \tag{3}$$

とおくと、

$$I_n = \sum_{k=1}^{n} A_k f(x_k). \tag{4}$$

一般に、数値積分公式 $I_n(f)$ が m 次の精度 (m 位の公式) であるとは、$E_n(f) := I(f) - I_n(f)$ とおいたとき、次の条件が成り立つことをいう。

$$(\forall k \in \{0, 1, \cdots, m\}) \quad E_n\left(x^k\right) = 0 \quad \wedge \quad E_n\left(x^{m+1}\right) \ne 0. \tag{5}$$

(m 次以下の任意の多項式の積分を正確に計算できるが、$m+1$ 次の多項式で積分が正確に計算出来ないものがある、ということである。)

補間型数値積分公式 $I_n(f)$ は少なくとも $n-1$ 位の公式のはずである。

1.3 Newton-Cotes 公式 (等間隔標本点の補間型数値積分公式)

$x_0, x_1, \cdots, x_N$ を区間 $[a,b]$ の N 等分点とする。すなわち、

$$h = \frac{b-a}{N}, \quad x_k = a + kh \quad (k = 0, 1, \cdots, N). \tag{6}$$

$x_0, x_1, \cdots, x_N$ を標本点とする補間型数値積分公式

$$I_{N+1} = \sum_{k=0}^{N} A_k f(x_k) \tag{7}$$

を **Newton-Cotes の積分公式**と呼ぶ。ここで

$$A_k = \int_a^b L_k^{(N)}(x)\, dx, \quad L_k^{(N)}(x) = \prod_{\substack{0 \le j \le N \\ j \ne k}} \frac{x - x_j}{x_k - x_j} \quad (k = 0, 1, \cdots, N). \tag{8}$$

$N = 1$ の場合が**台形則** (例 1.2), $N = 2$ の場合が **Simpson 則** (例 1.3) である。
作り方から次が成り立つことは、ほぼ明らかであろう。

命題 1.6 N 次以下の多項式 $f(x)$ に対して、N 次 Newton-Cotes 型数値積分公式 I_{N+1} は正確な値を与える。すなわち

$$I_{N+1}(f) = I(f) \left(= \int_a^b f(x)\, dx \right).$$

すなわち I_{N+1} は少なくとも N 位の公式である。

証明 $f(x) = f_N(x)$ であるから

$$I(f) = \int_a^b f(x)\, dx = \int_a^b f_N(x)\, dx = I_{N+1}(f). \blacksquare$$

しかし実は N が偶数のときは、I_{N+1} は $N+1$ 次以下の多項式に対して正確な値を与える (後で証明する)。

系 1.7 (1) 台形則は 1 次以下の多項式の積分を正確に計算できる。

(2) Simpson 則は 2 次以下の多項式の積分を正確に計算できる (実は 3 次以下の多項式の積分を正確に計算できる)。

注意 1.8 (番号の付け方) I_n, $f_{n-1}(x)$, $L_j^{(n-1)}(x)$, I_{N+1}, $f_N(x)$, $L_j^{(N)}(x)$ というのが出て来たが、

- 積分公式 I_n, I_{N+1} の添字 n, $N+1$ は、標本点の個数を表している。
- 多項式 $f_{n-1}(x)$, $f_N(x)$, $L_j^{(n-1)}(x)$, $L_j^{(N)}(x)$ の添字 $n-1$, N は、多項式の次数を表している。$\blacksquare$

公式 I_{N+1} の具体形

N が小さい場合に A_k を計算してみよう。$x_0, x_1, \ldots, x_N$ を標本点とする補間多項式は

$$f_N(x) = \sum_{j=0}^{N} f(x_j) L_j^{(N)}(x), \quad L_j^{(N)}(x) = \prod_{k \neq j} \frac{x - x_k}{x_j - x_k}.$$

ただし $\prod_{k \neq j}$ は、$k \in \mathbb{Z}, 0 \leq k \leq N, k \neq j$ を満たす k についての積を表す。

$$I_{N+1} = \int_a^b \sum_{j=0}^{N} f(x_j) L_j^{(N)}(x)\, dx = \sum_{j=0}^{N} \frac{f(x_j)}{\prod_{k \neq j}(x_j - x_k)} \int_a^b \prod_{k \neq j} (x - x_k)\, dx.$$

$x = a + th$ $(t \in [0, N])$ と変数変換すると

$$dx = h\, dt,$$
$$\prod_{k \neq j} (x - x_k) = \prod_{k \neq j} (a + th - (a + kh)) = h^N \prod_{k \neq j} (t - k),$$

また

$$\prod_{k \neq j} (x_j - x_k) = \prod_{k \neq j} (j - k)h = h^N \prod_{k \neq j} (j - k) = h^N (-1)^{N-j} \frac{N!}{\binom{N}{j}}$$

であるから

$$I_{N+1} = \sum_{j=0}^{N} f(x_j) A_j, \quad A_j = h \cdot (-1)^{N-j} \frac{\binom{N}{j}}{N!} \int_0^N \prod_{k \neq j} (t - k) dt. \tag{9}$$

$N = 1$ のとき

$$(A_0, A_1) = \left(\frac{h}{2}, \frac{h}{2} \right) = \frac{b-a}{2} (1, 1)$$
$$I_2(f) = \frac{b-a}{2} \left(f(a) + f(b) \right).$$

$N = 2$ のとき

$$(A_0, A_1, A_2) = \left(\frac{h}{3}, \frac{4h}{3}, \frac{h}{3} \right) = \frac{b-a}{6} (1, 4, 1),$$
$$I_3(f) = \frac{b-a}{6} \left(f(a) + 4f\left(\frac{a+b}{2} \right) + f(b) \right).$$

$N = 3$ のとき

$$(A_0, A_1, A_2, A_3) = \left(\frac{3h}{8}, \frac{9h}{8}, \frac{9h}{8}, \frac{3h}{8} \right) = \frac{b-a}{8} (1, 3, 3, 1),$$
$$I_4(f) = \frac{b-a}{8} \left(f(a) + 3f\left(\frac{2a+b}{3} \right) + 3f\left(\frac{a+2b}{3} \right) + f(b) \right).$$

$N = 4$ のとき、

$$(A_0, A_1, A_2, A_3, A_4) = \left(\frac{14h}{45}, \frac{64h}{45}, \frac{24h}{45}, \frac{64h}{45}, \frac{14h}{45} \right) = \frac{b-a}{90} (7, 32, 12, 32, 7),$$
$$I_5(f) = \frac{b-a}{90} \left(7f(a) + 32f\left(\frac{3a+b}{4} \right) + 12f\left(\frac{2a+2b}{4} \right) + 32f\left(\frac{a+3b}{4} \right) + 7f(b) \right).$$

Mathematica で検算

```
a[NN_] := h Table[(-1)^(NN - j) Binomial[NN, j]/NN!
   Integrate[Product[t - k, {k, 0, NN}]/(t - j), {t, 0, NN}], {j, 0, NN}]
a[1]
a[2]
a[3]
a[4]
```

(h のところは (b-a)/NN とする方が良いか？)

2 古典的公式訪問 (2016/5/11)

2.1 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[2])

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[3]。

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。

2.2 前回紹介した公式のまとめ

2.2.1 (複合) 中点公式

区間 $[a,b]$ を N 等分して、各小区間 $[a_j,b_j]$ $(j=1,\dots,N)$ 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]微積分の講義は、通常忙しくて他にやることが一杯ある。筆者は数値積分については結構関心を持っている方で、一方で積分の基礎づけを長い時間をかけて講義した経験も持っているが、両者をどう接続するのか正直見当がつかない。微積分の教科書で数値積分に触れてあるものは何冊か見た覚えがあるが、古典的な公式の紹介だけにとどめてあるのは、何となく時代錯誤な感じがして、自分がそういう内容を講義をする気になれないからか。

[3]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;
```

2.2.2 (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;
```

2.2.3 (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

$$S_{2m} = (T_m + 2M_m)/3 \tag{10}$$

という関係がある。

2.2.4 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;
}
```

プログラム一式は

http://nalab.mind.meiji.ac.jp/~mk/lecture/applied-complex-function-2016/prog20160511.tar.gz

にまとめてある。

2.3 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 2.1 (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
```

老婆心ながら: 6.250000e-02 とは？

いわゆる指数形式 (exponential form) の表示で、6.25×10^{-2} という数値を意味している。

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 2.2 (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

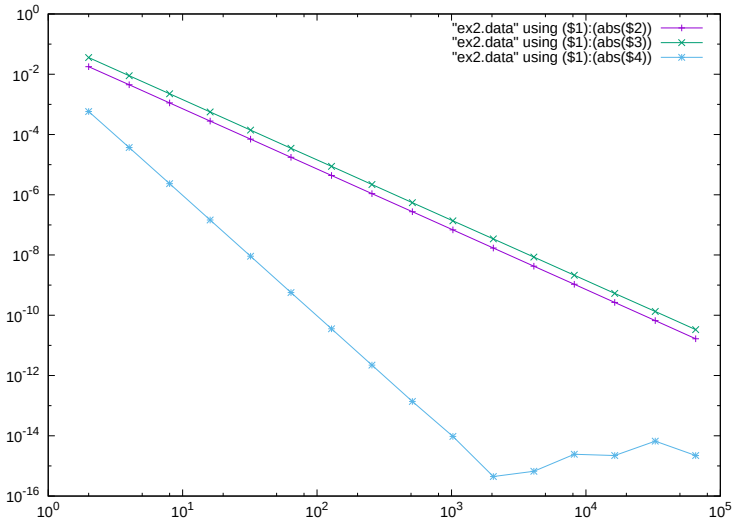

図 1: $I = \displaystyle\int_0^1 e^x\,dx$ を中点公式、台形公式、Simpson 公式で計算したときの誤差

あることが分かる。 ■

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、$I = \displaystyle\int_a^b f(x)\,dx$ の値が分かるものを試すのが良い。) ■

余談 2.3 関数の 3 次関数補間に基づく、Simpson $\dfrac{3}{8}$ 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 $\dfrac{3}{8}$ 公式の順に $r = 2, 2, 4, 4$. (1, 2, 3, 4 ではない！)■

2.4 一般論の守備範囲外の話

2.4.1 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 2.4 (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

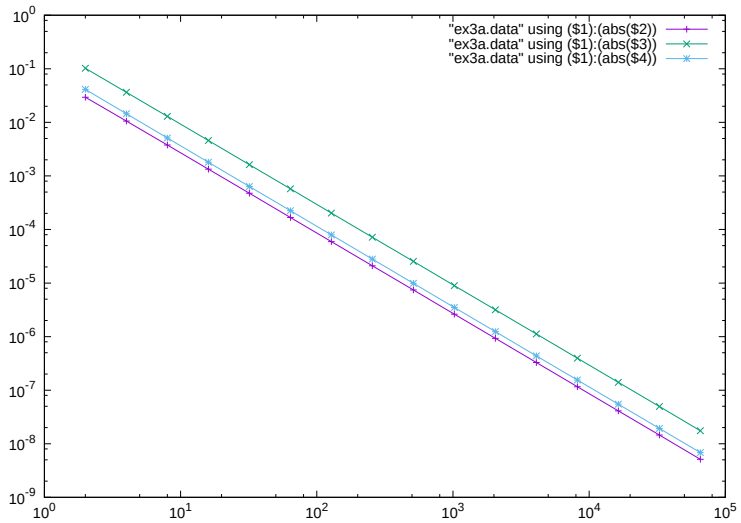

図 2: $I = \int_0^1 \sqrt{1 - x^2}\, dx$ を中点公式、台形公式、Simpson 公式で計算したときの誤差

速さが中点公式、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。 ■

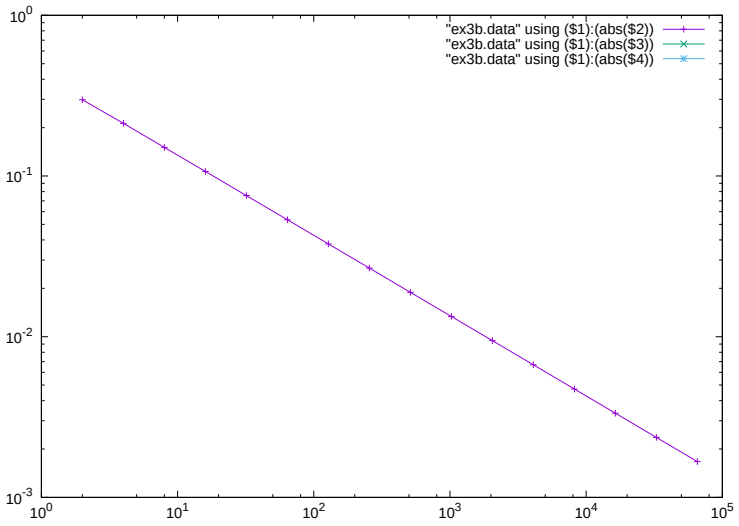

図 3: $I=\displaystyle\int_0^1 \frac{1}{\sqrt{1-x^2}}\,dx$ を中点公式で計算したときの誤差

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、この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。 ■

2.4.2 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 2.5 (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[4]、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 example4.gp -
```

$N = 16$ 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[5]。 ■

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

(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。

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

[5] 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。

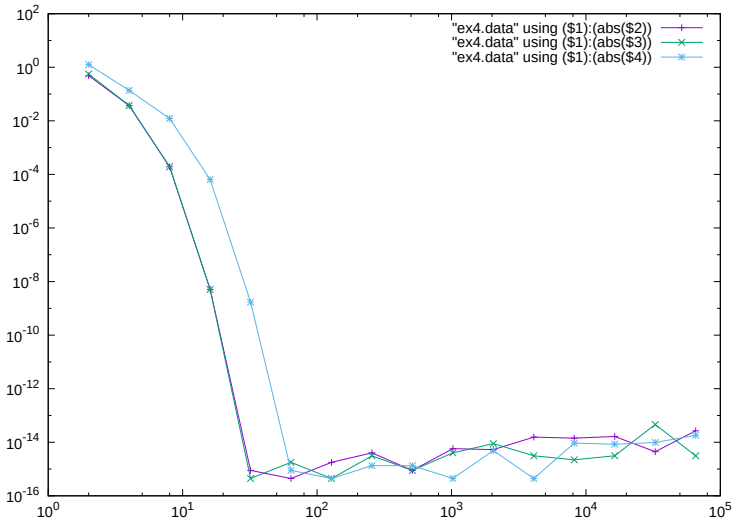

図 4: $I = \displaystyle\int_0^{2\pi} \frac{1}{2+\cos x}\,dx$ を中点公式, 台形公式, Simpson 公式で計算したときの誤差

余談 2.6 (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。)

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 2.7 (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 (相対誤差 10^{-16} 程度) に到達している。 ■

2.5 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) …

§§2.4.2 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 ([7], [8])。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 [4] 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 ([9], [10] が報告され、[11], [12] 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[6] 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 [7] 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[7]) で、直後の発表が高橋・森 [13] であった (後に [14] 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。

3 2016/5/18

まだまだ粗い。

5 月 18 日の予定

1. 周期関数の一周期積分に台形公式が有効な理由の説明。Euler-Maclaurin 展開。
2. 最初に Mathematica で変数変換をやってみる方が良いかも (§5.5)。DE 公式のアイディアの説明。数値例の紹介。
3. DE 公式の誤差解析についてのお話 — これは出来なかった。この文書からもカットした (5/25 の講義ノートの付録に回した)。

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

4　台形公式の優秀性を見る

4.1　周期関数の周期積分に対する台形公式の誤差, 実関数論的な説明

周期関数の周期積分に対しては台形公式がベスト

この事実は古くから知られているが、通常、次の **Euler-Maclaurin 展開** (実は有名な公式) によって説明される。左辺は $I = \int_a^b f(x)\,dx$, 右辺の第 1 項は複合台形則 T_n であるから、誤差が右辺の第 2 項と第 3 項の和であるが、f が周期 $b-a$ であれば、右辺第 2 項の $\sum$ の項はすべて 0 になり、誤差は $O(h^{2m+1})$ であることが分かる。

命題 4.1 (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、

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

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

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:

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

(Bernoulli 数については、荒川・伊吹山・金子 [15] が詳しい。「複素関数」の講義ノート [16] の §9.5 にも少し解説しておいた。最初の数項を書いておく (それで数値積分の話には十分のはず)。

$$B_0 = 1, \quad B_1 = -\frac{1}{2}, \quad B_2 = \frac{1}{6}, \quad B_3 = 0, \quad B_4 = -\frac{1}{30}, \quad B_5 = 0, \quad B_6 = \frac{1}{42}, \quad \cdots$$

Mathematica には、ベルヌーイ数, ベルヌーイ多項式を計算する関数 `BernoulliB[n]`, `BernoulliB[n, x]` が用意されている。)

余談 4.2 (歴史メモ) ハイラー・ヴァンナー [17] によると、Euler と Maclaurin は、$\sum_{k=1}^{n}\frac{1}{k}$ や、$\sum_{k=2}^{n}\log k = \log n!$, $\sum_{k=1}^{n}k^r$, $\sum_{k=1}^{n}\frac{1}{k^r}$ などを計算するために

$$\sum_{i=1}^{n} f(i) = \int_0^n f(x)\,dx + \frac{1}{2}\left(f(n)-f(0)\right) + \frac{1}{12}\left(f'(n)-f'(0)\right) - \frac{1}{720}\left(f'''(n)-f'''(0)\right) - \frac{1}{30240}\left(f^{(5)}(n)-f^{(5)}(0)\right) + \cdots$$

という公式を独立に導いた (例えば、r を自然数として、$f(x) = x^r$ としてみると冪乗和 $\sum_{k=1}^{n}k^r$ の公式が得られる。公式の右辺が有限和になることに注意せよ。)。

$$B_0 = 1, \quad \sum_{i=0}^{k-1}\binom{k}{i}B_i = 0 \quad (k = 2, 3, \cdots)$$

で定義される Bernoulli 数[8]を用いると

$$\sum_{i=1}^{n} f(i) = \int_0^n f(x)\,dx + \frac{1}{2}\left(f(n) - f(0)\right) + \sum_{k=1}^{\infty} \frac{B_{2k}}{(2k)!}\left(f^{(2k-1)}(n) - f^{(2k-1)}(0)\right)$$

と書ける。この式は実は一般に成り立つ式ではない。例えば $f(x) = \cos 2\pi x$ に対して、左辺は $1+1+\cdots$, 右辺は $0+0+\cdots$ となり、左辺は収束すらしない。有限項で打ち切って、剰余項を与える必要がある。その結果

$$\sum_{i=1}^{n} f(i) = \int_0^n f(x)\,dx + \frac{1}{2}\left(f(n) - f(0)\right) + \sum_{j=2}^{k} \frac{B_j}{j!}\left(f^{(j-1)}(n) - f^{(j-1)}(0)\right) + \widetilde{R}_k,$$

$$\widetilde{R}_k = \frac{(-1)^{k-1}}{k!} \int_0^n \widetilde{B}_k(x) f^{(k)}(x) dx.$$

が得られた (Jacobi, 1834 年)。ここで $\widetilde{B}_k(x)$ は、$[0,1]$ 区間での $B_k(x)$ を周期 1 の周期関数となるように $\mathbb{R}$ 全体に拡張したものを表す。その後、Wirtinger により発見された部分積分を繰り返す証明が簡明であるとか。 ■

(参考) IMT 公式

(ここは 2016 年度の授業ではカットする。)

IMT 公式は、周期関数の 1 周期区間の積分は台形公式で高精度に近似出来るという事実に注目して考案された (伊理・森口・高澤 [7], [8])。

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

を計算するために、次式で定義される φ を用いて、変数変換 $x = \varphi(t)$ を行い、台形公式を用いる。

$$\varphi(x) = \frac{1}{Q}\int_0^x \exp\left(-\frac{1}{t} - \frac{1}{1-t}\right) dt,$$

$$\begin{aligned} Q :&= \int_0^1 \exp\left(-\frac{1}{t} - \frac{1}{1-t}\right) dt \\ &= 0.00702\ 98584\quad 06609\ 65623\quad 92412\ 70530\quad 35395\ 60761\quad 55399\ 47535 \\ &\quad\ 72487\ 96129\quad 73882\ 86445\quad 45869\ 54635\quad 93462\ 85080\quad 43897\ 70339\ 32 \end{aligned}$$

(一松 [4] では、小数点以下 34 桁までの数値が記してあって、25 桁以降が 70530 3401 となっているが、32 桁目から食い違っている。)

すなわち、次の I_n を近似値に採用する。

$$I_n = \frac{1}{n}\sum_{k=1}^{n-1} w_k^{(n)} f\left(x_k^{(n)}\right), \quad x_k^{(n)} := \varphi\left(\frac{k}{n}\right), \quad w_k^{(n)} := \varphi'\left(\frac{k}{n}\right).$$

適当な n に対して、数列 $\left\{x_k^{(n)}\right\}, \left\{w_k^{(n)}\right\}$ を用意する必要がある。(今でこそ、Mathematica を使えば容易に計算できるが、そういうものがなければ、結構面倒である。)

---

[8] $\dfrac{u}{e^u - 1} = \displaystyle\sum_{n=0}^{\infty} \frac{B_n}{n!} u^n$ で定義されると言っても同じである。

4.2 $\mathbb{R}$ 上の広義積分に対する台形公式の誤差, 数値実験例の紹介

$\mathbb{R}$ 上の広義積分

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

で、f が解析関数 ($\mathbb{C}$ における実軸の近傍で正則な関数に拡張できる) の場合、I の近似値として

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

を採用した場合の精度が非常に (神秘的に感じられるくらい) 良い。特に $|x| \to \infty$ での減衰が速い場合、I_h を有限和で打ち切った

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

が少ない計算量で高精度の値を与える。I_h, $I_{h,N}$ も**台形公式**と呼ばれる。

この事実の定式化と証明は一応用意したが (http://nalab.mind.meiji.ac.jp/~mk/complex2/tmh.pdf の付録 B)、ここでは数値例を紹介するにとどめて、次節ではそれを利用したアルゴリズム (DE 公式) の話をしよう。

例 4.3 (遠方で速く減衰する関数の $\mathbb{R}$ 全体での積分) (これは先週の資料にも載せたもののコピー) 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。$|x| > 6$ 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)。

```
$ 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。 ■

5 二重指数関数型数値積分公式

高橋秀俊, 森正武の研究として名高い**二重指数関数型数値積分公式** (**double exponential formula**, DE 公式) を解説する。

授業では、先に絵的に理解してもらった方が良いかもしれないので、話す順番は工夫すること (先に 5.5 節の内容をやった)。

5.1 基本的なアイディア

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

に

$$a = \lim_{t\to-\infty} \varphi(t), \quad b = \lim_{t\to\infty} \varphi(t)$$

を満足する滑らかな単調増加関数 $\varphi\colon \mathbb{R} \to (a,b)$ を用いて変数変換

$$x = \varphi(t)$$

を施すと

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

十分小さな $h > 0$ を取り、次の式で I を近似することを考える。

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

実際には、無限和の計算は実行出来ないので、適当な $N \in \mathbb{N}$ を選び、

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

と打ち切ったものを計算する。I_h, $I_{h,N}$ も台形公式と呼ばれる。

注 $I_{h,N}$ は、一応 $\sum_n f(x_n)w_n$ (標本点における f の値に重みをかけたものの総和) という形をしている。

φ をどう選択するのが良いだろうか？高橋・森 ([10], [12]) の結論は、(有界区間における積分 $\int_a^b f(x)\,dx$ の場合には)

$$\varphi'(t) \fallingdotseq \exp(-C\exp|t|) \quad (|t| \to \infty) \tag{11}$$

という評価が成り立つようにするのが最適である、というものであった。これが[9]「二重指数関数型数値積分公式」の名前の由来である (変数変換そのものも「二重指数関数型変数変換」と呼ばれる)。

念のため: (11) はもちろん

$$\lim_{t\to\infty} \varphi'(t) = 0 \quad (\text{収束が非常に速い})$$

を意味するが、それから $\varphi(t)$ が $t \to -\infty, \infty$ のとき $\varphi(a), \varphi(b)$ に急速に近づくことにもなる。

以下は話が細かくなるのでお話。

[9] その後、他のタイプの積分についても、同様の公式が提案され (5.2.2, 5.2.3) 、それらについては、φ' が二重指数関数的に減衰するとは限らず、$f(\varphi(t))\varphi'(t)$ が二重指数関数的に減衰するようになっている。だから、ここの由来だけを覚えていると、話に食い違いが生じる。

(11) がどう導かれたか

話を簡単にするために h は固定しておくことにする。

$$I - I_{h,N} = \Delta I_h + E_{h,N}, \quad \Delta I_h := I - I_h, \quad E_{h,N} := I_h - I_{h,N}$$

と分解して考える。

$E_{h,N} = I_h - I_{h,N}$ (「項の打ち切り誤差」) を小さくしたいが、そのためには $|t| \to \infty$ とするとき、$f(\varphi(t))\varphi'(t)$ は速く 0 に減衰することが望まれる。ところがあまり急激に減衰すると、刻み幅 h が相対的に大きくなり、逆に精度が落ちると考えられる。

離散化誤差 $\Delta I_h = I - I_h$ と $E_{h,N}$ がほぼ等しくなるところで無限和を切ると仮定して解析を行うと、(11) が導かれる、ということである。その後、杉原正顯氏によって、この最適性は定理の形でより厳密に述べられるようになった ([18], [19])。

5.2 具体的な積分公式

5.2.1 有限区間上の積分

有界区間 (a, b) 上の積分は、1 次関数による変数変換 $x = a + \dfrac{b-a}{2}(u+1)$ により、$(-1, 1)$ 上の積分に変換できるので、

$$I = \int_{-1}^{1} f(x)\, dx$$

を考えれば十分である ((a, b) の場合に具体的にどうするか、「レポート課題 part 1 について」[10] に書いておいた)。

$$\int_a^b f(x)\, dx = \frac{b-a}{2} \int_{-1}^{1} f\left(a + \frac{b-a}{2}(u+1)\right)\, du.$$

これに対しては

$$\varphi_1(t) := \tanh\left(\frac{\pi}{2}\sinh t\right) \quad (t \in \mathbb{R}) \tag{12}$$

とおいて、変数変換 $x = \varphi_1(t)$ を施す。

$$\varphi_1'(t) = \frac{\pi}{2} \cdot \frac{\cosh t}{\cosh^2(\pi/2 \sinh t)}$$

であり、

$$\lim_{t \to \pm\infty} \varphi_1(t) = \pm 1 \quad (\text{複号同順}), \qquad \lim_{t \to \pm\infty} \varphi_1'(t) = 0.$$

念のため公式を書いておくと

$$I_h = \frac{\pi}{2} h \sum_{n=-\infty}^{\infty} f\left(\tanh\left(\frac{\pi}{2}\sinh(nh)\right)\right) \frac{\cosh nh}{\cosh^2(\pi/2 \sinh(nh))}.$$

[10] http://nalab.mind.meiji.ac.jp/~mk/complex2/report1-hint.pdf

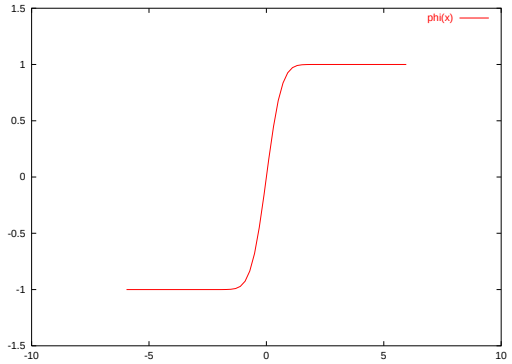

図 5: $\varphi(t) = \tanh\left(\frac{\pi}{2}\sinh t\right)$

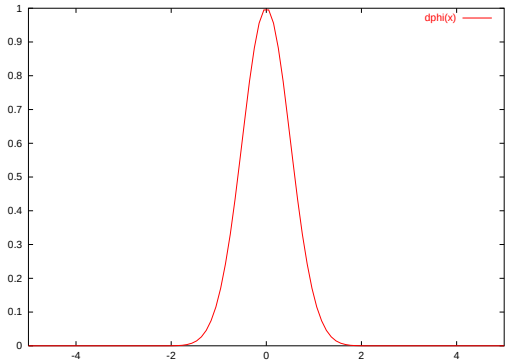

図 6: $\varphi'(t) = \dfrac{\pi}{2} \cdot \dfrac{\cosh t}{\cosh^2(\pi/2 \sinh t)}$

5.2.2 $\mathbb{R}$ 上の減衰の緩い関数の積分

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

において、$x \to \infty$ のとき f の減衰が緩い、例えば

$$\exists r > 1 \quad \text{s.t.} \quad f(x) \sim \frac{C}{|x|^r}$$

のような代数的な減衰の場合は、直接台形則を適用するのではなく、

$$\varphi_2(t) = \sinh\left(\frac{\pi}{2}\sinh t\right) \tag{13}$$

とおいて、変数変換 $x = \varphi_2(t)$ を施すことで、効率の良い公式が得られる。

$$\lim_{t\to\pm\infty} \varphi_2(t) = \pm\infty \quad \text{(複号同順)}.$$

この場合は、$t \to \pm\infty$ のとき $\varphi'(t)$ は減衰しないが、$f(\varphi_2(t))\varphi_2'(t)$ は二重指数関数的に減衰する。

念のため公式を書いておくと

$$I_h = \frac{\pi h}{2} \sum_{n=-\infty}^{\infty} f\left(\sinh\left(\frac{\pi}{2}\sinh nh\right)\right) \cosh nh \cosh\left(\frac{\pi}{2}\sinh nh\right).$$

($\mathbb{R}$ 上のそこそこ減衰の速い関数の場合はどうするか？と言っても、$f(x) = e^{-|x|}$ は解析的ではないし、$f(x) = e^{-x^2}$ は十分速いので、あまり考える必要はないか？$x = \sinh(t)$ くらいで良いか？)

5.2.3 半無限区間上の減衰の緩い関数の積分

(これは 2016/5/18 の授業では説明しなかった関数。)

$$I = \int_{0}^{\infty} f(x)\,dx$$

において f の減衰が代数的な場合は、

$$\varphi_3(t) := \exp(\pi \sinh t) \quad (t \in \mathbb{R}) \tag{14}$$

とおいて、変数変換 $x = \varphi_3(t)$ を施す。

$$\varphi_3'(t) = \pi \exp(\pi \sinh t) \cosh t$$

であり、

$$\lim_{t\to-\infty} \varphi_3(t) = 0, \quad \lim_{t\to\infty} \varphi_3(t) = \infty, \quad \lim_{t\to-\infty} \varphi_3'(t) = 0.$$

$t \to \infty$ のとき $\varphi_3'(t)$ は 減衰しないが、$f(\varphi_3(t))\varphi_3'(t)$ は二重指数関数的に減衰する。

念のため公式を書いておくと

$$I_h = \pi h \sum_{n=-\infty}^{\infty} f(\exp(\pi \sinh nh)) \exp(\pi \sinh nh) \cosh nh.$$

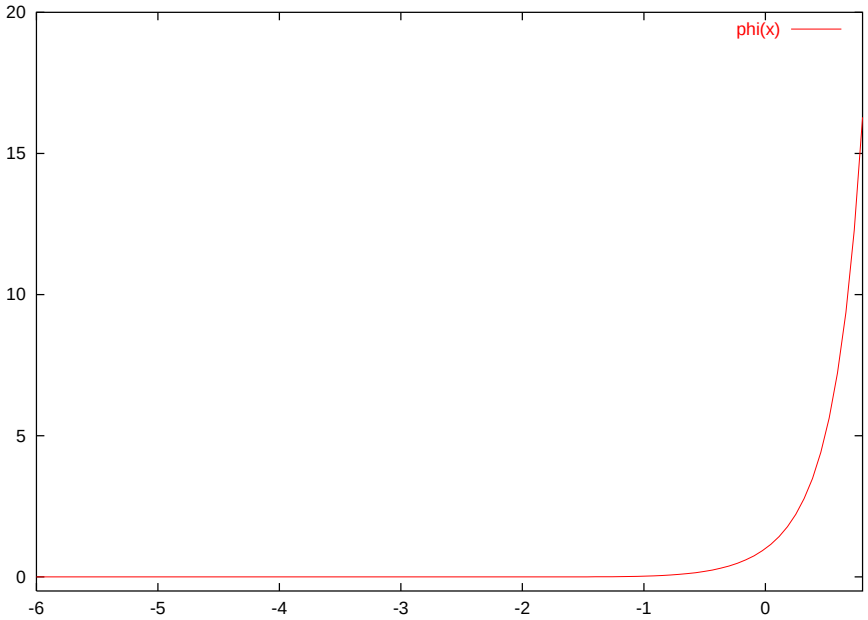

図 7: $\varphi(t) = \exp(\pi \sinh t)$

5.2.4 半無限区間上の一重指数関数的な減衰をする関数の積分

例えば

$$I = \int_0^{\infty} f(x)\, dx$$

において、

$$f(x) \sim f_1(x)e^{-x} \quad (x \to \infty), \quad (f_1(x) \text{ は減衰が代数的か、あるいは単に有界})$$

のような場合、

(15) $$\varphi_4(t) = \exp(t - \exp(-t))$$

とおいて、変数変換 $x = \varphi_4(t)$ を施してから台形公式を適用する。

$$\lim_{t\to-\infty} \varphi_4(t) = 0, \quad \lim_{t\to\infty} \varphi_4(t) = \infty, \quad \lim_{t\to-\infty} \varphi_4'(t) = 0.$$

$t \to \pm\infty$ のとき $\varphi_4'(t)$ は 減衰しないが、$f(\varphi(t))\varphi_4'(t)$ は二重指数関数的に減衰する。

公式は

$$I_h = h \sum_{n=-\infty}^{\infty} f\left(\exp(nh - \exp(-nh))\right) (1 + \exp(-nh)) \exp(nh - \exp(-nh)).$$

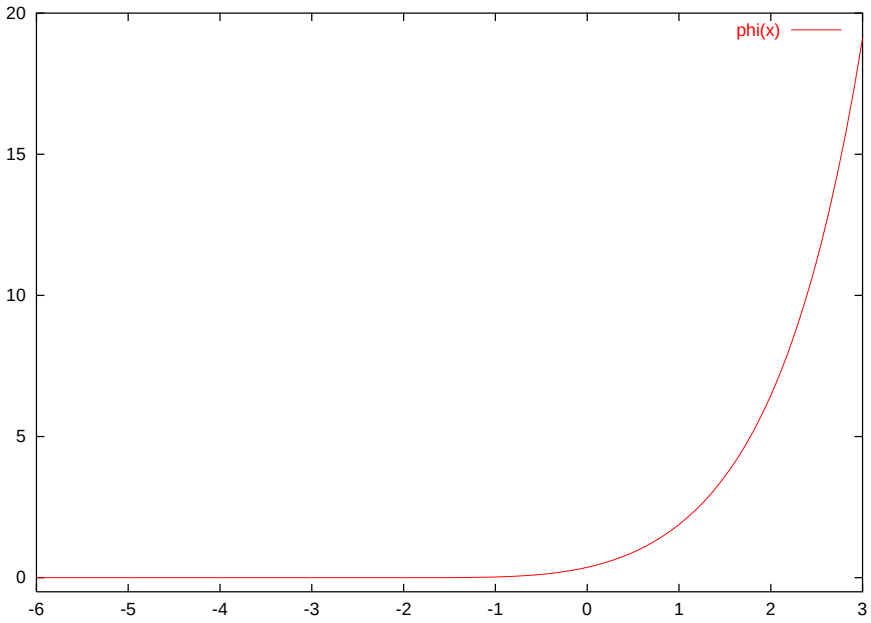

図 8: $\varphi(t) = \exp(t - \exp(-t))$

5.3 試してみよう

例 5.6 (端点の特異性に強い)

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

プログラム example6.c[11] を実行してみよう。

```
$ cc -o example6 example6.c
$ ./example6
test1 (sqrt(1-x^2) の積分)
h=1.000000, I_h=       1.7125198292703636, I_h-I=1.417235e-01
h=0.500000, I_h=       1.5709101233831166, I_h-I=1.137966e-04
h=0.250000, I_h=       1.5707963267997540, I_h-I=4.857448e-12
h=0.125000, I_h=       1.5707963267948970, I_h-I=4.440892e-16
h=0.062500, I_h=       1.5707963267948968, I_h-I=2.220446e-16
h=0.031250, I_h=       1.5707963267948972, I_h-I=6.661338e-16
h=0.015625, I_h=       1.5707963267948974, I_h-I=8.881784e-16
h=0.007812, I_h=       1.5707963267948941, I_h-I=-2.442491e-15
h=0.003906, I_h=       1.5707963267948979, I_h-I=1.332268e-15
h=0.001953, I_h=       1.5707963267948972, I_h-I=6.661338e-16
test2 (1/sqrt(1-x^2) の積分)
h=1.000000, I_h=       3.1435079763395439, I_h-I=1.915323e-03
h=0.500000, I_h=       3.1415926717394895, I_h-I=1.814970e-08
h=0.250000, I_h=       3.1415926194518016, I_h-I=-3.413799e-08
h=0.125000, I_h=       3.1415926318228000, I_h-I=-2.176699e-08
h=0.062500, I_h=       3.1415926343278699, I_h-I=-1.926192e-08
h=0.031250, I_h=       3.1415926326210668, I_h-I=-2.096873e-08
h=0.015625, I_h=       3.1415926323669527, I_h-I=-2.122284e-08
h=0.007812, I_h=       3.1415926327540080, I_h-I=-2.083579e-08
h=0.003906, I_h=       3.1415926312582507, I_h-I=-2.233154e-08
h=0.001953, I_h=       3.1415926319069589, I_h-I=-2.168283e-08
```

I については、$h = 1/8$, $N = 24$ で誤差が 10^{-16} 程度になっている。J については、$h = 1/2$, $N = 6$ で誤差が 10^{-8} 程度になっている (それ以上細かくしても改善されない)。シンプソン則

[11] http://nalab.mind.meiji.ac.jp/~mk/complex2/example6.c

の場合と比較してみること。N は試行錯誤で決めたが、h や N を自動的に選択するプログラムも作成可能である (詳しいことは省略)。■

なお、$1/\sqrt{1-x^2}$ の数値積分の精度が 10^{-8} 程度までしか上がらない理由は、± 1 に近い x に対して $\sqrt{1-x^2}$ を計算する際に、桁落ちが生じて精度が落ちるからである。きちんと桁落ちの対策をすると 10^{-16} 近くまで精度をあげることが出来る (2016 年度はレポート課題にするので、桁落ち対策プログラムの公開は遅らせる)。

例 5.7 (減衰の遅い関数)

$$I = \int_{-\infty}^{\infty} \frac{dx}{1+x^2} = \pi.$$

$x \to \pm\infty$ のとき、被積分関数の減衰はあまり速くないため、単純に台形公式を適用すると、なかなか精度の良い近似値が得られない (その数値例も用意すべきであったか…)。$x = \varphi_2(t)$ を用いた DE 公式を用いると次のようになる。($|t| \le 4$ まで計算している。)

```
$ cc -o example7 example7.c
$ ./example7
1 / (1 + x^2) の積分)
h=1.000000, I_h=       3.1435079789309333, I_h-I=1.915325e-03
h=0.500000, I_h=       3.1415926733057047, I_h-I=1.971591e-08
h=0.250000, I_h=       3.1415926535897944, I_h-I=1.332268e-15
h=0.125000, I_h=       3.1415926535897931, I_h-I=0.000000e+00
h=0.062500, I_h=       3.1415926535897931, I_h-I=0.000000e+00
h=0.031250, I_h=       3.1415926535897936, I_h-I=4.440892e-16
h=0.015625, I_h=       3.1415926535897927, I_h-I=-4.440892e-16
h=0.007812, I_h=       3.1415926535897913, I_h-I=-1.776357e-15
h=0.003906, I_h=       3.1415926535897913, I_h-I=-1.776357e-15
h=0.001953, I_h=       3.1415926535897976, I_h-I=4.440892e-15
```

$h = 1/4$, $N = 16$ で誤差 10^{-15} になっている。■

5.4 基本的な性質

- (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。

- 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 (IMT 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 [20] が詳しい (と言われるけれど、これは入手しにくいでの、誰かエッセンスを抽出して新しく解説を書くべきだよね)。

5.5 Mathematica で変数変換を見る

5.5.0 おまけ

次の φ_0 も使うことがある (説明を準備する？でも一重指数関数的にしか減衰しないような気がするが)。

$$\varphi_0(t) = \tanh(t).$$
$$\varphi_0(\mathbb{R}) = (-1, 1)\,.$$

次の ψ はぱっと見は φ_0 と似たグラフになるが、役に立たない。

$$\psi(t) = \frac{2}{\pi}\arctan(t).$$
$$\psi(\mathbb{R}) = (-1, 1)\,.$$

5.5.1 $\int_{-1}^{1} f(x)\,dx$ 用の変数変換

$$I = \int_{-1}^{1} f(x)\,dx$$

を計算するため。

$$\varphi_1(t) := \tanh\left(\frac{\pi}{2}\sinh(t)\right).$$
$$\varphi_1(\mathbb{R}) = (-1, 1)\,.$$

$F(t) := f(\varphi_1(t))\varphi_1'(t)$ のグラフは釣鐘形をしている (図 9)。

```
phi1[t_] := Tanh[Pi/2 Sinh[t]]
Plot[phi1[t], {t, -3, 3}]
Plot[phi1'[t], {t, -3, 3}]

f[x_]:=1/(x+2)
Plot[f[x],{x,-1,1}]
Plot[phi1'[t], {t, -3, 3}]
Plot[F[t], {t, -5, 5}, PlotRange -> All]

f[x_]:=1/Sqrt[1-x^2]
Plot[f[x], {x, -1, 1}]
F[t_] := f[phi1[t]] phi1'[t]
g1=Plot[F[t], {t, -3, 3}, PlotRange -> All]

g2=LogPlot[F[t], {t, -3, 3}]
```

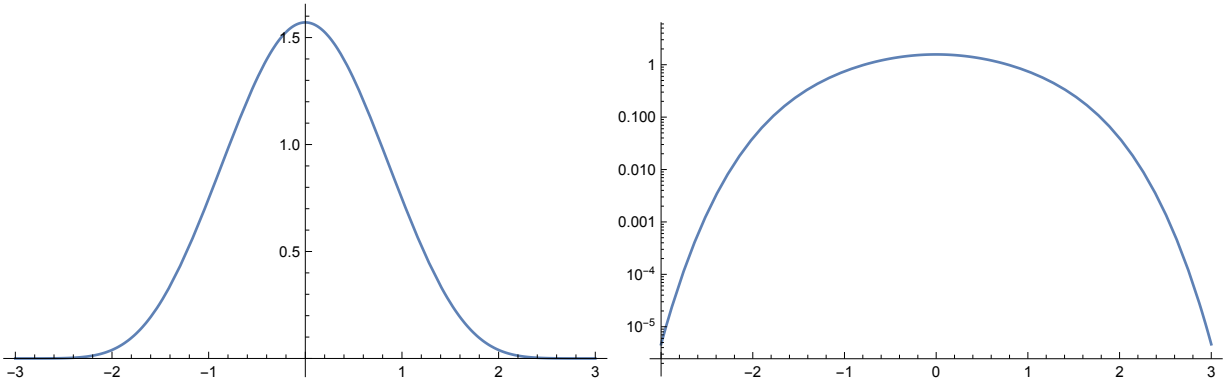

図 9: $F(t) = f\left(\varphi_1(t)\right)\varphi_1'(t)$ のグラフ (普通の目盛り、縦軸対数目盛

5.5.2 $\int_{-\infty}^{\infty} f(x)\,dx$ (ゆっくり減衰) 用の変数変換

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

を計算するため。ここで $f(x)$ の $x \to \pm\infty$ での減衰はゆっくりとする (例えば $f(x) = \dfrac{1}{1+x^2}$)。

$$\varphi_2(t) := \sinh\left(\frac{\pi}{2}\sinh(t)\right).$$

$$\varphi_2(\mathbb{R}) = \mathbb{R}.$$

```
phi2[t_] := Sinh[Pi/2 Sinh[t]]
Plot[phi2[t], {t, -2, 2}]
Plot[phi2'[t], {t, -2, 2}]

f[x_]:=1/(1+x^2)
Plot[f[x],{x,-5,5}]
F[t_]:=f[phi2[t]]phi2'[t]
Plot[F[t], {t, -5, 5}, PlotRange -> All]

LogPlot[F[t], {t, -5, 5}, PlotRange -> All]
```

やはり $F(t) := f(\varphi_2(t))\varphi_2'(t)$ のグラフは釣鐘形になる。F が急激に減衰することは、やや分かりにくい。導関数 φ_2' は、$t \to \pm\infty$ のとき急激に減衰する関数でなく、その反対に急激

に増大する。この点は φ_1 とは大きく異なる。それにもかかわらず、$f(\varphi_2(t))\,\varphi_2'(t)$ は急激に減衰する関数になる。これは少し考えると納得できるが (ぜひ考えてみよう)、ここでは論より証拠、目で見てみよう (図 10)。($t = \pm 5$ では、$10^{52.39}$ に $10^{-100.64}$ をかけて $10^{-48.25}$ 弱とか、出入りが実に激しい…実に豪快な技ですね。それにしても良くこういうことを思いつくものだ。)

```
gr=LogPlot[{phi2'[t],f[phi2[t]],F[t]},{t,-5,5},PlotRange->All,AspectRatio->1]
Export["phi2.eps", gr]
```

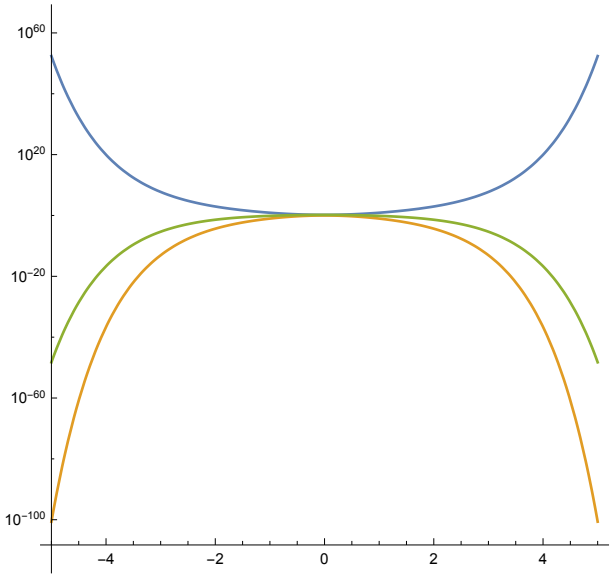

図 10: 上から順に φ_2', $f(\varphi_2(t))\,\varphi_2'(t)$, $f(\varphi_2(t))$ のグラフ (縦軸対数目盛)

5.5.3 $\int_0^\infty f(x)\,dx$ 用の変数変換

$$I = \int_0^\infty f(x)\,dx$$

を計算するため。$x \to +\infty$ のときの $f(x)$ の減衰はそこそこ速い (一重指数関数的) とする。

2016/5/18 の授業では、この関数に φ_3 という名前をつけたが、φ_4 という名前に変更することにした。

この場合は、

$$\varphi_4(t) := \exp(t - \exp(-t)) \tag{16}$$

という変数変換を用いる。これは $\mathbb{R}$ を $(0,\infty)$ に写す。

$$\varphi_4(\mathbb{R}) = (0,\infty).$$

結論を… $F(t) := f(\varphi_4(t))\varphi_4'(t)$ は釣鐘形。

```
phi4[t_] := Exp[t-Exp[-t]]
Plot[phi4[t], {t, -5, 5}]
Plot[phi4'[t], {t, -5, 5}]

f[x_]:=x^3 Exp[-x]
Plot[f[x],{x,0,10}]
F[t_]:=f[phi4[t]]phi2'[t]
g1=Plot[F[t], {t, -5, 5}, PlotRange -> All]

g2 = LogPlot[F[t], {t, -4.5, 7}, PlotRange -> All]
```

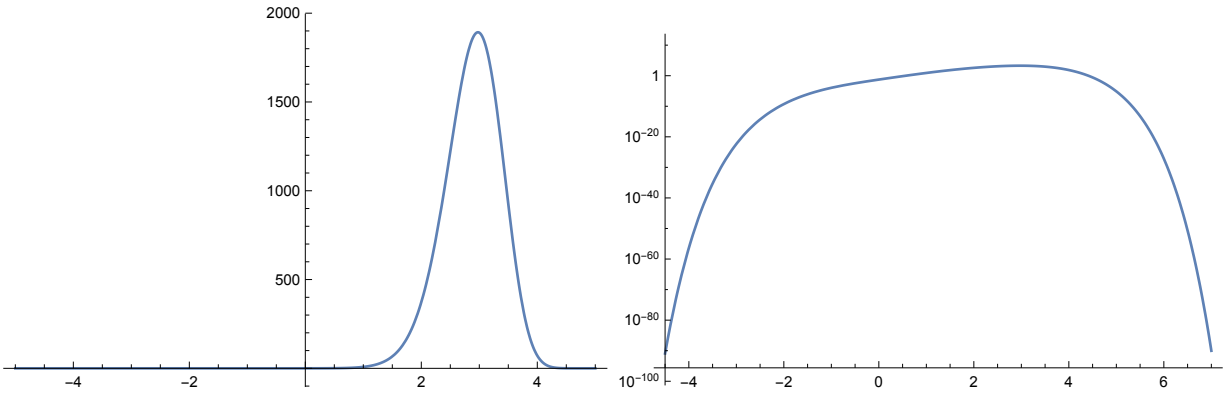

図 11: $F(t) = f(\varphi_4(t))\,\varphi_4'(t)$ のグラフ (普通の目盛り、縦軸対数目盛

グラフを見ると、この変数変換を使って数値積分するときは、$I_{h,N}$ でなく、

$$I_{h,N_1,N_2} = h \sum_{n=-N_1}^{N_2} f(\varphi_4(nh))\,\varphi_4'(nh)$$

を使うべきということも良く分かる。

6 高橋-森の数値積分誤差解析理論概説

(2016/5/25, 6/1)

原論文はあまり読みやすくないので、主に一松 [4] の第 5 章と森 [21] を参考に説明する。(この文書では、[4] の記号を採用したが、[4] の Φ, Λ, Ψ は、[21] ではそれぞれ Ψ, Ψ_n, Φ_n になっている。後のことを考えると、[21] の記号を用いるようにした方が良いかもしれない。途中で変更すると混乱するので、今年度はこのままにしておく。)

6.3 は、Mathematica を使うので授業向きかと思ったが、結構時間を食いそうなので (その割に地味で、自分で知りたい、確かめたいと思わないならば退屈だろう)、定理 6.4 を紹介した後は、簡単に済ませた。

当初は、杉原による誤差解析を解説する予定だったが、他の部分に思いの外時間がかかってしまったので今年度は割愛する。台形公式の誤差解析については付録 F で説明しておいた (DE 公式の誤差解析は、残念ながら今年度は完全スルーする)。

6.1 考える問題

定積分

$$I = \int_a^b f(x)\,dx \quad (-\infty \le a < b \le \infty) \tag{17}$$

か、あるいは少し一般化した定積分

$$I = \int_a^b f(x)p(x)\,dx. \tag{18}$$

を考える。p は重み関数と呼ばれる関数であり、(a,b) で連続で、$\int_a^b |p(x)|\,dx < \infty$ を満たすとする。今日の話は、$p(x) = 1$ として聴いていても構わない。

$f(x)$ は、$[a,b]$ を含む複素平面内の領域 D で正則な関数 f の実軸上の値とする。

多くの数値積分公式は次のような形をしている (すでに紹介した公式は皆そうである)。

$$I_n = \sum_{k=1}^{n} w_k f(x_k). \tag{19}$$

(ここで $x_k \in [a,b]$, $w_k \in \mathbb{R}$.)

この公式 (19) を用いたとき、誤差 $\Delta I_n = I - I_n$ を評価することを目標とする。

6.2 誤差の特性関数

Γ は、D 内の区分的に滑らかな単純閉曲線で、$[a,b]$ を内部に含み、正の向きに一周するとすると、Cauchy の積分公式により

$$f(x) = \frac{1}{2\pi i}\oint_\Gamma \frac{f(z)}{z-x}\,dz \quad (x \in [a,b]). \tag{20}$$

これを (18) に代入すると

$$I = \int_a^b \left(\frac{1}{2\pi i}\oint_\Gamma \frac{f(z)}{z-x}\,dz\right) p(x)\,dx.$$

積分の順序を変えて

$$I = \frac{1}{2\pi i}\oint_\Gamma \left(\int_a^b \frac{p(x)}{z-x}dx\right) f(z)\,dz.$$

すなわち

$$I = \frac{1}{2\pi i}\oint_\Gamma \Phi(z)f(z)\,dz, \quad \Phi(z) := \int_a^b \frac{p(x)}{z-x}\,dx. \tag{21}$$

($p(x) = 1$ の場合は、$\Phi(z) = \mathrm{Log}\,\dfrac{z-a}{z-b}$ である。Φ を p の Hilbert 変換と呼ぶらしい。)

同様に、(20) を (19) に代入して、$\int$ と $\sum$ の順序を変えて

$$I_n = \frac{1}{2\pi i}\oint_\Gamma \Lambda(z)f(z)\,dz, \quad \Lambda(z) := \sum_{k=1}^{n} \frac{w_k}{z-x_k}. \tag{22}$$

(21), (22) を辺々引き算して次を得る。

定理 6.1 (高橋-森理論の基本定理) 積分 (18) を数値積分 (19) で計算したとき、

$$I - I_n = \frac{1}{2\pi i} \oint_\Gamma \Psi(z) f(z)\, dz, \quad \Psi(z) := \Phi(z) - \Lambda(z). \tag{23}$$

Ψ は数値積分公式 (19) の**誤差の特性関数**と呼ばれる。Φ は $\mathbb{C} \setminus [a,b]$ で一価正則である。

Φ, Λ は、重み関数 $p(x)$ と数値積分公式 (19) だけで定まり、個々の被積分関数 f には依存しない。従って、Ψ もそうである。このことから、誤差の特性関数 Ψ を調べることで、数値積分公式 (19) そのものの良し悪しが議論できることが期待される。

Λ は、Φ の有理関数による近似であり、その差が誤差の特性関数 Ψ ということである。

定理 6.1 の無限区間台形公式版を書いておく。積分路は図に描くこと。

命題 6.2 (高橋-森理論の基本定理 (無限区間版)) $d, h > 0$. f は $\{z \in \mathbb{C} \mid |\mathrm{Im}\, z| < d\}$ で正則、$\mathbb{R}$ で積分可能とする。

$$I := \int_{-\infty}^{\infty} f(x)\, dx, \quad I_h := h \sum_{n=-\infty}^{\infty} f(nh)$$

とおくとき、$0 < \varepsilon < d$ を満たす ε に対して、Γ_ε を $\mathrm{Im}\, z = -\varepsilon$ を左から右に進む直線と、$\mathrm{Im}\, z = \varepsilon$ を右から左に進む直線との和とすると、

$$I - I_h = \frac{1}{2\pi i} \int_{\Gamma_\varepsilon} \Psi_h(z) f(z)\, dz.$$

ただし

$$\Psi_h(z) := \Phi(z) - \pi \cot \frac{\pi z}{h}, \quad \Phi(z) := \begin{cases} -\pi i & (\mathrm{Im}\, z > 0) \\ +\pi i & (\mathrm{Im}\, z < 0). \end{cases}$$

($\Phi(z)$ の定義、これまで i が抜けていたのを修正した。2016/7/7)

証明をしようとして詰まってしまった。上の定理は [4] から採ったものだが、証明は書いていない。類似した命題を載せている [2] では、条件が追加してある。その条件を仮定すると確かに証明できる。

- $0 \le \varepsilon < d$ を満たす任意の ε に対して
$$\int_{-\infty}^{\infty} (|f(x + i\varepsilon)| + |f(x - i\varepsilon)|)\, dx < \infty.$$
- $0 \le \varepsilon < d$ を満たす任意の ε に対して
$$\lim_{x \to \pm\infty} \int_{-\varepsilon}^{\varepsilon} f(x + iy)\, dy = 0.$$

(2016/7/13 追記)

余談 6.3 有界区間 (a,b), $p(x) = 1$ のときの $\Phi(z) = \mathrm{Log} \dfrac{z-a}{z-b}$ に相当するのが、$\Phi(z) = \begin{cases} -\pi i & (\mathrm{Im}\, z > 0) \\ +\pi i & (\mathrm{Im}\, z < 0) \end{cases}$. 積分公式 $I_n = \displaystyle\sum_{k=1}^{n} w_k f(x_k)$ に付随する $\Lambda(z) = \displaystyle\sum_{k=1}^{n} \frac{w_k}{z - x_k}$ に相当するのが、I_h に付随する $\pi \cot \dfrac{\pi z}{h} = h \left[\dfrac{1}{z} + \displaystyle\sum_{n=1}^{\infty} \left(\frac{1}{z - nh} + \frac{1}{z + nh} \right) \right]$ ということである。 ■

(授業ではもう少し丁寧に説明したので、そのときのメモの内容を反映させること。)

6.3 古典的数値積分公式は、Φ の有理関数 Λ による近似に対応すること

(もっとピッタリの見出しはないか？)

簡単のため $[a,b]$ を有界区間とする ($a,b\in\mathbb{R}$, $a<b$ ということ)。Ψ は $\mathbb{C}\setminus[a,b]$ で正則だから、$c=\dfrac{a+b}{2}$ を中心とする Laurent 展開

$$\Psi(z)=\sum_{m=1}^{\infty}\frac{a_m}{(z-c)^m}\quad(|z-c|>\frac{b-a}{2}) \tag{24}$$

が出来る。主部 ($(z-c)$ の正の冪の項を集めた部分) も定数項もないのは、$\lim_{z\to\infty}\Psi(z)=0$ であるからである[12]。

$\lim_{z\to\infty}\Psi(z)=0$ となるのは、$\Phi(z)$, $\Lambda(z)$ の両方がそういう性質を持つからである。Λ については定義式の形から明らか (各項が 0 に収束する)。Φ の方は

$$|\Phi(z)|\le\frac{1}{\min\limits_{x\in[a,b]}|z-x|}\int_a^b|p(x)|\,dx\to0\quad(|z|\to\infty).$$

定理 6.4 数値積分公式 (19) が、ℓ 次以下の任意の多項式 $f(z)$ に対して、積分 (18) の正しい値を与えるためには、

$$a_1=a_2=\cdots=a_{\ell+1}=0 \tag{25}$$

が必要十分である。

この定理の結論が成り立つとき、数値積分公式 (19) は $(\ell+1)$ 位の公式という。

証明　ℓ 次以下の多項式全体の空間は、$(z-c)^k$ $(k=0,1,\cdots,\ell)$ を基底に持つ。

$$a_{k+1}=\frac{1}{2\pi i}\int_\Gamma(z-c)^k\,\Psi(z)\,dz$$

であるから、

$a_1=a_2=\cdots=a_{\ell+1}=0\Leftrightarrow\ell$ 次以下の任意の多項式 $f(z)$ に対して、$\displaystyle\int_\Gamma f(z)\Psi(z)\,dz=0.$ ■

一松 [4] には、

> 要するに数値積分公式 (2) を求めることは、$|z|$ が大きいところで、$\Phi(z)$ を有理関数 $\Lambda(z)$ で近似することと解釈されるのだから、ローラン展開の係数がなるべく沢山消えるというのは、$\Psi(z)$ が $|z|$ が大になると早く 0 に収束し、従って誤差が小さくなることを期待させることになるのである。

とある。誤差が小さいというけれど、どこら辺での話なのだろう？(まだ良く理解できていない…)

[12] [4] には、「$|z|\to\infty$ のとき、$\Psi(z)\to0$ だから、定数項は 0 である。」と書いてあるが、主部と定数項が 0 というべきであろう。

Newton-Cotes の公式

(ここは授業でははしょって説明した。標本点を等間隔に取ることにすると、自然に、すでに紹介済みの補間型数値積分公式である、台形公式、Simpson 公式、さらにはまだ紹介していない補間型数値積分公式も導かれる。それを確認している。割と計算が面倒である。Mathematica を使うと、かなり自動化出来て、Mathematica 様々。)

例 6.5 (Newton-Cotes 公式, $n = 2, 3, 4, 5$**)** $[a, b] = [-1, 1]$, $p(x) = 1$ とする。$c = 0$, $\Phi(z) = \mathrm{Log}\,\dfrac{z+1}{z-1}$ である。区間の $n-1$ 等分点を

$$x_j = -1 + (j-1)\frac{2}{n-1} = -\frac{n-2j+1}{n-1} \quad (j = 1, \cdots, n)$$

とおく。なるべく高次の多項式まで $I = I_n$ が成り立つように重み w_j を定めたい。$w_1, \cdots, w_n$ を自由に選べるので、$\ell = n-1$ 次多項式まで $I = I_n$ と出来ると期待される。

$$\Lambda(z) = \frac{G_n(z)}{F_n(z)}$$

と書ける。ただし

$$F_n(z) := \prod_{j=1}^{n} (z - x_j) = (z+1)\left(z + \frac{n-3}{n-1}\right) \cdots \left(z - \frac{n-3}{n-1}\right)(z-1)$$

であり、$G_n(z)$ は $n-1$ 次の未知多項式である。$F_n(z)$ は、n が偶数ならば偶関数, n が奇数ならば奇関数である。

補題 6.6 $\Phi(z) = \mathrm{Log}\,\dfrac{z+1}{z-1}$ は $1 < |z| < \infty$ で次のように Laurent 展開できる。

$$\Phi(z) = \frac{2}{z} + \frac{2}{3z^3} + \frac{2}{5z^5} + \cdots + \frac{2}{(2m+1)\,z^{2m+1}} + \cdots \quad (1 < |z| < \infty). \tag{26}$$

証明 Φ は $|z| > 1$ で正則であるので、$g(w) := \Phi(1/w)$ とおくと、g は $0 < |w| < 1$ で正則であるが、実は $|w| < 1$ で

$$\begin{aligned} g(w) = \Phi(1/w) &= \mathrm{Log}\,\frac{1/w+1}{1/w-1} = \mathrm{Log}\,\frac{1+w}{1-w} = \mathrm{Log}(1+w) - \mathrm{Log}(1-w) \\ &= \sum_{n=1}^{\infty} \frac{(-1)^{n-1}}{n} w^n - \sum_{n=1}^{\infty} \frac{(-1)^{n-1}}{n} (-w)^n = \sum_{n=1}^{\infty} \frac{(-1)^{n-1}+1}{n} w^n \\ &= 2\sum_{k=1}^{\infty} \frac{w^{2k-1}}{2k-1}. \end{aligned}$$

と Taylor 展開出来る。ゆえに

$$\Phi(z) = g\left(\frac{1}{z}\right) = 2\sum_{k=1}^{\infty} \frac{1}{(2k-1)z^{2k-1}}. \blacksquare$$

問 1. $|w|<1$ のとき、$\mathrm{Log}\,\dfrac{1+w}{1-w}=\mathrm{Log}(1+w)-\mathrm{Log}(1-w)$ が成り立つことを示せ。

実は、任意の n に対して、$G_n(z)$ を上手く選んで

$$\Phi(z)-\frac{G_n(z)}{F_n(z)}=O\left(\frac{1}{z^{n+1}}\right)\quad(|z|\to\infty)$$

となるように出来る。この条件は

$$\Phi(z)F_n(z)-G_n(z)=O\left(\frac{1}{z}\right)\quad(|z|\to\infty)$$

ということだから、$\Phi(z)F_n(z)$ を z の冪の和の形に表し、そのうちの非負指数の部分を $G_n(z)$ とすれば良い。

$n=2$ の場合　$F_2(z)=(z+1)(z-1)$.

$$F_2(z)\,\Phi(z)=2\left(\frac{1}{z}+\frac{1}{3z^3}+\frac{1}{5z^5}+\cdots\right)\left(z^2-1\right)=2z-\frac{4}{3z}+\cdots$$

であるから、$G_2(z)=2z$. $\Lambda(z)$ の部分分数分解は

$$\Lambda(z)=\frac{G_2(z)}{F_2(z)}=\frac{2z}{(z+1)(z-1)}=\frac{1}{z+1}+\frac{1}{z-1}.$$

ゆえに $x_1=-1$, $w_1=1$, $x_2=1$, $w_2=1$. これから

$$I_2=\sum_{k=1}^{2}f(x_k)w_k=f(-1)+f(1).$$

これは一般の区間 $[a,b]$ の場合では、

$$I_2=\frac{b-a}{2}\left(f(a)+f(b)\right)$$

に対応している (台形公式)。また

$$\Lambda(z)=\sum_{k=0}^{\infty}\frac{1+(-1)^k}{z^{k+1}}=2\left(\frac{1}{z}+\frac{1}{z^3}+\frac{1}{z^5}+\cdots\right)\quad(1<|z|<\infty).$$

これから、

$$\Psi(z)=\Phi(z)-\Lambda(z)=2\left[\frac{-2}{3z^3}+\frac{-4}{5z^5}+\frac{-6}{7z^7}+\cdots\right].$$

ゆえに $a_1=a_2=0$. これは 2 位の公式であることを示す。

$n=3$ の場合　$F_3(z)=(z+1)z\,(z-1)=z^3-z$.

$$\begin{aligned}F_3(z)\,\Phi(z)&=2\left(\frac{1}{z}+\frac{1}{3z^3}+\frac{1}{5z^5}+\cdots\right)\left(z^3-z\right)\\&=2z^2-\frac{4}{3}+2\left[\left(\frac{1}{5}-\frac{1}{3}\right)\frac{1}{z^2}+\left(\frac{1}{7}-\frac{1}{5}\right)\frac{1}{z^4}+\cdots\right].\end{aligned}$$

ゆえに $G_3(z)=2z^2-\dfrac{4}{3}$. $\Lambda(z)$ の部分分数分解は

$$\Lambda(z)=\frac{G_3(z)}{F_3(z)}=\frac{2z^2-\frac{4}{3}}{(z+1)z(z-1)}=\frac{1}{3}\left(\frac{1}{z+1}+\frac{4}{z}+\frac{1}{z-1}\right).$$

ゆえに $x_1=-1$, $w_1=\dfrac{1}{3}$, $x_2=0$, $w_2=\dfrac{4}{3}$, $x_3=1$, $w_3=\dfrac{1}{3}$. これから

$$I_3=\sum_{k=1}^{3}f(x_k)w_k=\frac{1}{3}\left(f(-1)+4f(0)+f(1)\right).$$

これは一般の区間 $[a,b]$ の場合では、

$$I_3=\frac{b-a}{3}\left(f(a)+4f\left(\frac{a+b}{2}\right)+f(b)\right)$$

に対応している (Simpson 公式)。また

$$\Lambda(z)=\frac{2}{z}+\frac{2}{3z^3}+\frac{2}{3z^5}+\frac{2}{3z^5}+\cdots\quad(1<|z|<\infty).$$

これから

$$\Psi(z)=\Phi(z)-\Lambda(z)=2\left[\left(\frac{1}{5}-\frac{1}{3}\right)\frac{1}{z^5}+\left(\frac{1}{7}-\frac{1}{3}\right)\frac{1}{z^7}+\cdots\right]=-\frac{4}{15z^5}-\frac{8}{21z^7}-\cdots$$

ゆえに $a_1=a_2=a_3=a_4=0$. これは 4 位の公式であることを示す。

$n=4$ の場合　(ここから先は、あまり実用にならない公式なので、理論を確かめる興味くらいしかない。計算はかなり面倒になってくるが、Mathematica を使えるならば簡単である。)

$$F_4(z)=(z+1)\left(z+\frac{1}{3}\right)\left(z-\frac{1}{3}\right)(z-1)=z^4-\frac{10}{9}z^2+\frac{1}{9}.$$

$$F_4(z)\Phi(z)=2\left(\frac{1}{z}+\frac{1}{3z^3}+\frac{1}{5z^5}+\frac{1}{7z^7}+\cdots\right)\left(z^4-\frac{10}{9}z^2+\frac{1}{9}\right)=2z^3-\frac{14}{9}z+\cdots$$

であるから、$G_4(z)=2z^3-\dfrac{14}{9}z$.

$$\Lambda(z)=\frac{G_4(z)}{F_4(z)}=\frac{2z^3-\frac{14}{9}z}{(z+1)\left(z+\frac{1}{3}\right)\left(z-\frac{1}{3}\right)(z-1)}=\frac{1}{4}\cdot\frac{1}{z+1}+\frac{9}{4}\cdot\frac{1}{z+1/3}+\frac{9}{4}\cdot\frac{1}{z-1/3}+\frac{1}{4}\cdot\frac{1}{z-1}.$$

ゆえに

$$x_1=-1,\quad w_1=\frac{1}{4},\quad x_2=-1/3,\quad w_2=\frac{9}{4},\quad x_3=1/3,\quad w_3=\frac{9}{4},\quad x_4=1,\quad w_4=\frac{1}{4}.$$

$$I_4=\frac{1}{4}\left(f(-1)+9f\left(-\frac{1}{3}\right)+9f\left(\frac{1}{3}\right)+f(1)\right).$$

これは一般の区間 $[a,b]$ の場合には

$$I_4=\frac{b-a}{4}\left(f(a)+9f\left(\frac{2a+b}{3}\right)+9f\left(\frac{a+2b}{3}\right)+f(b)\right)$$

に対応している。また

$$\Lambda(z)=\frac{2}{z}+\frac{2}{3z^3}+\frac{14}{27z^5}+\cdots$$

$$\Phi(z)-\Lambda(z)=-\frac{16}{135z^5}-\frac{368}{1701z^7}-\cdots$$

ゆえに $a_1=a_2=a_3=a_4=0$. これは 4 位の公式であることを示す ($n=3$ の場合よりも複雑になったのに、位数は改善されない)。

$n=5$ **の場合** $G_5(z)=2z^4-\dfrac{11}{6}z^2+\dfrac{1}{15}$,

$$\Lambda(z)=\frac{G_5(z)}{F_5(z)}=\frac{1}{90}\left(\frac{14}{z+1}+\frac{64}{z+1/2}+\frac{24}{z}+\frac{64}{z-1/2}+\frac{14}{z-1}\right).$$

$$\Psi(z)=\Phi(z)-\Lambda(z)=-\frac{1}{21z^7}-\frac{17}{180z^9}-\frac{23}{176z^{11}}-\cdots$$

ゆえに位数は 6. ■

Mathematica の `Series[式, {変数, 点, 次数}]` で、Taylor 展開や Laurent 展開の最初の有限項の計算が出来る。点として `Infinity` が指定できて、無限遠点の周りの Laurent 展開が計算出来る。有限項しか得られないが、それで一般項を推測出来れば、`Sum[ ]` を使って、その推測が正しいかチェックすることが出来る。

「複素関数」でやった計算を Mathematica にさせてみる

```
f[z_]:=1/(3+z)
Series[f[z],{z,0,4}]
Series[f[z],{z,Infinity,4}]
```

これで 0 の周りの Taylor 展開

$$\frac{1}{3+z}=\frac{1}{3}-\frac{z}{9}+\frac{z^2}{27}-\frac{z^3}{81}+\frac{z^4}{243}+O(z^5),$$

∞ の周りの Laurent 展開

$$\frac{1}{3+z}=\frac{1}{z}-\frac{3}{z^2}+\frac{9}{z^3}-\frac{27}{z^4}+O\left(\frac{1}{z^5}\right)$$

が得られる。

なお、`Apart[]` で部分分数分解が出来る。

例 6.5 を Mathematica でどう計算したか

```
Phi[z_]:=Log[(z+1)/(z-1)]
Series[Phi[z],{z,Infinity,10}]
F[n_,z_]:=Product[(z-(-1+2(j-1)/(n-1))),{j,1,n}]

Series[Phi[z]F[3,z],{z,Infinity,10}]
```

この結果を読み取る。

```
G3[z_]:=2z^2-4/3
Apart[G3[z]/F[3,z]]
Series[Phi[z]-G3[z]/F[3,z],{z,Infnitiy,10}]
```

もう少し自動化出来る。

```
G[n_,z_]:=Normal[Series[Phi[z] F[n,z],{z,Infinity,0}]]
G[3,z]
Apart[G[3,z]/F[3,z]]
Series[Phi[z]-G[3,z]/F[3,z],{z,Infnitiy,10}]
```

例 6.7 (26) の右辺の最初の項 $2/z$ を取ると、つまり、$\Lambda(z) = \frac{2}{z}$ $(F(z) = z,\ G(z) = 2)$ とすると、

$$x_1 = 0, \quad w_1 = 2.$$

これは

$$I_1 = 2f(0) = (b-a)f\left(\frac{a+b}{2}\right) \tag{27}$$

ということを意味している。これは中点公式である。 ■

6.4 有理関数の積分への応用

数値積分公式の誤差を、誤差の特性関数の積分の形に表した。そこから先の誤差解析には、一般には**鞍点法**と呼ばれる積分の近似計算法が有効であるが、その話をすることは諦め (これについては、森 [22], [1] が参考になる)、ここでは簡単な (被積分関数が有理関数である) 場合だけを考える。この場合は留数定理だけで解決する。

$[a,b]$ は実軸上の有限区間 $(a,b \in \mathbb{R},\ a<b)$, $p(x)=1$ とする。このとき、Φ は

$$\Phi(z) = \int_a^b \frac{1}{z-x}\,dx = \operatorname{Log}\frac{z-a}{z-b} \quad (z \in \mathbb{C}\setminus[a,b]). \tag{28}$$

命題 6.8 $[a,b]$ は $\mathbb{R}$ 内の閉区間, $P(z), Q(z) \in \mathbb{C}[z]$, $\deg P(z) \geq \deg Q(z)+1$, $(\forall x \in [a,b])$ $P(x) \neq 0$, $f(z) := \dfrac{Q(z)}{P(z)}$ とするとき、

$$\int_a^b f(x)\,dx = -\sum_{j=1}^{m} \operatorname{Res}(f\Phi; c_j)\,.$$

ただし、Φ は (28) で定義した関数であり、$c_1, \ldots, c_m$ を $P(z)$ のすべての零点とする。

(一松 [4] の p. 118 の定理 5.3 には、$\displaystyle\int_a^b f(x)\,dx = \sum_{j=1}^{m} \operatorname{Res}(f\cdot\Phi;\alpha_j)$ と書いてあるが、これは誤植で、右辺は負号 $-$ が落ちていると考えられる。)

証明 (講義をするときは、大きな図を黒板に描くこと。) $I := \displaystyle\int_a^b f(x)\,dx$ とおく。$c_1, \cdots, c_m$ は $[a,b]$ 上にはないので、$[a,b]$ を正の向きに一周す区分的に滑らかな単純閉曲線 Γ を、曲線上と曲線の囲む範囲に $c_1, \cdots, c_m$ が存在しないように選べる。既に紹介した論法で次が得られる。

$$I = \frac{1}{2\pi i}\int_\Gamma f(z)\Phi(z)\,dz. \tag{29}$$

R が十分大きな正数ならば、円 C_R: $|z| = R$ は、その内部にすべての c_j と、Γ を含む。このとき、留数定理から

$$\frac{1}{2\pi i}\int_{C_R} f(z)\Phi(z)\,dz - I = \frac{1}{2\pi i}\int_{C_R-\Gamma} f(z)\Phi(z)\,dz = \sum_{j=1}^{m}\operatorname{Res}(f\Phi; c_j)\,.$$

$z \to \infty$ のとき $\Phi(z) \to 0$ であるから、十分遠方では、$|f(z)\Phi(z)| \le \dfrac{M}{|z|^2}$ が成り立つので、$R \to \infty$ とするとき、$\dfrac{1}{2\pi i}\displaystyle\int_{C_R} f(z)\Phi(z)\,dz \to 0$. ゆえに

$$I = -\sum_{j=1}^{m} \mathrm{Res}\,(f\Phi; c_j)\,. \blacksquare$$

系 6.9 $[a,b]$ は $\mathbb{R}$ 内の閉区間, $P(z), Q(z) \in \mathbb{C}[z]$, $\deg P(z) \ge \deg Q(z) + 1$, $(\forall x \in [a,b])$ $p(x) \ne 0$, $f(z) := \dfrac{q(z)}{P(z)}$ とするとき、$I = \displaystyle\int_a^b f(x)\,dx$ を $I_n = \sum_{k=1}^{n} w_k f(x_k)$ で数値積分すると

$$I - I_n = -\sum_{j=1}^{m} \mathrm{Res}\,(f\cdot\Psi; c_j)\,, \quad \Psi(z) := \mathrm{Log}\,\frac{z-a}{z-b} - \sum_{k=1}^{n} \frac{w_k}{z-x_k}.$$

ただし $c_1, \ldots, c_m$ は $P(z)$ のすべての零点とする。

証明　命題 6.8 の証明と同様にして、

$$I_n = -\sum_{j=1}^{m} \mathrm{Res}\,(f\Lambda; c_j)\,.$$

ゆえに

$$\begin{aligned} I - I_n &= -\sum_{j=1}^{m} \left(\mathrm{Res}\,(f\Phi; c_j) - \mathrm{Res}\,(f\Lambda; c_j)\right) = -\sum_{j=1}^{m} \mathrm{Res}\,(f(\Phi-\Lambda); c_j) \\ &= -\sum_{j=1}^{m} \mathrm{Res}\,(f\Psi; c_j)\,. \blacksquare \end{aligned}$$

($f\cdot\Psi$ の留数を計算出来るくらいならば、$f\cdot\Phi$ の留数を計算して、I を求めることが出来る場合が多いはずである。系 6.9 は、何のために使うのか、ちょっと変な感じがしないでもない。しかし、誤差のメカニズムが見える式で、参考になる。実際に後の数値例 6.12 は面白い (初めて理解したとき少し震えました)。)

余談 6.10 (29) をこれまでは、Cauchy の積分公式と積分の順序交換を用いて証明したが、

$$\lim_{\varepsilon\to+0}\left(\mathrm{Log}\,\frac{z-a}{z-b}\bigg|_{z=x+i\varepsilon} - \mathrm{Log}\,\frac{z-a}{z-b}\bigg|_{z=x-i\varepsilon}\right) = -2\pi i$$

を用いて、積分路の変形で証明することも出来る。というか、そういう解説をしてある文書が割と多い。 ■

問 2. c が f の 1 位の極ならば、$\mathrm{Res}(f\Phi; c) = Res(f; c)\Phi(c)$ であることを示せ。

例 6.11 (命題 6.8 の直接的な応用)

$$I = \int_{-1}^{1} \frac{dx}{x^4+1}.$$

この場合は $\Phi(z)=\operatorname{Log}\dfrac{z+1}{z-1}$. c が $z^4+1=0$ の根であるとき、c は $f(z):=\dfrac{1}{z^4+1}$ の 1 位の極であり、

$$\operatorname{Res}(f\Phi;c)=\Phi(c)\operatorname{Res}(f;c)=\Phi(c)\left.\frac{1}{(z^4+1)'}\right|_{z=c}=\Phi(c)\frac{1}{4c^3}=\Phi(c)\frac{c}{4c^4}=-\frac{c\Phi(c)}{4}$$

であるから

$$\begin{aligned}I&=-\sum_{c=\frac{1+i}{\sqrt2},\frac{-1+i}{\sqrt2},\frac{-1-i}{\sqrt2},\frac{-1-i}{\sqrt2}}\operatorname{Res}(f\Phi;c)=\frac14\sum_{c=\frac{1+i}{\sqrt2},\frac{-1+i}{\sqrt2},\frac{-1-i}{\sqrt2},\frac{-1-i}{\sqrt2}}c\Phi(c)\\&=\frac{1}{\sqrt2}\operatorname{Log}\left(1+\sqrt2\right)+\frac{\pi}{2\sqrt2}.\end{aligned}$$

最後のところの計算は実は少し面倒で、あまり試験向きではない。教師として残念だ (笑)。Mathematica にやらせてみたら $\dfrac{\pi+\operatorname{arctanh}(2\sqrt2/3)}{2\sqrt2}$ となった。一応一致しているようだ (ちょっと分かりにくい…)。 ■

6.5 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}

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

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

これは

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

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

とおくと、

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

と書ける。

$$\Phi(z)=\operatorname{Log}\frac{z+1}{z-1},\quad \Psi(z)=\Phi(z)-\Lambda(z)$$

として、$|\Psi(z)|$ $(-4\le\operatorname{Re}z\le4,\ -4\le\operatorname{Im}z\le4)$ の等高線を描いてみる。

(これは森 [1] にある図と見比べるためである。)

プログラムは、付録 G を見よ。

z 平面において、積分区間 $[a,b]$ から遠ざかると、$|\Psi(z)|$ 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、6.3 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。

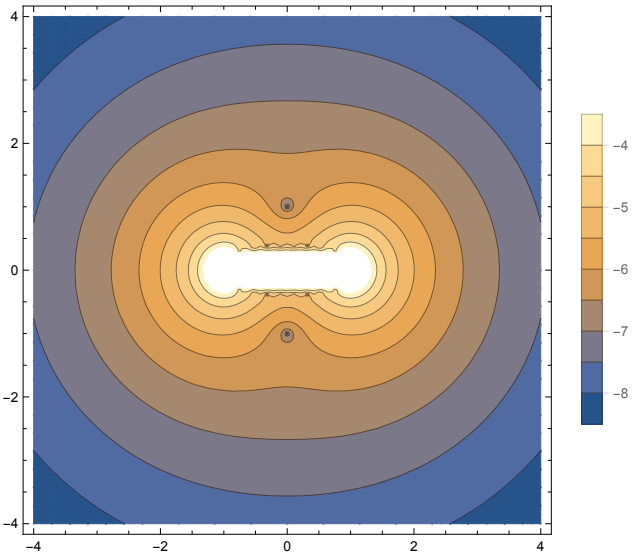

図 12: 21 点複合シンプソン公式の誤差の特性関数 (絶対値の常用対数)

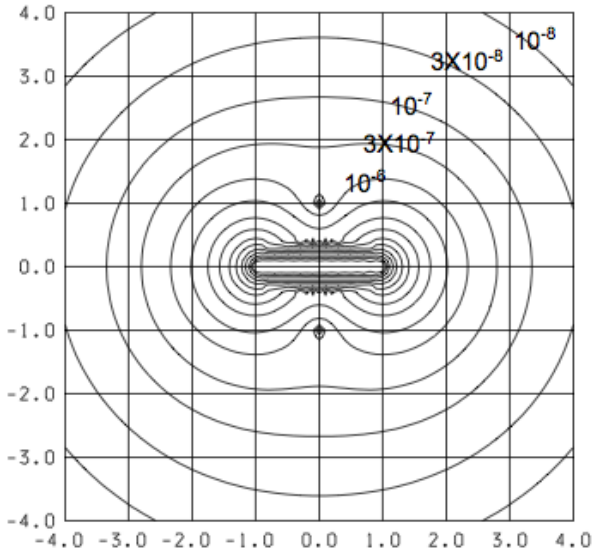

図 13: 森 [21] から $|\Phi_n(z)|$ for Simpson's formula ($h = 0.1$)

例 6.12 (森 [21] 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

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

系 6.9 を用いて

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

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 [22] 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。

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 $|\Psi(z)| = 10^{-16}$ 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。

比較のために DE 公式の誤差の特性関数の等高線図も載せたかったのですが、時間切れです。そのうち載せますが、この際、レポート課題にしてしまおうと思います。それを描くプログラム (言語は何でも良い) とその実行結果をレポートとして提出したら、高く評価します (締め切りは 2016 年 6 月末日)。

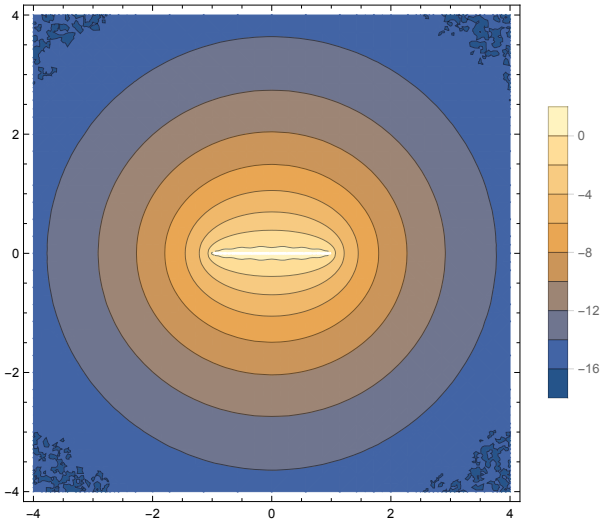

図 14: $(-1,1)$ における 8 次 Gauss-Legendre 公式の誤差の特性関数 (絶対値の常用対数)

6.6 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、

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

(これまで単に Λ と書いて来たが、h に依存するので、Λ_h と書くことにする。)

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。

$$\Lambda_h(z) = \lim_{N\to\infty} \sum_{n=-N}^{N} \frac{h}{z-nh} \overset{\text{本当?}}{=} \pi \cot \frac{\pi z}{h}.$$

($\pi \cot \pi z = \lim_{N\to\infty} \sum_{n=-N}^{N} \frac{1}{z-n}$ という公式の説明をまだやっていなかった。直観的な説明をすると、Λ は (多分) 極が nh, 留数が h という条件を満たすだろう。それについては $\pi \cot \pi z$ も同じで、実は両者は一致することが証明できる。つまり $\overset{\text{本当?}}{=}$ が証明できる)

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

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

(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 6.10 と見比べると良い。) 実際にこの Φ について

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

が成り立つことは直接簡単に証明できる。

これから

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

$|\Psi_h(z)|$ の等高線を見よう。

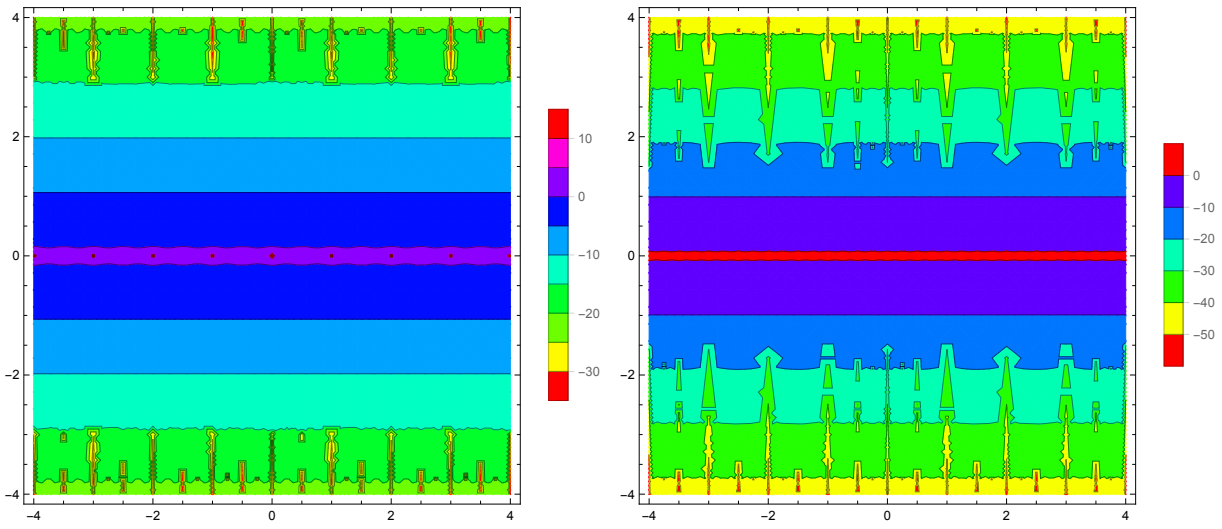

図 15: $\mathbb{R}$ 上の積分に対する台形公式 I_h $(h = 1/2, 1/4)$ の誤差の特性関数の絶対値

ぞっとするほど小さいことが見て取れる。

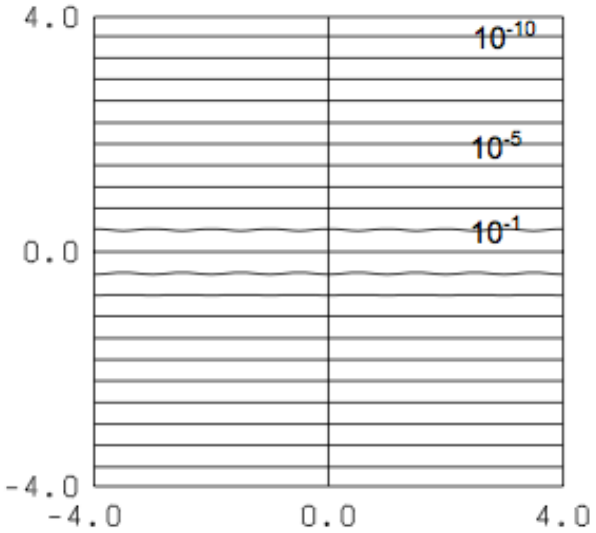

図 16: 森 [21], $\mathbb{R}$ 上の積分に対する台形公式 I_h $(h = 0.1)$ の誤差の特性関数の絶対値… 多分誤植で、実際は $h = 1$ の場合の図だと思われる。

A 関数近似

A.1 補間公式 (補間多項式)

関数 $f\colon [a, b] \to \mathbb{R}$ のグラフから、相異なる n 個の点 $(x_1, f(x_1)), \cdots, (x_n, f(x_n))$ を取ったとき、$n-1$ 次以下の多項式 $f_n(x)$ で、グラフがそれら n 個の点を通るものが一意的に存在することを示す。$f_n(x)$ を f の補間多項式と呼ぶ。

(導関数も近似するという Hermite 補間というものがあり、それと区別するために、Lagrange 補間ということがある。)

命題 A.1 (補間多項式の一意存在) $x_1, \cdots, x_n$ は区間 $[a,b]$ 内の相異なる点、$f\colon [a,b] \to \mathbb{R}$ とするとき、

$$f_n(x) \in \mathbb{R}[x], \quad \deg f_n(x) \leq n-1, \quad f_n(x_k) = f(x_k) \quad (k = 1, \ldots, n)$$

を満たす $f_n(x)$ が一意的に存在する。

定義 A.2 (補間多項式) $x_1, \cdots, x_n$ は区間 $[a,b]$ 内の相異なる点、$f\colon [a,b] \to \mathbb{R}$ とするとき、命題 A.1 で一意的な存在が保証される $f_n(x)$ を、$x_1, \cdots, x_n$ を標本点とする、関数 f の**補間多項式**と呼ぶ。

証明　$f_n(x) \in \mathbb{R}[x]$, $\deg f_n(x) \leq n-1$ であることから、

$$f_n(x) = \sum_{j=0}^{n-1} b_j x^j, \quad b_0, b_1, \cdots, b_{n=1} \in \mathbb{R}$$

と置くことが出来る。

$$f_n(x) = \begin{pmatrix} 1 & x & x^2 & \cdots & x^{n-1} \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{n-1} \end{pmatrix}$$

であるから、条件 $f_n(x_k) = f(x_k)$ $(k = 1, \cdots, n)$ は

$$\begin{pmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_n) \end{pmatrix} = \begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ \vdots & & & \vdots & \\ 1 & x_n & x_n^2 & \cdots & x_n^{n-1} \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-1} \end{pmatrix}$$

と同値であり、$\varphi\colon \mathbb{R}^n \to \mathbb{R}^n$ を $\varphi\left(\begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-1} \end{pmatrix}\right) := \begin{pmatrix} f_n(x_1) \\ f_n(x_2) \\ \vdots \\ f_n(x_n) \end{pmatrix}$ で定めると、φ は線形写像であり、(定義域と終域の次元が等しいので) 3 条件 (i) φ が全単射, (ii) φ が単射, (iii) φ が全射, が同値であることに注意する。

以下、3 通りの証明 (a), (b), (c) を与える。

(a) (構成的な証明) $k \in \{1, \cdots, n\}$ に対して、

$$L_k^{(n-1)}(x) := \frac{\displaystyle\prod_{\substack{1 \leq j \leq n \\ j \neq k}} (x - x_j)}{\displaystyle\prod_{\substack{1 \leq j \leq n \\ j \neq k}} (x_k - x_j)} = \frac{(x-x_1)\cdots(x-x_{k-1})(x-x_{k+1})\cdots(x-x_n)}{(x_k-x_1)\cdots(x_k-x_{k-1})(x_k-x_{k+1})\cdots(x_k-x_n)} \tag{30}$$

とおくと、

$$L_k^{(n-1)}(x) \in \mathbb{R}[x], \quad \deg L_k^{(n-1)}(x) = n-1, \quad L_k^{(n-1)}(x_j) = \delta_{jk} \quad (1 \leq j \leq n) \tag{31}$$

が成り立つ。ゆえに任意の $\begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} \in \mathbb{R}^n$ に対して、

$$f_n(x) := \sum_{k=1}^{n} a_k L_k^{(n-1)}(x) \tag{32}$$

とおくと、

$$f_n(x) \in \mathbb{R}[x], \quad \deg f_n(x) \leq n-1, \quad f_n(x_j) = a_j \quad (1 \leq j \leq n)$$

が成り立つ (実際、$f_n(x_j) = \sum_{k=1}^{n} a_k L_k^{(n-1)}(x_j) = \sum_{k=1}^{n} a_k \delta_{jk} = a_j$)。これは φ が全射であることを示している。ゆえに φ は全単射である。

(b) (省エネの証明. (志村 [23] に載っていた。) $(b_0, b_1, \cdots, b_n)^T \in \ker \varphi$ とする。$\varphi(b_0, b_1, \cdots, b_n) = \mathbf{0}$ は

$$f_n(x_1) = f_n(x_2) = \cdots = f_n(x_n) = 0$$

を意味する。$x_1, \cdots, x_n$ が相異なるならば、因数定理を用いて、$f_n(x) = 0$ (多項式として) が導かれる。ゆえに $b_0 = \cdots = b_{n-1} = 0$. ゆえに $\ker \varphi = \{\mathbf{0}\}$ であるので、φ は単射である。ゆえに φ は全単射である。

(c) (もしも Vandermonde の行列式を知っているならば) φ の表現行列は、Vandermonde 行列であり、その行列式は差積 $\prod_{1 \leq i < j \leq n} (x_j - x_i)$ に等しい。$x_1, \cdots, x_n$ が相異なるならば、これは 0 ではない。ゆえに φ は全単射である。 ■

定義 A.3 (Lagrange 補間係数) $x_1, \cdots, x_n$ を $\mathbb{R}$ 内の相異なる点とするとき、(30) で定まる $L_1^{(n-1)}(x), \ldots, L_n^{(n-1)}(x)$ を、$x_1 \cdots, x_n$ を標本点とする **Lagrange 補間係数**と呼ぶ。

細かい注意になるが、Lagrange 補間係数は、条件 (31) で特徴づけられる。実際、命題 A.1 の証明中の φ が単射であることから、(31) を満たす $L_k^{(n-1)}(x)$ は一意的である。

命題 A.4 $x_1, \cdots, x_n$ を $\mathbb{R}$ 内の相異なる点、$L_k^{(n-1)}(x)$ $(k = 1, \cdots, n)$ を、$x_1, \cdots, x_n$ を標本点とする Langrange 補間係数とするとき、$F_n(x) := \prod_{j=1}^{n} (x - x_j)$ とおくと、

$$L_k^{(n-1)}(x) = \frac{F_n(x)}{(x - x_k) F_n'(x_k)} \quad (k = 1, 2, \cdots, n).$$

証明

$$\frac{F_n(x)}{x - x_k} = \prod_{\substack{1 \leq j \leq n \\ j \neq k}} (x - x_j).$$

一方、積の微分法から

$$F_n'(x) = \sum_{k=1}^{n} \prod_{\substack{1 \leq j \leq n \\ j \neq k}} (x - x_j), \quad F_n'(x_k) = \prod_{\substack{1 \leq j \leq n \\ j \neq k}} (x_k - x_j) \quad (k = 1, \cdots, n)$$

であるから

$$\frac{F_n(x)}{(x-x_k)F_n'(x_k)} = \frac{\displaystyle\prod_{\substack{1\le j\le n\\ j\neq k}} (x-x_j)}{\displaystyle\prod_{\substack{1\le j\le n\\ j\neq k}} (x_k-x_j)} = L_k^{(n-1)}(x).\blacksquare$$

この命題を用いると、補間多項式 $f_n(x)$ は次のように表現できる:

$$f_n(x) = \sum_{k=1}^{n} \frac{F_n(x)}{(x-x_k)F_n'(x_k)} f(x_k), \quad F_n(x) := \prod_{j=1}^{n} (x-x_j)\,. \tag{33}$$

これを **Lagrange の補間公式**と呼ぶ。

補間多項式を表す公式には、これ以外に **Newton の補間公式**と呼ばれるものがあるが、省略する。

次の命題は具体的な誤差評価の役に立つのかなあ？

命題 A.5 (Lagrange 補間公式の誤差) (準備中) $n \in \mathbb{N}$, $f \in C^n([a,b];\mathbb{R})$ ならば、任意の $x \in [a,b]$ に対して、ある $\xi_x \in J$ が存在して、

$$f(x) - f_n(x) = \frac{1}{n!} F_n(x) f^{(n)}(\xi_x).$$

ここで J は $x_1, \cdots, x_n, x$ のすべてを含む最小の区間 (いわゆる区間包) である。

証明　(準備中 — 実は閉店だったりして) ■

A.2　Runge の現象

補間多項式において、標本点を等間隔に取ることにすると、標本点の数を増やすにつれて、補間多項式 $f_n(x)$ がもとの関数 f とかけ離れてしまうことが起こり得る。詳しい分析は後回しにして、ここでは実例を見てもらうことにする。

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

として、関数

$$f(x) = \frac{1}{1+25x^2} \tag{34}$$

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)
```

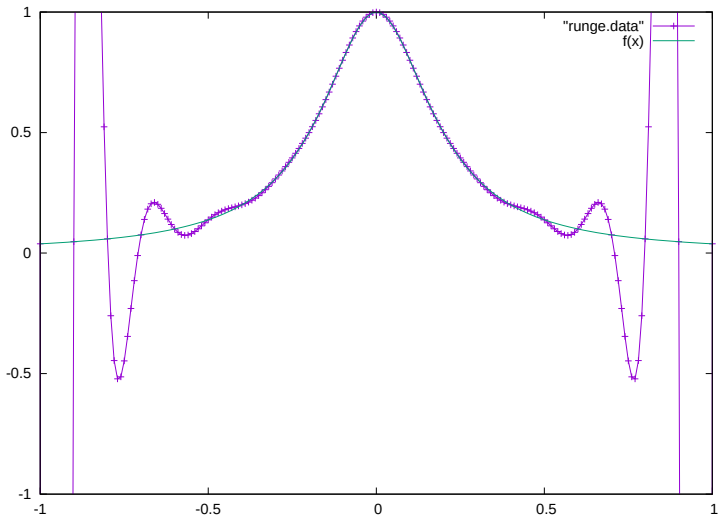

図 17: Runge の現象, $f(x) = \dfrac{1}{1+25x^2}$, $N = 10$

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。

この問題を回避するためにいくつか方法がある。

- 区間全体で 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、その各小区間で、小さい n に対して補間多項式 f_n を用いる。
  - スプライン近似
  - 有限要素法の区分的多項式
  - 数値積分では、積分区間を事前に小区間に分割しておき、各小区間で少ない個数の等間隔標本点による Lagrange 補間公式の積分を用いる (**複合則**と呼ぶ)。例えば、小区間 $[a, b]$ で $N = 2$, $x_1 = a$, $x_2 = \frac{a+b}{2}$, $x_2 = b$ としたものを **(複合) Simpson 公式**と呼ぶ。
- 直交多項式の根を標本点とする Lagrange 補間公式 (直交多項式補間) が利用する。直交多項式の根は (等間隔に分布するのではなく) 区間の端付近に集まる性質を持っていて、Runge の現象は起きない。

  直交多項式の根を標本点とする補間多項式に基づく補間型数値積分公式を、**Gauss 型の数値積分公式**と呼ぶ。n 次の Gauss 型数値積分公式は $2n-1$ 次以下の多項式に対して、正確な積分値を与える (この点で Newton-Cotes 公式と比べて非常に優れている)。

余談 A.6 Simpson 公式は、Thomas Simpson (1710–1761, 英国の Market Bosworth, Leicestershire に生まれ、没する) にちなむ。公式自体は有名な Newton が先に発見していたとか。歴史上最も良く使われた数値積分公式であるそうだ。

筆者が高校生の頃の教科書には、数値積分が載っていた。参考書には、関数の 3 次関数補間に基づく "Simpson $\frac{3}{8}$ 公式" というのも紹介されていた。(実は、積分に関しては、3 次関数補間が 2 次関数補間より優れているということはないので、Simpson $\frac{3}{8}$ 公式を使うのはくたびれ損なのだが…) ■

A.3 直交多項式補間

これは関数近似や数値積分では、重要性が高い事項であるが、今回は言及する余裕がないだろう (直交性がからむと色々面白いことが起こるのだが…)。

w は区間 $[a,b]$ で連続で、有限個の点を除き $w(x)>0$ を満たすとする。$C([a,b])$ における内積 $(\cdot,\cdot)_w$ を

$$(f,g)_w := \int_a^b f(x)g(x)w(x)\,dx \tag{35}$$

で定める。

次の条件を満たす多項式の列 $\{p_n(x)\}_{n\geq 0}$ を**直交多項式**と呼ぶ。

(a) $(\forall n\geq 0)\ p_n(x)\in\mathbb{R}[x],\ \deg p_n(x)=n.$

(b) $(\forall j,k\geq 0)\ j\neq k \Rightarrow (p_j,p_k)_w=0.$

(c) $p_0(x)=\mu_0>0.$

$1,\ x,\ x^2,\ \cdots$ に、内積 $(\cdot,\cdot)_w$ に関する Gram-Schmidt の直交化を施して得られる直交系 $p_0(x),\ p_1(x),\ \cdots,\ p_n(x),\ \cdots$ は直交多項式である。

注意 A.7 (直交多項式の定義について) 密度関数を定めたとき、直交多項式は定数倍を除いて定まるので、Gram-Schmidt の直交化をしたものを直交多項式と定義しても良さそうである。もしそうすると、最高次の係数は常に 1 のはずであるが、以下の (伝統的な直交多項式の) 例ではそうなっていない (負になったりしている)。そのため、上のように定義することにした。■

$$\mu_j := p_j(x) \text{ の最高次の係数},\quad \lambda_j := (p_j,p_j)_w \tag{36}$$

とおく。

$$f_n(x) := \sum_{j=0}^{n} c_j p_j(x),\quad c_j := \frac{(f,p_j)_w}{(p_j,p_j)_w}\quad (j=0,1,\cdots,n) \tag{37}$$

とおく。

命題 A.8 (直交多項式の根は積分区間内部に属する) w は区間 $[a,b]$ で連続で、有限個の点を除き $w(x)>0$ を満たすとする。このとき、任意の自然数 n に対して、$p_n(x)$ の根はすべて単純で、開区間 (a,b) に含まれる。

記号	名前	区間	密度関数 w	λ_n	
$P_n(x)$	Legendre 多項式	$[-1,1]$	1	$\dfrac{2}{2n+1}$	
$T_n(x)$	第 1 種 Chebyshev 多項式	$(-1,1)$	$\dfrac{1}{\sqrt{1-x^2}}$	$\pi/2\ (n\in\mathbb{N}),\ \pi\ (n=0)$	
$L_n(x)$	Laguerre 多項式	$[0,\infty)$	e^{-x}	1	
$H_n(x)$	Hermite 多項式	$(-\infty,\infty)$	e^{-x^2}	$\sqrt{\pi}2^n n!$	

表 1: 良く利用される直交多項式 (A.4 参照)

証明　(準備中)

命題 A.9 (直交多項式の 3 項漸化式)

$$p_k(x) = (\alpha_k x + \beta_k)\, p_{k-1}(x) - \gamma_k p_{k-2}(x) \quad (k \in \mathbb{N}). \tag{38}$$

ただし、

$$\begin{aligned} p_{-1}(x) &= 0, \\ \alpha_k &= \frac{\mu_k}{\mu_{k-1}}, \\ \beta_k &= -\frac{\alpha_k\,(x p_{k-1}, p_{k-1})}{\lambda_{k-1}}, \\ \gamma_k &= \frac{\alpha_k\,(x p_{k-1}, p_{k-2})}{\lambda_{k-2}} = \frac{\mu_k \mu_{k-2} \lambda_{k-1}}{\mu_{k-1}^2 \lambda_{k-2}}. \end{aligned}$$

証明　(準備中) ■

命題 A.10 (直交多項式は Strum 列) $p_0(x),\ p_1(x),\ \cdots,\ p_n(x),\ \cdots$ を直交多項式とするとき、任意の $n \in \mathbb{N}$ に対して、

$$p_n(x), p_{n-1}(x), \cdots, p_0(x)$$

は $[a,b]$ における Strum 列である。

証明　(準備中 — 当面使わないので後回し？) ■

命題 A.11 (Christoffel-Darboux の恒等式)

$$\sum_{k=0}^{n-1} \frac{p_k(x)p_k(y)}{\lambda_k} = \frac{\mu_{n-1}}{\mu_n \lambda_{n-1}} \cdot \frac{p_n(x)p_{n-1}(y) - p_{n-1}(x)p_n(y)}{x-y}. \tag{39}$$

証明　(準備中) 命題 A.9 を用いる。 ■

直交多項式の零点を標本点に用いた Lagrange 補間係数は簡単に求まる。

命題 A.12 w は区間 $[a,b]$ で連続で、有限個の点を除き $w(x)>0$ を満たすとする。$p_0(x)$, $\cdots$, $p_n(x)$ を内積 $(f,g)_w := \int_a^b f(x)g(x)w(x)\,dx$ に関する直交多項式で、x_1, $\cdots$, x_n を $p_n(x)$ の零点とする。このとき、

$$P_j^{(n-1)}(x) := w_j \sum_{k=0}^{n-1} \frac{p_k(x_j)p_k(x)}{\lambda_k}, \quad w_j := \left(\sum_{k=0}^{n-1} \frac{p_k(x_j)^2}{\lambda_k} \right)^{-1}$$

とおくと、$P_j^{(n-1)}$ は x_1, $\cdots$, x_n を標本点とする Lagrange 補間係数である。すなわち

$$P_j^{(n-1)}(x) = L_j^{(n-1)}(x).$$

証明　$j \neq \ell$ のとき (39) に $x=x_j$, $y=x_\ell$ を代入すると ($p_n(x_j)=p_n(x_k)=0$ であるから)

$$P_j^{(n-1)}(x_\ell) = w_j \sum_{k=0}^{n-1} \frac{p_k(x_j)\,p_k(x_\ell)}{\lambda_k} = w_j \frac{\mu_{n-1}}{\mu_n \lambda_{n-1}} \cdot \frac{p_n(x_j)p_{n-1}(x_\ell) - p_{n-1}(x_j)p_n(x_\ell)}{x_j - x_\ell} = 0.$$

一方、$j=\ell$ のとき、w_j の定義によって、

$$P_j^{(n-1)}(x_\ell) = w_j \sum_{k=0}^{n-1} \frac{p_k(x_j)\,p_k(x_\ell)}{\lambda_k} = w_j \sum_{k=0}^{n-1} \frac{p_k(x_j)^2}{\lambda_k} = 1.$$

ゆえに $P_j^{(n-1)}(x)$ は (31) を満たす。ゆえに $L_j^{(n-1)}(x)$ と一致する。 ■

$$f_n(x) = \sum_{j=1}^{n} f(x_j) P_j^{(n-1)}(x) = \sum_{j=1}^{n} f(x_j) \left(w_j \sum_{k=0}^{n-1} \frac{p_k(x_j)p_k(x)}{\lambda_k} \right) = \sum_{k=0}^{n-1} c_k p_k(x),$$

$$c_k := \frac{1}{\lambda_k} \sum_{j=1}^{n} w_j p_k(x_j) f(x_j).$$

これが効率よく計算できることを以下に示す。
(工事中…)

A.4　良く使われる直交多項式

定義式と重要な公式 (証明抜き) を列挙する。

例 A.13 (Legendre (ルジャンドル) 多項式, Legendre の球関数) Legendre 多項式 $P_n(x)$ とは、$a=-1$, $b=1$, $w(x)\equiv 1$ の場合の直交多項式である。しばしば、次の Rodrigues の公式で定義される。

$$P_n(x) = \frac{1}{2^n n!} \left(\frac{d}{dx} \right)^n \left[(x^2-1)^n \right]. \tag{40}$$

最初の 5 個を具体的に書くと

$$P_0(x)=1, \quad P_1(x)=x, \quad P_2(x)=\frac{1}{2}\left(3x^2-1\right), \quad P_3(x)=\frac{1}{2}\left(5x^3-3x\right),$$
$$P_4(x)=\frac{1}{8}\left(35x^4-30x^2+3\right), \quad \cdots$$

漸化式

$$(n+1)P_{n+1}(x) = (2n+1)\,xP_n(x) - nP_{n-1}(x)$$

を用いて計算するのが簡単なことが多い。

$P_n(x)$ は、Legendre の微分方程式の解である。

$$\lambda_n = \frac{2}{2n+1}.$$

Mathematica では、`LegendreP[n,x]` で $P_n(x)$ が計算できる。 ■

例 A.14 ((第 1 種)Chebyshev 多項式) (第 1 種)Chebyshev 多項式 $T_n(x)$ とは、$a=-1$, $b=1$, $w(x)=\dfrac{1}{\sqrt{1-x^2}}$ の場合の直交多項式である。

$$T_n(x) = \cos(nt), \quad x = \cos t. \tag{41}$$

$$\cos(2t) = 2\cos^2 t - 1, \quad \cos(3t) = 4\cos^3 t - 3\cos t, \quad \cdots$$

であるから、最初の 4 個を具体的に書くと

$$T_0(x) = 1, \quad T_1(x) = x, \quad T_2(x) = 2x^2 - 1, \quad T_3(x) = 4x^3 - 3x,$$

漸化式

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x) \quad (n \in \mathbb{N})$$

が成り立つ。

母関数を用いた特徴付け

$$\sum_{n=0}^{\infty} T_n(x)t^n = \frac{1-tx}{1-2tx+t^2}$$

も出来る。

Chebyshev の微分方程式

$$\left(1-x^2\right)y'' - xy' + n^2 y = 0$$

の解である。

$$\lambda_0 = \pi, \quad \lambda_n = \frac{\pi}{2} \quad (n \in \mathbb{N}).$$

Mathematica では、`ChebyshevT[n,x]` で $T_n(x)$ が計算できる。 ■

例 A.15 (Laguerre 多項式) Laguerre 多項式 $L_n(x)$ とは、$a=0$, $b=\infty$, $w(x)=e^{-x}$ の場合の直交多項式である。しばしば、次の Rodrigues の公式で定義される。

$$L_n(x) := e^x \left(\frac{d}{dx}\right)^n \left[x^n e^{-x}\right]. \tag{42}$$

最初の 4 個を具体的に書くと

$$L_0(x) = 1, \quad L_1(x) = 1-x, \quad L_2(x) = \frac{1}{2}\left(x^2 - 4x + 2\right), \quad L_3(x) = \frac{1}{6}\left(-x^3 + 9x^2 - 18x + 6\right), \quad \cdots$$

(最高次の係数が負になることがあるのに注意。)

母関数を用いた特徴付け

$$\sum_{n=0}^{\infty} \frac{L_n(x)}{n!}t^n = \frac{\exp\left(-\frac{tx}{1-t}\right)}{1-t}$$

も出来る。

微分方程式

$$xy'' + (1-x)y' + ny = 0$$

の解である。

Mathematica では、`LaguerreL[n,x]` で $L_n(x)$ が計算できる。 ■

例 A.16 (Hermite 多項式) Hermite 多項式 $H_n(x)$ とは、$a = -\infty$, $b = \infty$, $w(x) = e^{-x^2}$ の場合の直交多項式である。しばしば、次の Rodrigues の公式で定義される。

$$H_n(x) = (-1)^n e^{x^2} \left(\frac{d}{dx}\right)^n e^{-x^2}.$$

漸化式

$$H_{n+1}(x) = 2xH_n(x) - 2nH_{n-1}(x)$$

が成り立つ。最初の 5 個を具体的に書くと

$$H_0(x) = 1, \quad H_1(x) = 2x, \quad H_2(x) = 4x^2 - 2, \quad H_3(x) = 8x^3 - 12x,$$
$$H_4(x) = 16x^4 - 48x^2 + 12, \quad \cdots$$

母関数を用いた特徴付け

$$\sum_{n=0}^{\infty} \frac{H_n(x)}{n!} t^n = e^{-t^2+2tx}$$

も出来る。

$$\lambda_n = \sqrt{\pi} 2^n n!.$$

Mathematica では、`HermiteH[n,x]` で $H_n(x)$ が計算できる。 ■

B Newton の補間公式

この節の内容は、山本 [3] に基づく。

定義 B.1 $n = 0, 1, \cdots$ に対して、$n+1$ 変数関数 $f[x_n, \cdots, x_1, x_0]$ を次のように定め、f の n **階差分商**と呼ぶ。

$$f[x_0] := f(x_0),$$
$$f[x_1, x_0] := \frac{f[x_1] - f[x_0]}{x_1 - x_0},$$
$$f[x_2, x_1, x_0] := \frac{f[x_2, x_1] - f[x_1, x_0]}{x_2 - x_0},$$
$$\cdots$$
$$f[x_n, \cdots, x_1, x_0] := \frac{f[x_n, \cdots, x_1] - f[x_{n-1}, \cdots, x_0]}{x_n - x_0},$$
$$\cdots$$

命題 B.2

$$f\left[x_n, \cdots, x_1, x_0\right] = \sum_{i=0}^{n} \frac{f(x_i)}{(x_i - x_n) \cdots (x_i - x_{i+1}) (x_i - x_{i-1}) \cdots (x_i - x_0)}.$$

証明

$$\begin{aligned}
f\left[x_1, x_0\right] &= \frac{f(x_1) - f(x_0)}{x_1 - x_0}, \\
f\left[x_2, x_1, x_0\right] &= \frac{f\left[x_2, x_1\right] - f\left[x_1, x_0\right]}{x_2 - x_0} = \frac{1}{x_2 - x_0} \left(\frac{f(x_2) - f(x_1)}{x_2 - x_1} - \frac{f(x_1) - f(x_0)}{x_1 - x_0} \right) \\
&= \frac{f(x_2)}{(x_2 - x_1)(x_2 - x_0)} + \frac{f(x_1)}{x_2 - x_0} \cdot \left(\frac{1}{x_1 - x_2} - \frac{1}{x_1 - x_0} \right) + \frac{f(x_0)}{(x_0 - x_2)(x_0 - x_1)} \\
&= \frac{f(x_0)}{(x_0 - x_2)(x_0 - x_1)} + \frac{f(x_1)}{(x_1 - x_2)(x_1 - x_0)} + \frac{f(x_2)}{(x_2 - x_1)(x_2 - x_0)}.
\end{aligned}$$

$n = 1, 2$ のとき、確かに成り立つ。

$n = k$ まで成り立ったとする。

$$\begin{aligned}
&f\left[x_{k+1}, x_k, \cdots, x_1, x_0\right] \\
&= \frac{f\left[x_{k+1}, \cdots, x_0\right] - f\left[x_k, \cdots, x_1\right]}{x_{k+1} - x_0} \\
&= \frac{1}{x_{k+1} - x_0} \left(\sum_{i=1}^{k+1} \frac{f(x_i)}{(x_i - x_{k+1}) \cdots (x_i - x_{i+1}) (x_i - x_{i-1}) \cdots (x_i - x_1)} \right. \\
&\qquad \left. - \sum_{i=0}^{k} \frac{f(x_i)}{(x_i - x_k) \cdots (x_i - x_{i+1}) (x_i - x_{i-1}) \cdots (x_i - x_0)} \right) \\
&= \frac{1}{x_{k+1} - x_0} \left[\frac{f(x_{k+1})}{(x_{k+1} - x_k) \cdots (x_{k+1} - x_1)} \right. \\
&\qquad + \sum_{i=1}^{k} f(x_i) \left(\frac{1}{(x_i - x_{k+1}) \cdots (x_i - x_{i+1}) (x_i - x_{i-1}) \cdots (x_i - x_1)} \right. \\
&\qquad\qquad \left. \left. - \frac{1}{(x_i - x_k) \cdots (x_i - x_{i+1}) (x_i - x_{i-1}) \cdots (x_i - x_0)} \right) \right. \\
&\qquad \left. - \frac{f(x_0)}{(x_0 - x_k) \cdots (x_0 - x_1)} \right] \\
&= \frac{f(x_{k+1})}{(x_{k+1} - x_k) \cdots (x_{k+1} - x_0)} + \frac{f(x_0)}{(x_0 - x_{k+1}) \cdots (x_0 - x_1)} \\
&\quad + \frac{1}{x_{k+1} - x_0} \sum_{i=1}^{k} f(x_i) \frac{(x_i - x_0) - (x_i - x_{k+1})}{(x_i - x_{k+1}) \cdots (x_i - x_{i+1}) (x_i - x_{i-1}) \cdots (x_i - x_0)} \\
&= \frac{f(x_{k+1})}{(x_{k+1} - x_k) \cdots (x_{k+1} - x_0)} + \frac{f(x_0)}{(x_0 - x_{k+1}) \cdots (x_0 - x_1)} \\
&\quad + \sum_{i=1}^{k} \frac{f(x_i)}{(x_i - x_{k+1}) \cdots (x_i - x_{i+1}) (x_i - x_{i-1}) \cdots (x_i - x_0)} \\
&= \sum_{i=0}^{k+1} \frac{f(x_i)}{(x_i - x_{k+1}) \cdots (x_i - x_{i+1}) (x_i - x_{i-1}) \cdots (x_i - x_0)}.
\end{aligned}$$

これは $n = k + 1$ のときも成り立つことを示す。 ■

命題 B.3 (差分商の対称性) n 階差分商 $f[x_n, \cdots, x_1, x_0]$ は $x_0, \cdots, x_n$ の対称式である。すなわち σ を $\{0, 1, \cdots, n\}$ の任意の置換とするとき

$$f\left[x_{\sigma(n)}, \cdots, x_{\sigma(0)}\right] = f[x_n, \cdots, x_0].$$

証明　命題 B.2 からすぐ分かる。 ■

次の定理は、Taylor の定理に対応すると考えられる。

命題 B.4 (Newton の補間公式) $x_0, x_1, \cdots, x_n$ を $\mathbb{R}$ 内の相異なる点とする。$x_0, x_1, \ldots, x_k$ を標本点とする k 次補間多項式を $p_k(x)$ とする $(k = 1, 2, \cdots, n)$ とき、

$$\begin{aligned} p_n(x) &= f[x_0] + f[x_0, x_1](x - x_0) + \cdots \\ &\qquad + f[x_0, x_1, \cdots, x_{n-1}, x_n](x - x_0)(x - x_1)\cdots(x - x_{n-1}) \\ &= p_{n-1}(x) + f[x_0, x_1, \cdots, x_{n-1}, x_n] \prod_{i=0}^{n-1} (x - x_i). \end{aligned}$$

C　補間多項式を計算するアルゴリズム

(工事中)

C.1　はじめに

この節の内容は杉原・室田 [2] の第 9 章に基づく (山本 [3] を読んで良く分からなかったので)。

Vandermonde の行列の条件数は大きくなりがちで、連立 1 次方程式を解いて補間多項式を求めるのは避けるべきらしい (このことを自分で検討したことはまだない)。

以下、標本点の集合の部分集合を標本点とする補間多項式を導入し、漸化式を導く。

C.2　標本点全体の部分集合を標本点集合とする補間多項式

$[a, b]$ は $\mathbb{R}$ の区間、$x_1, \ldots, x_n$ は $[a, b]$ 内の相異なる点、$f\colon [a, b] \to \mathbb{R}$ とする。

これまで「$x_1, \ldots, x_n$ を標本点とする f の補間多項式」を考えたが、補間多項式は標本点の並べ方には依らずに定まるので、それを「$\{x_1, \ldots, x_n\}$ を標本点集合とする f の補間多項式」という呼び方をしても混乱は生じない。

(念のため表現を変えて繰り返し: これまでは (重複のない) 有限点列に関して補間多項式を定義したが、有限点集合に関して補間多項式を定義できる。)

$\{x_1, \cdots, x_n\}$ の任意の空でない部分集合 $S = \{x_{i_1}, \cdots, x_{i_m}\}$ (m は S の要素数) に対して、S を標本点集合とする f の補間多項式 $P[S](x)$ を導入する。すなわち、$P[S](x)$ は次の条件で定まる。

$$P[S](x) \in \mathbb{R}[x], \quad \deg P[S](x) \leq m - 1, \quad P[S]\left(x_{i_j}\right) = f\left(x_{i_j}\right) \quad (j = 1, \cdots, m).$$

$P[S](x)$ を $P[x_{i_1}, \ldots, x_{i_m}](x)$ あるいは $P[i_1, \ldots, i_m](x)$ と略記し、さらに $i_1, \cdots, i_m$ が連続している、すなわち

$$i_2 = i_1 + 1, \quad i_3 = i_2 + 1, \quad \cdots, \quad i_m = i_{m-1} + 1$$

が成り立っているときは $P[i_1 \sim i_m](x)$ とも表す。

$P[S](x)$ の $m-1$ 次の係数を $C[S]$ と表す。(これは $P[S](x)$ の最高次の係数ではないことに注意する。$P[S](x)$ は $m-1$ 次以下の多項式であるが、$m-1$ 次多項式であるとは限らない。)

補題 C.1 (漸化式) $[a,b]$ は $\mathbb{R}$ の区間、$x_1, \ldots, x_n$ は $[a,b]$ 内の相異なる点、$f\colon [a,b] \to \mathbb{R}$, S は $\{x_1, \cdots, x_n\}$ の任意の部分集合 ($S = \emptyset$ も許す) とする。$1 \le j \le n$, $1 \le k \le n$, $j \ne k$, $S \cap \{x_j, x_k\} = \emptyset$ ならば、

$$P[S; x_j, x_k](x) = \frac{(x - x_j)P[S; x_k](x) - (x - x_k)P[S; x_j](x)}{x_k - x_j}, \tag{43}$$

$$C[S; x_j, x_k] = \frac{C[S; x_k] - C[S; x_j]}{x_k - x_j}. \tag{44}$$

ただし、$P[S \cup \{x_j\}](x)$, $P[S \cup \{x_k\}](x)$, $P[S \cup \{x_j, x_k\}](x)$ をそれぞれ $P[S; x_j](x)$, $P[S; x_k](x)$, $P[S; x_j, x_k](x)$ と略記した。

証明

(43) の右辺を $p(x)$ とおく。S の要素数を m とおくとき、$S \cup \{x_k\}$, $S \cup \{x_j\}$ の要素数はともに $m+1$ であるから、$P[S; x_k](x)$, $P[S; x_j](x)$ はともに m 次以下の多項式である。ゆえに $p(x)$ は $m+1$ 次以下の多項式である。また

$$p(x) = \begin{cases} f(x_j) & (x = x_j) \\ f(x_k) & (x = x_k) \\ f(x_i) & (x = x_i \in S) \end{cases}$$

が成り立つ。実際、

(i) $x = x_j$ のとき、補間多項式の定義により $P[S; x_j](x) = f(x_j)$ であるから、

$$p(x) = \frac{(x_j - x_j)\, P[S; x_k](x_j) - (x_j - x_k)\, f(x_j)}{x_k - x_j} = \frac{0 - (x_j - x_k)\, f(x_j)}{x_k - x_j} = f(x_j).$$

(ii) $x = x_k$ のときも、(i) と同様にして $p(x) = f(x_k)$ が証明できる。

(iii) $x = x_i \in S$ のとき、補間多項式の定義により $P[S; x_k](x) = P[S; x_j](x) = f(x_i)$ であるから、

$$p(x) = \frac{(x - x_j)\, f(x_i) - (x - x_k)\, f(x_i)}{x_k - x_j} = \frac{(x_k - x_j)\, f(x_i)}{x_k - x_j} = f(x_i).$$

ゆえに $p(x)$ は、$S \cup \{x_j, x_k\}$ の要素を標本点とする f の補間多項式 $P[S; x_j, x_k](x)$ に一致する。

(43) の $m+1$ 次の係数を取ると、(44) が得られる。 ■

補間多項式の、この漸化式に基づくアルゴリズムとして、**Neville アルゴリズム**, **Aitken アルゴリズム**と呼ばれるものが知られている。前者について次項で解説する。

C.3 Neville アルゴリズム

f の $x_1, \ldots, x_n$ に関する補間多項式は、前項 (C.2) の記号で $P[1 \sim n](x)$ である。これを計算するアルゴリズムである Neville アルゴリズムを解説する。

C.3.1　1 点における値の計算

杉原・室田 [2] には、漸化式

$$P[k\sim k+j](x)=\frac{(x-x_{k+j})\,P[k\sim k+j-1](x)-(x-x_k)\,P[k+1\sim k+j](x)}{x_k-x_{k+j}} \tag{45}$$

と、図 18 が載っているが、計算の開始部分が若干分かりにくいと思われるので、説明を補足しておく。

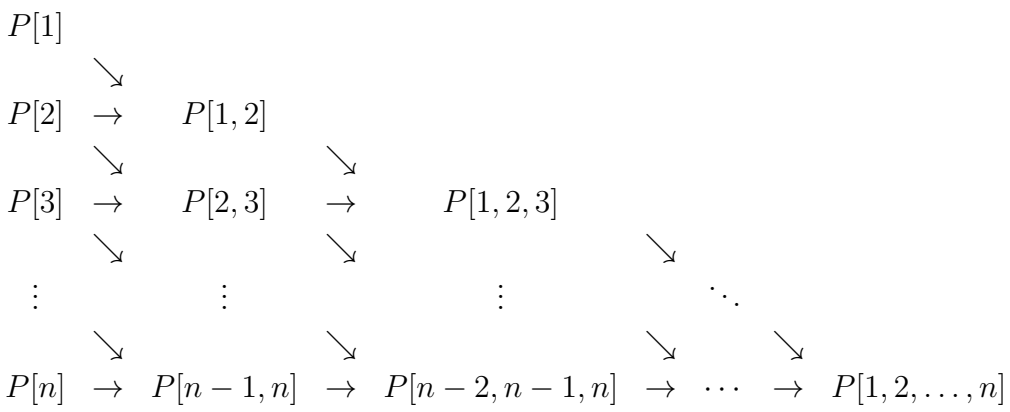

図 18: Neville 算法

第 1 段　まず 0 次多項式 (定数) $P[1](x)$, $P[2](x)$, $\cdots$, $P[n](x)$ を

$$P[k](x)=f(x_k)\quad(k=1,\dots,n)$$

で求める。

第 2 段　1 次以下の多項式 $P[1,2](x)$, $P[2,3](x)$, $\cdots$, $P[n-1,n](x)$ を、補題 C.1 を $S=\emptyset$ として用いて、

$$P[k,k+1](x):=\frac{(x-x_{k+1})P[k](x)-(x-x_k)P[k+1](x)}{x_{k+1}-x_k}$$

で求める。

第 3 段　$P[k,k+1,k+2](x)$ $(k=1,2,\cdots,n-2)$ を、補題 C.1 を $S=\{x_{k+1}\}$ として用いて、

$$\begin{aligned}P[k,k+1,k+2](x)&=P[\{x_{k+1}\};x_k,x_{k+2}](x)\\&:=\frac{(x-x_{k+2})P[k,k+1](x)-(x-x_k)P[k+1,k+2](x)}{x_k-x_{k+2}}.\end{aligned}$$

で求める。

上に引用した (45) は、(ここでの数え方で) 第 $j+1$ 段の計算式である。
第 n 段で、$P[1\sim n-1](x)$ と $P[2\sim n](x)$ から $P[1\sim n](x)$ が求まる。

C.3.2　多数の点における値の計算

x の多数の値に対して、補間多項式 $P(x) = P[1 \sim n](x)$ の値を求めるには、事前に式をある程度整理しておくと都合が良い。

ここでは、Newton 補間公式

$$P(x) = P[1 \sim n](x) = \sum_{j=0}^{n-1} c_j R_j(x), \quad R_j(x) = \prod_{i=1}^{j} (x - x_i)$$

を用いる。あらかじめ c_j $(j = 0, \ldots, n-1)$ を求めておくと、x が与えられるごとに、Horner 法 (組立除法) と類似したアルゴリズム

```
b_0 := c_{n-1};
for (j = 1; j <= n - 1; j++)
  b_j = b_{j-1}(x - x_{n-j}) + c_{n-1-j};
```

で $P(x) = b_{n-1}$ が計算できる。Newton 補間公式は計算に便利であることが分かる。

$$P[1 \sim k](x) = \sum_{j=0}^{k-1} c_j R_j(x) \quad (k = 1, \ldots, n)$$

であるから、その $k-1$ 次の係数として

$$c_{k-1} = C[1 \sim k] \quad (k = 1, \ldots, n). \tag{46}$$

C.3.1 において、多項式 $P[k \sim k+j](x)$ $(0 \le j \le n-1,\ 1 \le k \le n-j)$ を求めた。$P[k \sim k+j](x)$ は j 次以下の多項式であるが、その j 次の係数 $C[k \sim k+j]$ を考える。まず $j = 0$ に対しては

$$C[k] = P[k](x) = f(x_k) \quad (k = 1, \cdots, n). \tag{47}$$

$j = 1$ に対して、

$$P[k, k+1](x) = \frac{(x - x_{k+1})\, P[k](x) - (x - x_k)\, P[k+1](x)]}{x_k - x_{k+1}}$$

であるから、

$$C[k, k+1] = \frac{C[k] - C[k+1]}{x_k - x_{k+1}} \quad (k = 1, \ldots, n-1) \tag{48}$$

である。

各 $j = 2, \ldots, n-1$ と、各 $k = 1, \ldots, n-j$ に対して、

$$P[k \sim k+j](x) = \frac{P[k \sim k+j-1](x) - P[k+1 \sim k+j](x)}{x_k - x_{k+j}}$$

であるから、

$$C[k \sim k+j] = \frac{C[k \sim k+j-1] - C[k+1 \sim k+j]}{x_k - x_{k+j}} \quad (j = 2, \ldots, n-1;\ k = 1, \ldots, n-j) \tag{49}$$

である。(この式で $j = 1$) としたものは、(48) に一致する。

これから $C[k \sim k+j]$ を (c[k][$k+j$] でなく) c[k][j] に記憶するとして、次のようなコードが出来る (まだ全然チェックしていないので信用しないように)。

2 次元配列を用いたコーディング

```
for (k = 1; k <= n; k++)
  c[k][0] = f(x[k]);
for (j = 1; j < n; j++)
  for (k = 1; k <= n-j; k++)
    c[k][j] = (c[k][j-1] - c[k+1][j-1]) / (x[k] - x[k+j]);
```

1 次元配列を用いたコーディング

```
for (k = 1; k <= n; k++)
  c[k] = f(x[k]);
for (j = 1; j < n; j++)
  for (k = 1; k <= n-j; k++)
    c[k+j] = (c[k+j-1] - c[k+j]) / (x[k] - x[k+j]);
```

実例が、山本 [3]、ハイラー・ヴァンナー [17]、篠原 [5] にあった気がする。探して電子化して、それをプログラムにしてチェックして、ここを完備化する。

C.4 プログラム例

次のプログラムは、多項式関数 $f(x) = x^3 - 2x^2 + 7x - 5$ の、0, 1, 2, 3 を標本点とする補間多項式を Newton 補間公式で求め、標本点での値を計算させてみたものである。

```
/*
 * dif.c --- 階差、差分商、Newton の補間公式
 */

#define MAXN (100)
#include <stdio.h>

double fun(double x)
{
  int i, n = 3;
  double a[4] = {1, -2, 7, -5};
  double s;
  // Horner 法による多項式の計算
  s = a[0];
  for (i = 1; i <= n; i++)
    s = s * x + a[i];
  return s;
}

// 階差の計算
void differences(int n, double x[], double f[])
{
```

```
  int i, k;
  for (k = 1; k <= n; k++)
    for (i = n; i >= k; i--) {
      f[i] = f[i] - f[i-1];
    }
}

// 差分商の計算
void difference_quotients(int n, double x[], double f[])
{
  int i, k;
  for (k = 1; k <= n; k++)
    for (i = n; i >= k; i--) {
      f[i] = (f[i] - f[i-1]) / (x[i] - x[i-k]);
    }
}

// Mewton の補間公式による補間多項式の計算
// 入力 n: 次数, x[]: 標本点, dq[]: 差分商
//      xx: 計算したい点
double newton(int n, double x[], double dq[], double xx)
{
  int i;
  double s;
  s = dq[n];
  for (i = n - 1; i >= 0; i--)
    s = dq[i] + (xx - x[i]) * s;
  return s;
}

int main(void)
{
  double f[MAXN+1], x[MAXN+1], fv[MAXN+1];
  int i, n;
  n = 3;
  // 標本点
  for (i = 0; i <= n; i++)
    x[i] = i;
  // 標本値
  for (i = 0; i <= n; i++)
    fv[i] = fun(x[i]);
  for (i = 0; i <= n; i++)
    printf("%g %g\n", x[i], fv[i]);

  for (i = 0; i <= n; i++)
    f[i] = fv[i];
```

```
  differences(n, x, f);
  for (i = 0; i <= n; i++)
    printf("%d 階階差 %g\n", i, f[i]);

  for (i = 0; i <= n; i++)
    f[i] = fv[i];
  difference_quotients(n, x, f);
  for (i = 0; i <= n; i++)
    printf("%d 階差分商 %g\n", i, f[i]);
  for (i = 0; i <= n; i++)
    printf("%f %f\n", x[i], newton(n, x, f, x[i])
);
  return 0;
}
```

実行例

```
$ cc dif.c
$ ./a.out
0 -5
1 1
2 9
3 25
0 階階差 -5
1 階階差 6
2 階階差 2
3 階階差 6
0 階差分商 -5
1 階差分商 6
2 階差分商 1
3 階差分商 1
0.000000 -5.000000
1.000000 1.000000
2.000000 9.000000
3.000000 25.000000
```

D 浮動小数点数メモ & DE 公式のプログラミングのヒント (2016/5/18)

D.1 浮動小数点数の精度、値の範囲

要点

Mac の C 言語処理系の double は、10 進法に換算して 16 桁弱の精度、その精度を保って計算できる値の範囲は絶対値で 2.2×10^{-308}〜1.79×10^{308}

現在、C 言語等でプログラミングする場合、実数は浮動小数点数で表現される。パソコン、ワークステーションでは、IEEE 754 規格に準拠した浮動小数点数が使える。

double 型のデータでは、10 進法に換算して、16 桁弱の精度を持つ (2 進法で 53 桁なので、いわゆる計算機イプシロンは $2^{-52} = 2.22045\cdots \times 10^{-16}$ である)。正の正規化数のうち、最小の数は $2^{-1022} \fallingdotseq 2.2250738585 \times 10^{-308}$, 最大の数は $2^{1024}(1-2^{-53}) \fallingdotseq 1.797693134862315 \times 10^{308}$ であり、正の非正規化数のうち最小の数は $2^{-1074} \fallingdotseq 4.940656 \times 10^{-324}$ である ([24])。

D.2 DE 公式を使う場合の注意

DE 公式では、指数関数や二重指数関数が出て来るので、絶対値が極度に大きい数、極度に小さい数が登場する。

$\exp(x)$ は $-708.396419 < x < 709.782712\cdots$ 位までは、オーバーフローもアンダーフローもしない。

いわゆる IBM 互換の大型電子計算機では、10^{75} 位までしか表現できない (詳しいことは知らない)。$-173 < x < 172$ 位まではオーバーフローもアンダーフローもしない。

$\varphi_1(t) = \tanh\left(\frac{\pi}{2}\sinh t\right)$ が、$\varphi_1'(t) = 10^{-75}$ となるのは、$t = 4.733$ 位。これはかなり小さい。$\varphi_1'(t) = 10^{-308}$ となるのは、$t = 6.12163$ 位。

x	$\varphi_1^{-1}(x)$ ($\varphi_1(t) = x$ となる t)
$2^{-53} = 2.22045\cdots \times 10^{-16}$	3.249732644072075477431541124 23
10^{-75}	4.733487533849132053405072338 10
2.225×10^{-308}	6.121631243191162965791067442 97

表 2: 注意すべき限界値

$|nh| \leq 3$ では十分な精度が得られない可能性がある。一方 $|nh| = 10$ のようなのは大きすぎて明らかな無駄。IBM 互換機ならば $|nh| \simeq 4.5$ 程度, IEEE 754 規格を用いたコンピューターでは $|nh| \simeq 6$ 程度が目安か。

$t = 3.154$ のとき、$1 - \varphi_1(t) = 2.22 \times 10^{-16}$ なので、それを超えると、$\varphi_1(t)$ の計算値は 1 になるおそれがある。$f(x) = \dfrac{1}{\sqrt{1-x^2}}$ のように、$x = \pm 1$ で定義されていない関数の積分を計算する場合は対策が必要である。

t が 3 を超えて $|f(\varphi_1(t))\varphi_1'(t)|$ が目標とする精度よりも小さくなっていれば良いが、そうでない場合は、何か対策を施さないと十分な精度が得られない可能性が高い。

以上は $t > 0$ の場合を考えたが、$t < 0$ でも同様である。$t = -3.145$ のとき、$\varphi_1(t) - (-1) = 2.22 \times 10^{-16}$ なので、t がそれより小さいと、$\varphi_1(t)$ の計算値が -1 になるおそれがある。t が -3 より小さくなったとき、$|f(\varphi_1(t))\varphi_1'(t)|$ が目標とする精度よりも小さくなっていれば良いが、そうでない場合は、対策が必要である。

一度、どういう数値の和を取っているのか、個々の数値を表示して検討してみることを勧める。

```
N = ceil(3.25 / h);
s = 0.0;
for (n = - N; n <= N; n++) {
  t = n * h;
  f1 = f(phi(t));
  f2 = dphi(t);
  term = f1 * f2;
  s += term;
  printf("%f %e %e %e %20g\n", t, f1, f2, term, s);
}
```

E $\log\frac{z-1}{z+1}$ $(z \in \mathbb{C} \setminus [-1,1])$ 等について

$\log\dfrac{z-a}{z-b}$ $(z \in \mathbb{C} \setminus [-1,1])$ や、$\log\dfrac{1+z}{1-z}$ $(z \in D(0;1))$ などが出て来る。

記号の復習: $a, b \in \mathbb{C}$, $a \neq b$ のとき、$[a,b] := \{(1-t)a + tb \mid t \in [0,1]\}$. $c \in \mathbb{R}$, $r > 0$ のとき、$D(c;r) := \{z \in \mathbb{C} \mid |z-c| < r\}$.

言いたいこと

$a, b \in \mathbb{C}$, $a \neq b$ とするときに $\log\dfrac{z-a}{z-b}$ $(z \in \mathbb{C} \setminus [a,b])$ や、$\log\dfrac{1-z}{1+z}$ $(z \in D(0;1))$ がよく出て来る。これらは正則関数としてきちんと定義できる。

早い話

log を対数関数の主値 Log とすれば良い。

補題 E.1 $z, a, b \in \mathbb{C}$ は相異なるとして、$X := \dfrac{z-a}{z-b}$ とおくとき、次が成り立つ。

(1) z から見て、a, b が同じ方向であれば、$X > 0$.

(2) z から見て、a, b が反対方向であれば、$X < 0$.

(3) (1), (2) のいずれでもない場合、$X \notin \mathbb{R}$.

特に $z \in \mathbb{C} \setminus [a,b]$ であれば ((2) でないので) $X \in \mathbb{C} \setminus (-\infty, 0]$.

証明 $a \neq z$, $b \neq z$ であるから、$a = z + r_1 e^{i\theta_1}$, $b = z + r_2 e^{i\theta_2}$ $(r_1, r_2 > 0,\ \theta_1, \theta_2 \in \mathbb{R})$ と表される。そして

$$X = \frac{z-a}{z-b} = \frac{r_1 e^{i\theta_1}}{r_2 e^{i\theta_2}} = \frac{r_1}{r_2} e^{i(\theta_1 - \theta_2)}.$$

(1) の場合、$\theta_1 - \theta_2 \equiv 0 \pmod{2\pi}$ であるから、$X > 0$. (2) の場合、$\theta_1 - \theta_2 \equiv \pi \pmod{2\pi}$ であるから、$X < 0$. それ以外の場合、$\theta_1 - \theta_2 \not\equiv 0 \pmod{\pi}$ であるから、$X \notin \mathbb{R}$. ■

系 E.2 $a, b \in \mathbb{C}$, $a \neq b$ とするとき

$$f(z) := \operatorname{Log} \frac{z-a}{z-b} \quad (z \in \mathbb{C} \setminus [a,b]).$$

は $\mathbb{C} \setminus [a,b]$ で定義された正則関数である。

証明　$z \in \mathbb{C} \setminus [a,b]$ のとき、$\dfrac{z-a}{z-b}$ は Log の定義域に含まれる。 ■

補題 E.3 $z \in D(0;1)$ のとき、$\dfrac{1+z}{1-z} = \dfrac{-z-1}{z-1}$ は負の値を取らない。

証明　もし負の値を取ったとすると、1 から見て、z と $-z$ は正反対の方向である。z が単位円板内の点であれば、$-z$ も単位円板内の点であり、1 から見て正反対の方向にあるはずはない。 ■

系 E.4

$$f(z) := \operatorname{Log} \frac{1+z}{1-z} \quad (z \in D(0;1)).$$

は $D(0;1)$ で定義された正則関数である。

証明　$z \in D(0;1)$ のとき、$\dfrac{1+z}{1-z}$ は Log の定義域に含まれる。 ■

以下、復習がてらの、ゆっくりした説明。
複素関数で次のことを学んだ。

- $\sqrt{z}$ は $\mathbb{C}$ 上の正則関数として定義出来ない (無理にそうすると多価関数になってしまう)
  $z = re^{i\theta}$ が 0 の周りを正の向きに一周すると、偏角が $\dfrac{2\pi}{2} = \pi$ 増える (-1 倍になる)。2 周すると元に戻る ($\sqrt{z} = \pm\sqrt{r}e^{i\theta/2}$)。
- $\log z$ は $\mathbb{C} \setminus \{0\}$ 上の正則関数として定義出来ない (無理にそうすると同上)
  $z = re^{i\theta}$ が 0 の周りを正の向きに一周すると、虚部が 2π 増える ($\log z = \log r + i(\theta + 2n\pi)$, $n \in \mathbb{Z}$)。螺旋階段状態。

1 価正則な値を選べるようにするため、定義域を制限するのだった。よくあるのは、実軸の負の部分を除いた $\Omega := \mathbb{C} \setminus (-\infty, 0]$ で考え、いわゆる主値

$$\operatorname{Log} z = \log r + i\theta \quad (z = re^{i\theta},\ r > 0,\ \theta \in (-\pi, \pi))$$

を使う。(より一般に、原点を始点とする半直線を除いて、同じようなことが出来る。) 対数 $\log z$ が決まれば、

$$\sqrt{z} = \exp\left(\frac{\log z}{2}\right)$$

で $\sqrt{z}$ を決めることが出来る。

$a \in \mathbb{C}$ とするとき、$\log(z-a)$, $\sqrt{z-a}$ についても同様で、$\mathbb{C}$ から a を始点とする半直線を除いた領域で、1 価正則な関数が定められる。

おまけ

$a, b \in \mathbb{C}$, $a \neq b$ とするとき、$\sqrt{(z-a)(z-b)}$ $(z \in \mathbb{C} \setminus [a,b])$ は正則関数としてきちんと定義できる。

F　複素関数論を用いたもう一つの誤差解析 — 杉原理論

ここでは、杉原による誤差解析を述べる (杉原・室田 [2], 杉原 [18])。

級数の和の評価 (「応用複素関数」の 4/20 の講義でも取り上げた, 桂田 [16] の §§2.3) と同様のことを行う部分がある。しかし、他の部分は重いので、実際には講義していない。(時間配分をうまくすれば、講義できて、複素関数論の講義として有益だったかもしれない。)

F.1　実軸上の積分に対する台形公式

実解析的な $f\colon \mathbb{R} \to \mathbb{C}$ に対して、

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

の近似値を求めるため、台形則を用いたときの誤差を調べたい。

$h > 0$, $N \in \mathbb{N}$ とするとき、

$$T_h := h \sum_{n=-\infty}^{\infty} f(nh), \quad T_{h,N} := h \sum_{n=-N}^{N} f(nh)$$

とおく。

f はある $d > 0$ に対して、

$$\mathcal{D}(d) := \{z \in \mathbb{C} \mid |\mathrm{Im}\, z| < d\}$$

上の正則な関数に拡張されると仮定する。

$\mathcal{D}(d)$ で正則で 0 にならない関数 w を一つ取り、

$$\|f\| := \sup_{z \in \mathcal{D}(d)} \left|\frac{f(z)}{w(z)}\right|,$$
$$H(w,d) := \{f \mid f\colon \mathcal{D}(d) \to \mathbb{C} \text{ 正則}, \|f\| < +\infty\}$$

とおく。

w は、$x \to \pm\infty$ のときの $f(x)$ の減衰の度合いを示す関数である。

補題 F.1 (無限和台形則の誤差評価) $f\colon \mathcal{D}(d) \to \mathbb{C}$ は正則で、次の 2 条件を満たすとする。

(a) $\forall c \in (0,d)$ に対して、

$$\Lambda(f,c) := \int_{-\infty}^{\infty} (|f(x+ic)| + |f(x-ic)|)\, dx < +\infty.$$

さらに

$$\Lambda(f, d-0) := \lim_{c \to d-0} \Lambda(f,c)$$

は有限確定である。

(b) $\forall c \in (0,d)$ に対して、

$$\lim_{x \to \pm\infty} \int_{-c}^{c} |f(x+iy)|\, dy = 0.$$

このとき、任意の $h > 0$ に対して、

$$|I - T_h| \le \frac{\exp(-2\pi d/h)}{1-\exp(-2\pi d/h)} \Lambda(f, d-0). \tag{50}$$

(50) の右辺は、$h \to +0$ とするとき、急速に減衰する (0 に近づく) ことに注意すること。その減衰の速さは、d が大きいほど大きい。

証明　$c \in (0,d)$, $N \in \mathbb{N}$ を固定して、4 点 $-(N+1/2)\,h - ci$, $(N+1/2)\,h - ci$, $(N+1/2)\,h + ci$, $-(N+1/2)\,h + ci$ を頂点とする長方形を正の向きに一周する曲線を $C_{c,N}$ とする。

$\varphi(z) := \cot \dfrac{\pi z}{h} = \dfrac{\cos(\pi z/h)}{\sin(\pi z/h)}$ において、

$$z \text{ が極} \quad \Leftrightarrow \quad (\exists k \in \mathbb{Z}) \quad \frac{\pi z}{h} = k\pi \qquad \Leftrightarrow \quad (\exists k \in \mathbb{Z}) \quad z = kh.$$

kh は φ の 1 位の極であり、

$$\operatorname{Res}(\varphi; kh) = \left. \frac{\cos(\pi z/h)}{(\sin(\pi z/h))'} \right|_{z=kh} = \frac{h}{\pi}.$$

留数定理により、

$$\int_{C_{c,N}} f(z) \cot \frac{\pi z}{h}\, dz = 2\pi i \sum_{k=-N}^{N} f(kh) \operatorname{Res}(\varphi; kh) = 2ih \sum_{k=-N}^{N} f(kh).$$

ゆえに

$$\sum_{k=-N}^{N} f(kh) = \frac{1}{2i} \int_{C_{c,N}} f(z) \cot \frac{\pi z}{h}\, dz.$$

この式で、$N \to \infty$ とすると、

$$T_h = \sum_{k=-\infty}^{\infty} f(kh) = \frac{1}{2i} \int_{-\infty}^{\infty} \left(-f(x+ic) \cot \frac{\pi(x+ic)}{h} + f(x-ic) \cot \frac{\pi(x-ic)}{h} \right) dx.$$

一方、Cauchy の積分定理より

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

であるから、

$$I = \frac{1}{2}\int_{-\infty}^{\infty} \left(f(x+ic) + f(x-ic)\right)\, dx.$$

ゆえに

$$T_h - I = \int_{-\infty}^{\infty} \left[f(x+ic)\frac{\exp\frac{2\pi i(x+ic)}{h}}{1-\exp\frac{2\pi i(x+ic)}{h}} - f(x-ic)\frac{\exp\frac{-2\pi i(x-ic)}{h}}{1-\exp\frac{-2\pi i(x-ic)}{h}} \right] dx$$

であるから、

$$\begin{aligned} |T_h - I| &\le \frac{\exp\left(-2\pi c/h\right)}{1-\exp\left(-2\pi c/h\right)} \int_{-\infty}^{\infty} \left(|f(x+ic)| + |f(x-ic)|\right) dx \\ &= \frac{\exp\left(-2\pi c/h\right)}{1-\exp\left(-2\pi c/h\right)} \Lambda(f,c). \end{aligned}$$

$c \to d-0$ とすると、(50) を得る。 ■

$T_{h,N}$ の誤差については、次の定理を得る。

命題 F.2 $f \in H(w,d)$ が補題 F.1 の条件 (a), (b) を満たすならば、

$$|I - T_{h,N}| \le \|f\| \left(\frac{\exp\left(-2\pi d/h\right)}{1-\exp\left(-2\pi d/h\right)} \Lambda(f, d-0) + h \sum_{|k|>N} w(kh) \right). \tag{51}$$

証明 まず

$$|I - T_{h,N}| \le |I - T_h| + |T_h - T_{h,N}|.$$

右辺第 1 項については、補題 F.1 で評価済みである。右辺第 2 項については、

$$T_h - T_{h,N} = h \sum_{|k|>N} f(kh) = h \sum_{|k|>N} w(kh)\frac{f(kh)}{w(kh)}$$

であるから

$$|T_h - T_{h,N}| \le h\,\|f\| \sum_{|k|>N} |w(kh)|.$$

以上から (51) を得る。 ■

F.2 DE 公式の誤差解析

(工事中)

例 F.3 $0 < A < \frac{\pi}{2d}$ を満たす A に対して、

$$w(z) := \frac{1}{\cosh(Az)}$$

とおくとき、

$$\sum_{|k|>N} w(kh) \le 4\frac{\exp\left(-Ah\left(N+1\right)\right)}{1-\exp\left(-Ah\right)}.$$

ゆえに

$$\frac{2\pi d}{h} = ANh$$

となるように h, N を選ぶと、

$$|I - T_{n,N}| \le C\,\|f\| \exp\left(-\sqrt{2\pi Ad}\sqrt{N}\right).$$

例 F.4

$$w(z) := \frac{d}{dz}\tanh\left(\frac{\pi}{2}\sinh z\right) = \frac{\frac{\pi}{2}\cosh z}{\cosh^2\left(\frac{\pi}{2}\sinh z\right)}$$

とおくと、$x \to \pm\infty$ のとき

$$w(x) \simeq \pi \exp\left(-\frac{\pi}{2}\exp|x| + |x|\right).$$

h, N を

$$\frac{2\pi d}{h} = \frac{\pi}{2}\exp(Nh)$$

と選ぶと

$$|I - T_{h,N}| \le C\,\|f\| \exp\left(-CN/\log N\right).$$

G　誤差の特性関数を Mathematica で描いてみる

まだ工事中で、整理されるまで時間がかかります。

図 12, 図 14 などを描くために利用した Mathematica のコマンドを参考までに載せておく。

G.1　複合 Simpson 公式

$$\begin{aligned}
&a = -1, \quad b = 1, \quad m = 10,\\
&h = \frac{b-a}{2m},\\
&x_j = a + jh \quad (j = 0, 1, \ldots, 2m),\\
&w_0 = w_{2m} = \frac{h}{3}, \quad w_{2j} = \frac{2h}{3} \quad (j = 1, 2, \ldots, m-1), \quad w_{2j-1} = \frac{4h}{3} \quad (j = 1, 2, \ldots, m),\\
&\Psi(z) = \operatorname{Log}\frac{z+1}{z-1} - \sum_{k=0}^{2m}\frac{w_k}{z - x_k}.
\end{aligned}$$

21 点複合 Simpson 公式の誤差の特性関数の描画

```
Clear[a, b, m, n, h, xs, w, Lambda, Psi]
a = -1; b = 1; m = 10; n = 2 m + 1; h = (b - a)/(2 m);
xs[j_] := x[j] = a + j*h
Table[xs[j], {j, 0, 2 m}]
w[0] = h/3.0; w[2 m] = h/3.0;
w[k_] := w[k] = If[EvenQ[k], 2 h/3.0, 4 h/3.0]
Table[w[k], {k, 1, 2 m - 1}]
Lambda[z_] = Sum[w[k]/(z - xs[k]), {k, 0, 2 m}]
Psi[z_] := Log[(z + 1)/(z - 1)] - Lambda[z]
g1=ContourPlot[Log10[Abs[Psi[x + I y]]], {x,-4,4}, {y,-4,4}, PlotLegends -> Automatic]
g2=Plot3D[Log10[Abs[Psi[x+I y]]], {x,-4,4}, {y,-4,4}, PlotRange -> All]
```

(細かいことだけれど、`Lambda[]` の定義で `:=` でなく、`=` を使うのが、効率上重要である。)

G.2 Gauss-Legendre 公式

Gauss-Legendre 公式は、有名な Legendre 多項式を用いた積分公式であるが、時間の関係でこれまで講義では説明していないが、積分公式の誤差の特性関数の絶対値の等高線の例として。

$P_n(x)$ を n 次 Legendre 多項式とする。

$$
\begin{aligned}
&a=-1, \quad b=1, \quad w(x)=1, \quad n=8,\\
&\{x_k\}_{k=1}^n = P_n(x) \text{ の根},\\
&w_k = \frac{2}{nP_{n-1}(x_k)P_n'(x_k)} = \frac{2\left(1-x_k^2\right)}{\left(nP_{n-1}(x_k)\right)^2},\\
&\Psi(z) = \operatorname{Log}\frac{z+1}{z-1} - \sum_{k=1}^{n}\frac{w_k}{z-x_k}.
\end{aligned}
$$

8 次 Legendre-Gauss 公式の誤差の特性関数の描画

```
n=8; ndigits=30;
xs=x/.NSolve[LegendreP[n,x]==0,x,ndigits];
ws=Table[2(1-xs[[k]]^2)/(n LegendreP[n-1,xs[[k]]])^2,{k,1,n}];
Lambda[z_]=Sum[ws[[k]]/(z-xs[[k]]),{k,1,n}];
Psi[z_]:=Log[(z+1)/(z-1)]-Lambda[z];
g1=ContourPlot[Log10[Abs[Psi[x+I y]]],{x,-4,4},{y,-4,4},PlotLegends->Automatic]
g2=Plot3D[Log10[Abs[Psi[x+I y]]], {x,-4,4}, {y,-4,4}, PlotRange -> All]
```

G.2.1 おまけ: Legendre 多項式とその零点を求めるアルゴリズム

杉原・室田 [2] の §11.2 に載っていた。

Legendre 多項式 $P_n(x)$ とその導関数 $P_n'(x)$ の計算

$n \in \mathbb{N}$, $x \in (-1,1)$ が与えられたとして、以下の漸化式で $P_n(x)$ が計算できる。

$$
\begin{aligned}
&P_0(x) = 1,\\
&P_1(x) = x,\\
&P_{k+1}(x) = \frac{(2k+1)xP_k(x) - kP_{k-1}(x)}{k+1} \quad (k=1,2,\cdots,n-1).
\end{aligned}
$$

微分係数 $P_n'(x)$ は、$P_n(x)$, $P_{n-1}(x)$ から次のように計算できる

$$
P_n'(x) = \frac{n\left(P_{n-1}(x) - xP_n(x)\right)}{1-x^2}.
$$

Legendre 多項式 $P_n(x)$ の零点の計算

$n \in \mathbb{N}$ が与えられたとして、$k = 1, 2, \ldots, n$ に対して、次のような Newton 法で $x_k^{(\infty)}$ を求める。

$$x_k^{(0)} = \cos\left(\frac{\pi\,(k - 1/4)}{n + 1/2}\right),$$

$$x_k^{(j+1)} = x_k^{(j)} - \frac{P_n\left(x_k^{(j)}\right)}{P_n'\left(x_k^{(j)}\right)} \quad (j = 0, 1, 2, \cdots).$$

サンプル・プログラム legendre.c

以下のプログラムの実行結果は Mathematica で確認できた。

```
// legendre.c --- Legendre 多項式，零点の計算
//   参考: 杉原・室田，数値計算法の数理，岩波書店

#define MAXN (100)
#include <stdio.h>
#include <math.h>

double pi = 0;
// Pn(x), Pn-1(x)
void compute_legendre(int n, double x, double *pn, double *pnm1)
{
  int k;
  double pkp1, pk = x, pkm1 = 1.0;
  for (k = 1; k < n; k++) {
    pkp1 = ((2 * k + 1) * x * pk - k * pkm1) / (k + 1);
    pkm1 = pk; pk = pkp1;
  }
  *pn = pk; *pnm1 = pkm1;
}

// Pn(x)=0 の根
void compute_legendre_zeros(int n, double *root)
{
  int k, j;
  double x, dx, pn, dv, pnm1;
  if (pi == 0)
    pi = 4.0 * atan(1.0);
  for (k = 1; k <= n; k++) {
    x = cos((pi * (k - 0.25)) / (n + 0.5));
    for (j = 0; j <= 10; j++) { // 3,4 回で収束するみたい
      compute_legendre(n, x, &pn, &pnm1);
      // 導関数
      dv = n * (pnm1 - x * pn) / (1.0 - x * x);
      dx =  pn / dv;
      x -= dx;
      if (fabs(dx / x) < 1e-15)
        break;
    }
    root[k-1] = x;
  }
}

// Pn(x)
double legendre(int n, double x)
```

```
{
  double pn, pnm1;
  compute_legendre(n, x, &pn, &pnm1);
  return pn;
}

int main(void)
{
  int i, n;
  double x, root[MAXN];
  n = 8;
  // Pn(0), Pn(0.1), Pn(0.2), ..., Pn(1)
  printf("P_%d(x) (x=0,0.1,..,1.0)\n", n);
  for (i = 0; i <= 10; i++) {
    x = i / 10.0;
    printf("%20.15f %20.15f\n", x, legendre(n, x));
  }
  // Pn(x)=0 の根
  compute_legendre_zeros(n, root);
  printf("root\n");
  for (i = 0; i < n; i++)
    printf("%20.15f\n", root[i]);

  return 0;
}
```

legendre.c の実行結果

```
$ cc legendre.c
$ ./a.out
P_8(x) (x=0,0.1,..,1.0)
   0.000000000000000    0.273437500000000
   0.100000000000000    0.180320721484375
   0.200000000000000   -0.039564800000000
   0.300000000000000   -0.239074591015625
   0.400000000000000   -0.266999300000000
   0.500000000000000   -0.073638916015625
   0.600000000000000    0.212339200000000
   0.700000000000000    0.306704346484375
   0.800000000000000   -0.016655300000000
   0.900000000000000   -0.409685903515625
   1.000000000000000    1.000000000000000
root
   0.960289856497536
   0.796666477413627
   0.525532409916329
   0.183434642495650
  -0.183434642495650
  -0.525532409916329
  -0.796666477413627
  -0.960289856497536
$
```

G.3 無限区間の台形公式

```
Clear[Phi,Lambda,Psi,NN,n,h]
Phi[z_] = If[Im[z]>0,-Pi I, Pi I]
Lambda[z_, h_] = Pi Cot[Pi z/h]
Table[Residue[Lambda[z, h], {z, k h}], {k, -5, 5}]
Psi[z_, h_] = Phi[z] - Lambda[z, h]

g1=ContourPlot[Log10[Abs[Psi[x + I y, 1/2]]], {x, -4, 4}, {y, -4, 4},
                  Contours -> 9,
                  ColorFunction -> (Hue[#] &), PlotLegends -> Automatic]

g2=ContourPlot[Log10[Abs[Psi[x + I y, 1/4]]], {x, -4, 4}, {y, -4, 4},
                  Contours -> 6,
                  ColorFunction -> (Hue[#] &), PlotLegends -> Automatic]

(* 以下はうまく出来ていない。工事中。)
Lambda[z_, h_, NN_] := Sum[h/(z - n h), {n, -NN, NN}]
Psi[z_, h_, NN_] := Phi[z] - Lambda[z, h, NN]

g3=ContourPlot[Log10[Abs[Psi[x + I y, 1/2, 8]]], {x, -4, 4}, {y, -4, 4},
               Contours -> 9,
               ColorFunction -> (Hue[#] &), PlotLegends -> Automatic]

g4=ContourPlot[Log10[Abs[Psi[x + I y, 1/4, 16]]], {x, -4, 4}, {y, -4, 4},
               Contours -> 9,
               ColorFunction -> (Hue[#] &), PlotLegends -> Automatic]
```

ContourPlot[ ] の末尾のオプションについては解説しない。気になる場合は、?Contours のように Mathematica に問い合わせてみること。

(もしかすると、$|I - I_h|$ が小さいことは Ψ_h の評価から導けるが、$|I - I_{h,N}|$ が小さいことを $\Psi_{h,N}$ の評価から導くことは出来ないのかもしれない。$|I_h - I_{h,N}|$ が小さいことを別に評価して、$|I - I_{h,N}| \leq |I - I_h| + |I_h - I_{h,N}|$ と評価するのかも。)

H $\displaystyle\int_a^b f(x)\,dx$ (a, b 有限) 用の DE 公式

$$I = \int_{-1}^{1} f(x)\,dx$$

を計算する場合に $x = \varphi(t)$ という変数変換で

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

と、積分区間 $(-1, 1)$ を $(-\infty, \infty)$ に変換するのだった。ただし

$$\varphi(t) := \tanh\left(\frac{\pi}{2}\sinh(t)\right). \tag{52}$$

ここでは一般の有界区間 (a, b) での積分

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

を計算する手順を考えよう。まず

$$x = pu + q, \quad p := \frac{b-a}{2}, \quad q := \frac{a+b}{2}$$

という変数変換で、積分区間 (a,b) を $(-1,1)$ に変換出来る。それから $u=\varphi(t)$ を施せば良い。一つの変数変換にまとめると、

$$x = p\varphi(t) + q, \quad p = \frac{b-a}{2}, \quad q = \frac{a+b}{2}. \tag{53}$$

ゆえに $dx = p\varphi'(t)dt$ であるから

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

こうして I を近似するための、次の公式を得る。

$$I_{h,N} = hp\sum_{n=-N}^{N} f\left(p\varphi(nh)+q\right)\varphi'\left(nh\right) \quad (h>0,\, N\in\mathbb{N}).$$

```
/* DE 公式による (a,b) における定積分の計算 */
double de2(ddfunction f, double a, double b, double h, double N)
{
  int n;
  double p, q;
  double t, S, dS;
  p = (b - a) / 2.0; q = (a + b) / 2.0;
  S = f(p * phi1(0.0) + q) * dphi1(0.0);
  for (n = 1; n <= N; n++) {
    t = n * h;
    dS = f(p * phi1(t) + q) * dphi1(t) + f(p * phi1(-t) + q) * dphi1(- t);
    S += dS;
  }
  return p * h * S;
}
```

example6.c は $\displaystyle\int_{-1}^{1}\sqrt{1-x^2}\,dx$, $\displaystyle\int_{-1}^{1}\frac{dx}{\sqrt{1-x^2}}$ を計算するプログラムであったが、$\displaystyle\int_{0}^{1}\sqrt{1-x^2}\,dx$, $\displaystyle\int_{0}^{1}\frac{dx}{\sqrt{1-x^2}}$ を計算するように書き直すと、次のようになる。

```
/*
 * example6_1.c --- DE 公式 on (a,b)
 *
 * (a,b) における積分を計算できる de2() を作る。
 *
 *         1
 *  I1 = ∫   (1-x^2)^{1/2}dx = π/4
 *         0
 *
 *         1
 *  I2 = ∫   (1-x^2)^{-1/2}dx = π/2
 *         0
 *
 *  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_1 example6_1.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 (a,b) における定積分の計算 */
double de2(ddfunction f, double a, double b, double h, double N)
{
  int n;
  double p, q;
  double t, S, dS;
  p = (b - a) / 2.0; q = (a + b) / 2.0;
  S = f(p * phi1(0.0) + q) * dphi1(0.0);
  for (n = 1; n <= N; n++) {
    t = n * h;
    dS = f(p * phi1(t) + q) * dphi1(t) + f(p * phi1(-t) + q) * dphi1(- t);
    S += dS;
    if (fabs(dS) < 1.0e-16) {
      break;
    }
  }
  return p * 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) // セーフティ・ネット
    return 0;
  else
      return 1.0 / sqrt(1.0 - x * x);
}

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

  /* |t|≦3 まで計算することにする */
  h = 1.0; N = 3;
  /* 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 = de2(f, 0.0, 1.0, h, N);
    printf("h=%f, I_h=%25.16f, I_h-I=%e\n", h, 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, pi / 4);

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

  return 0;
}
```

I　DE 公式の桁落ち対策

I.1　はじめに — リターン・マッチをしよう

広義積分 $\displaystyle\int_{-1}^{1} \frac{dx}{\sqrt{1-x^2}}$ に対して，素朴な DE 公式で計算してみたところ、ほどほどの結果を得た。

二重指数関数型変数変換 $\varphi\colon [-1,1] \to \mathbb{R},\ x = \varphi(t) = \tanh\left(\frac{\pi}{2}\sinh t\right)$ を用いて、

$$\int_{-1}^{1} f(x)\,dx = \int_{-\infty}^{\infty} f\left(\varphi(t)\right)\varphi'(t)\,dt$$

と置換積分して、それから台形公式適用して得られる

$$\int_{-\infty}^{\infty} f\left(\varphi(t)\right)\varphi'(t)\,dt \fallingdotseq h\sum_{n=-N}^{N} f\left(\varphi(nh)\right)\varphi'(nh)$$

という数値積分公式を用いたわけである。

被積分関数 $f(x) = \dfrac{1}{\sqrt{1-x^2}}$ が、$x = \pm 1$ を定義域に含み得ないことに注意する。

nh は有限の値なので、$\varphi(nh)$ は ± 1 に等しくならないため、f の定義域 $(-1,1)$ に属していることに注意すべきである。ところが素朴に数値計算すると、$\varphi(nh)$ が ± 1 に等しくなってしまって、計算が頓挫してしまう場合がある。そこまでひどくことにならなくても、$\varphi(nh)$ の精度が著しく低下してしまうおそれがある。

これらの問題点を解決したい。そのためには

- t が大きい場合に、$\varphi(t) - 1$ を精度よく計算する
- t が小さい場合に、$\varphi(t) + 1$ を精度よく計算する

ことが必要となって来る。

I.2 準備: 桁落ち対策用の tanhm1(), tanhp1() を作る

$x \gg 1$ のとき、$\tanh(x)$ を浮動小数点数を用いて計算すると、桁落ちして 1 になってしまう。実際、IEEE 754 の double で素朴にやると、`tanh(20.0)` が 1 になる。

$$\mathrm{tanhm1}(t) := \tanh(t) - 1 \tag{54}$$

を精度よく計算する `tanhm1()` という C 言語の関数を作りたい。

C 言語の数学関数ライブラリィには、$x \fallingdotseq 0$ に対して、 $e^x - 1$ を桁落ちを避けて精度よく計算するために、`expm1()` という関数が用意されている。その真似をしてみよう。

$$\mathrm{tanhm1}(t) = \tanh(t) - 1 = \frac{\sinh(t)}{\cosh(t)} - 1 = \frac{e^t - e^{-t}}{e^t + e^{-t}} - 1 = \frac{-2}{1 + e^{2t}} = \frac{-2e^{-2t}}{1 + e^{-2t}}. \tag{55}$$

オーバーフローしても例外が発生して止まったりしなければ、次のコードで $x = 354$ 辺りまで計算できる。

```
// tanhm1(t) の計算, 桁落ち対策版
// t=354 で -6.615106e-308, t=355 ではけた落ちして 0 になってしまう。
// exp(2*354)=3.02338*10^307, exp(2*355)=2.233994766162*10^308
double tanhm1(double t)
{
  return - 2.0 / (1.0 + exp(2.0 * t));
}
```

もしも、オーバーフローして異常終了するのを避けたければ、次のように引数の範囲をチェックすれば良い。

```
// 心配症な人向け
double tanhm1(double t)
{
  if (t <= 354.0) // もう少し大きくできる？
    return - 2.0 / (1.0 + exp(2.0 * t));
  else
    return 0.0;
}
```

オーバーフローは許さないが、アンダーフローは許すシステムであれば、$\dfrac{-2e^{-2t}}{1 + e^{-2t}}$ を使うことも考えられる。

なお、IEEE 754 の 80 ビット実数の long double を用いて、同様のことをすると、$t = 5678$ 位まで (そのとき $\mathrm{tanhm1}(t) \fallingdotseq -2.837223 \times 10^{-4932}$) 計算できるようになる。

$\tanh(x)$ の計算は、$x \ll -1$ のときも同様の問題を引き起こす。それに対処するには

$$\mathrm{tanhp1}(t) := \tanh(t) + 1 \tag{56}$$

を精度よく計算する `tanhp1()` という関数を用意すれば良い。

$$\mathrm{tanhp1}(t) = \frac{e^t - e^{-t}}{e^t + e^{-t}} + 1 = \frac{2}{1 + e^{-2t}} = \frac{2e^{2t}}{1 + e^{2t}}. \tag{57}$$

```
//  tanh(t)+1 の計算 (t ≪-1 の場合用)
double tanhp1(double t)
{
  if (t >= - 354)
    return 2.0 / (1.0 + exp(- 2.0 * t));
  else {
    printf("tahnp1(): %g は小さ過ぎる\n", t);
    return 0;
  }
}
```

I.3 $\int_{-1}^{1} f(x)\,dx$ の $x=1$ での桁落ち対策

ここでは、$f\colon [-1,1) \to \mathbb{R}$ であるが、f は 1 を特異点とする場合に、$\int_{-1}^{1} f(x)\,dx$ を計算することを考える。

このような場合、1 に近い x に対して、$f(x)$ を精度よく計算するのに注意が必要な場合が多い。例えば

$$f(x) = \frac{1}{\sqrt{4-(x+1)^2}}$$

とするとき、$x<1,\ x \fallingdotseq 1$ に対して $f(x)$ を浮動小数点演算で計算しようとすると、桁落ちが生じ、十分な精度が得られない。このような場合に、

$$x = \varphi(t) = \tanh\left(\frac{\pi}{2}\sinh(t)\right)$$

として、$t \gg 1$ である t に対して、

$$f\left(\varphi(t)\right)\varphi'(t)$$

を浮動小数点数を用いて十分な精度で計算したい。どうすれば良いか。

(注意: $x=\varphi(t)<1$ であっても、$x=\varphi(t)$ が 1 に近い場合は、分母が 0 に近くなって困るのでは？と考えるかもしれない。しかし、$t \gg 1$ であるときは、$\varphi'(t) \fallingdotseq 0$ であって、$|f\left(\varphi(t)\right)|$ が大きくても、$|f\left(\varphi(t)\right)\varphi'(t)|$ は小さいことに注意しよう。十分な相対精度で $f\left(\varphi(t)\right)$ を計算すれば良いのである。)

(58)
$$\xi(t) := \varphi(t) - 1 = \tanh\left(\frac{\pi}{2}\sinh(t)\right) - 1 = \mathrm{tanhm1}\left(\frac{\pi}{2}\sinh(t)\right),$$
$$g(y) := f(y+1)$$

とおくと、

$$\varphi(t) = \xi(t) + 1, \quad f\left(\varphi(t)\right) = f\left(\xi(t)+1\right) = g(\xi(t)).$$

ナンセンスのようにも思える式変形であるが、桁落ちを避けるには、これが役に立つ。

g は f を (いわゆる) 原点移動した関数である。

$f(x) = \dfrac{1}{\sqrt{4-(x+1)^2}}$ $(-1 \le x < 1)$ の場合には、$g(y)$ を次のような ($y \fallingdotseq 0$ の場合に) 桁落ちの発生しにくい式に変形できる。

$$g(y) = f(y+1) = \frac{1}{\sqrt{4-((y+1)+1)^2}} = \frac{1}{\sqrt{4-(y^2+4y+4)}} = \frac{1}{\sqrt{-y(y+4)}} \quad (-2 \le y < 0).$$

```
// y<0, y ≒ 0 のときに f(y+1) を計算する g(y)。f(x)=1/sqrt(4-(x+1)^2) の場合
double g(double y)
{
  return sqrt(- y * (y + 4.0));
}
//  tanh(t)-1 の計算 (t ≫ 1 の場合用)
double tanhm1(double t)
{
  return - 2.0 / (1.0 + exp(2.0 * t));
}
// φ (t) の計算
double phi(double t)
{
  return tanh(halfpi * sinh(t));
}
// ξ (t)=φ (t)-1 の計算 (t ≫ 1 の場合用)
double xi(double t)
{
  return tanhm1(halfpi * sinh(t));
}

..
  if (t < LARGE)  // t があまり大きくないとき
    s = f(phi(t)) * dphi(t);
  else
    s = g(xi(t)) * dphi(t);
```

LARGE をどのくらいにするかが問題だが、$\frac{\pi}{2}\sinh(3) \fallingdotseq 15.7$, $\frac{\pi}{2}\sinh(4) \fallingdotseq 42.7$ であるから、$t \leq 3$ くらいで切り替えたほうが良い。…と最初は考えたのだが、もっと少なくとも $t \leq 1$ で切り替えたほうが良いようであった ($t \leq 1/2$ でもきちんと動いた)。

I.3.1 試し切り

$$I = \int_{-1}^{1} \sqrt{4-(x+1)^2}\, dx = \pi, \quad J = \int_{-1}^{1} \frac{dx}{\sqrt{4-(x+1)^2}} = \frac{\pi}{2}.$$

$h = 1/8$ で誤差 0 を達成した。これはさすがに出来過ぎだが、$h = 1/2, 1/4$ で 10^{-6}, 10^{-12} 程度であったので、double の精度一杯の結果が得られるのが順等であると考えられる。

```
$ cc example6_3.c
$ ./a.out
test1 (sqrt(1-x^2) の積分)
h=1.000000, I_h=       3.2819762938412307, I_h-I=1.403836e-01
h=0.500000, I_h=       3.1417095353766036, I_h-I=1.168818e-04
h=0.250000, I_h=       3.1415926535957124, I_h-I=5.919265e-12
h=0.125000, I_h=       3.1415926535897940, I_h-I=8.881784e-16
h=0.062500, I_h=       3.1415926535897922, I_h-I=-8.881784e-16
h=0.031250, I_h=       3.1415926535897931, I_h-I=0.000000e+00
h=0.015625, I_h=       3.1415926535897940, I_h-I=8.881784e-16
h=0.007812, I_h=       3.1415926535897953, I_h-I=2.220446e-15
h=0.003906, I_h=       3.1415926535897913, I_h-I=-1.776357e-15
h=0.001953, I_h=       3.1415926535897833, I_h-I=-9.769963e-15
test2 (1/sqrt(1-x^2) の積分)
h=1.000000, I_h=       1.5748562482325401, I_h-I=4.059921e-03
h=0.500000, I_h=       1.5707944657275292, I_h-I=-1.861067e-06
h=0.250000, I_h=       1.5707963267935228, I_h-I=-1.373790e-12
h=0.125000, I_h=       1.5707963267948966, I_h-I=0.000000e+00
h=0.062500, I_h=       1.5707963267948961, I_h-I=-4.440892e-16
h=0.031250, I_h=       1.5707963267948966, I_h-I=0.000000e+00
h=0.015625, I_h=       1.5707963267948959, I_h-I=-6.661338e-16
h=0.007812, I_h=       1.5707963267948968, I_h-I=2.220446e-16
h=0.003906, I_h=       1.5707963267948966, I_h-I=0.000000e+00
h=0.001953, I_h=       1.5707963267948937, I_h-I=-2.886580e-15
$
```

```
/*
 * example6_3.c --- DE 公式
 *
 *          1
 *  I1 = ∫   (4-(x+1)^2)^{1/2}dx = π
 *        -1
 *
 *          1
 *  I2 = ∫   (4-(x+1)^2)^{-1/2} dx = π/2
 *        -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_3 example6_3.c
 *
 */

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

typedef double ddfunction(double);

double pi, halfpi;

//  tanh(t)-1 の計算 (t ≫ 1 の場合用)
double tanhm1(double t)
{
  if (t <= 354)
    return - 2.0 / (1.0 + exp(2.0 * t));
  else {
    printf("tahnm1(): %g は大きすぎる\n", t);
    return 0;
  }
}
```

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

/* ξ (t)=φ (t)-1 */
double xi(double t)
{
  return tanhm1(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, x=1 で桁落ち対策 */
double de2(ddfunction f, ddfunction g, 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);
    if (t <= 0.5)
      dS = dphi1(t) * (f(phi1(t)) + f(phi1(-t)));
    else
      dS = dphi1(t) * (g(xi(t)) + g(xi(-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(4.0 - sqr(x + 1.0));
}

double g1(double y)
{
  return sqrt(- y * (y + 4.0));
}

/* テスト用の被積分関数 その2 */
double f2(double x)
{
    return 1.0 / sqrt(4.0 - sqr(x + 1.0));
}

double g2(double y)
{
  return 1.0 / sqrt(- y * (y + 4.0));
}

void test(ddfunction f, ddfunction g, 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 = de2(f, g, h, N);
    printf("h=%f, I_h=%25.16f, I_h-I=%e\n", h, 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, g1, pi);

  printf("test2 (1/sqrt(1-x^2) の積分)\n");
  test(f2, g2, pi / 2);

  return 0;
}
```

I.4 一般化: $\int_a^b f(x)\,dx$ で $x = a, b$ が特異点の場合

この節では、一般の場合を考える。つまり $f\colon (a,b) \to \mathbb{R}$ で、f が a, b を特異点とする場合である。

まず、f が b を特異点にする場合を考える。

$$g(y) := f(x+b) \tag{59}$$

を導入する。

$$x = p\varphi(t) + q, \quad p = (b-a)/2, \quad q = (a+b)/2, \quad \varphi(t) = \tanh\left(\frac{\pi}{2}\sinh(t)\right) \tag{60}$$

という変数変換で (a,b) を $(-\infty,\infty)$ に変換すると、$p+q=b$ に注意して

$$f(x) = f\left(p\varphi(t)+q\right) = f\left(p\left(\varphi(t)-1\right)+p+q\right) = f\left(p\xi(t)+b\right) = g(p\xi(t)).$$

ただし

$$\xi(t) = \mathrm{tanhm1}\left(\frac{\pi}{2}\sinh(t)\right). \tag{61}$$

ゆえに

$$f(x)dx = pg\left(p\xi(t)\right)\varphi'(t)\,dt. \tag{62}$$

```
// y<0, y ≒ 0 のときに f(y+b) を計算する g(y)。
// f(x)=1/sqrt(1-x*x), b=1 の場合
double g(double y)
{
  return sqrt(- y * (y + 2.0));
}
// tanhm1(), phi(), xi() は前と同じ
..
  if (t > LARGE)  // t が大きいとき
    s = g(p * psi(t)) * dphi(t);
  else
    s = f(p * phi(t) + q) * dphi(t);
```

積分区間の左端 a が特異点である場合、

$$\eta(t) := \varphi(t) + 1 = \tanh\left(\frac{\pi}{2}\sinh(t)\right) + 1 = \mathrm{tanhp1}\left(\frac{\pi}{2}\sinh(t)\right), \tag{63}$$

$$H(y) := f(y + a), \tag{64}$$

を導入して、$q - p = a$ に注意して、

$$f(x) = f\left(p\varphi(t) + q\right) = f\left(p\left(\varphi(t) + 1\right) - p + q\right) = f\left(p\eta(t) + a\right) = H\left(p\eta(t)\right).$$

これから

$$f(x)\,dx = pH\left(p\eta(t)\right)\varphi'(t)\,dt. \tag{65}$$

```
// y>0, y ≒ 0 のときに f(y+a) を計算する H(y)。f(x)=1/sqrt(1-x*x), a=-1 の場合
double H(double y)
{
  return sqrt(y * (2.0 - y));
}
// tanhm1(), phi(), psi() は前と同じ
double xi(double t)
{
  return tanhp1(halfpi * sinh(t));
}
..
  if (t > 1.0)  // t が大きいとき
    s = g(p * xi(t)) * dphi(t);
  else if (t < - 1.0) // t が小さいとき
    s = H(p * eta(t)) * dphi(t);
  else // t が大きくも小さくもないとき
    s = f(p * phi(t) + q) * dphi(t);
```

I.4.1 試し切り

$$I = \int_{-2}^{2} \sqrt{4 - (x+1)^2}\; dx = \pi, \quad J = \int_{-2}^{2} \frac{dx}{\sqrt{4 - (x+1)^2}} = \frac{\pi}{2}.$$

```
$ cc example6_4.c
$ ./a.out
test1 (sqrt(4-x^2) の積分)
h=1.000000, I_h=       6.8500793170814553, I_h-I=5.668940e-01
h=0.500000, I_h=       6.2836404935324666, I_h-I=4.551864e-04
h=0.250000, I_h=       6.2831853071990160, I_h-I=1.942979e-11
h=0.125000, I_h=       6.2831853071795862, I_h-I=0.000000e+00
h=0.062500, I_h=       6.2831853071795862, I_h-I=0.000000e+00
h=0.031250, I_h=       6.2831853071795871, I_h-I=8.881784e-16
h=0.015625, I_h=       6.2831853071795774, I_h-I=-8.881784e-15
h=0.007812, I_h=       6.2831853071795916, I_h-I=5.329071e-15
h=0.003906, I_h=       6.2831853071795880, I_h-I=1.776357e-15
h=0.001953, I_h=       6.2831853071795782, I_h-I=-7.993606e-15
test2 (1/sqrt(4-x^2) の積分)
h=1.000000, I_h=       3.1435079789309328, I_h-I=1.915325e-03
h=0.500000, I_h=       3.1415926733057051, I_h-I=1.971591e-08
h=0.250000, I_h=       3.1415926535897940, I_h-I=8.881784e-16
h=0.125000, I_h=       3.1415926535897940, I_h-I=8.881784e-16
h=0.062500, I_h=       3.1415926535897936, I_h-I=4.440892e-16
h=0.031250, I_h=       3.1415926535897927, I_h-I=-4.440892e-16
h=0.015625, I_h=       3.1415926535897918, I_h-I=-1.332268e-15
h=0.007812, I_h=       3.1415926535897976, I_h-I=4.440892e-15
h=0.003906, I_h=       3.1415926535897958, I_h-I=2.664535e-15
h=0.001953, I_h=       3.1415926535897905, I_h-I=-2.664535e-15
```

```
/*
 * example6_4.c --- DE 公式
 *
 *          2
 *  I1 = ∫   (4-x^2)^{1/2}dx = 2 π
 *        -2
 *
 *          2
 *  I2 = ∫   (4-x^2)^{-1/2} dx = π
 *        -2
 *
 *  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_4 example6_4.c
 *
 */

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

typedef double ddfunction(double);

double pi, halfpi;

//  tanh(t)-1 の計算 (t ≫ 1 の場合用)
double tanhm1(double t)
{
  if (t <= 354)
    return - 2.0 / (1.0 + exp(2.0 * t));
  else {
    printf("tahnm1(): %g は大きすぎる\n", t);
    return 0;
  }
}
```

```
//  tanh(t)+1 の計算 (t ≪-1 の場合用)
double tanhp1(double t)
{
  if (t >= - 354)
    return 2.0 / (1.0 + exp(- 2.0 * t));
  else {
    printf("tahnp1(): %g は小さ過ぎる\n", t);
    return 0;
  }
}

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

/* ξ (t)=φ (t)-1 */
double xi(double t)
{
  return tanhm1(halfpi * sinh(t));
}

/* η (t)=φ (t)+1
double eta(double t)
{
  return tanhp1(halfpi * sinh(t));
}

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

/* φ' */
double dphi1(double t)
{
  // printf("dphi1: %g\n", halfpi * cosh(t) / sqr(cosh(halfpi * sinh(t))));
  return halfpi * cosh(t) / sqr(cosh(halfpi * sinh(t)));
}

/* DE 公式による (a,b) における定積分の計算, x=a,b で桁落ち対策 */
/* \int_a^ b f(x) dx
 *   f を原点移動した g(y):=f(y+a), H(y):=f(y+b)
 *   刻み幅 h, 和を取る範囲 N
 */
double de3(ddfunction f, ddfunction g, ddfunction H,
   double a, double b, double h, double N)
{
  int n;
  double p, q;
  double t, S, dS;
  p = (b - a) / 2.0; q = (a + b) / 2.0;
  S = 0.0;
  for (n = -N; n <= N; n++) {
    t = n * h;
    if (t > 0.5)
      dS = dphi1(t) * g(p * xi(t));
    else if (t < - 0.5)
      dS = dphi1(t) * H(p * eta(t));
    else
      dS = dphi1(t) * f(p * phi1(t) + q);
    S += dS;
  }
  return p * h * S;
```

```
}

/* テスト用の被積分関数 その１ */
double f1(double x)
{
  return sqrt(4.0 - sqr(x));
}

// g1(y)=f1(y+b), b=2
double g1(double y)
{
  return sqrt(- y * (y + 4.0));
}

// h1(y)=f1(y+a), a=-2
double h1(double y)
{
  return sqrt(y * (4.0 - y));
}

/* テスト用の被積分関数 その２ */
double f2(double x)
{
  return 1.0 / sqrt(4.0 - sqr(x));
}

double g2(double y)
{
  return 1.0 / sqrt(- y * (y + 4.0));
}

double h2(double y)
{
  return 1.0 / sqrt(y * (4.0 - y));
}

void test(ddfunction f, ddfunction g, ddfunction H, 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 = de3(f, g, H, - 2.0, 2.0, h, N);
    printf("h=%f, I_h=%25.16f, I_h-I=%e\n", h, IhN, IhN - I);
    h /= 2; N *= 2;
  }
}

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

  printf("test1 (sqrt(4-x^2) の積分)\n");
  test(f1, g1, h1, 2 * pi);

  printf("test2 (1/sqrt(4-x^2) の積分)\n");
  test(f2, g2, h2, pi);

  return 0;
}
```

I.5 リターン・マッチ

おっと、もともとの問題をちゃんと解けていることを見せないと。

```
/*
 * kadai1-4.c --- DE 公式
 *
 *        1
 *  I = ∫  (1-x^2)^{1/2}dx = π/2
 *       -1
 *
 *  端点に特異性があって、古典的な数値積分公式ではうまく計算出来ない。
 *  double exponential formula (DE 公式) ならば計算できる。
 *  しかし素朴にやると端点近傍で桁落ちが起きて、フルの精度が出ない。
 *  定番テクニックである原点移動を行い、桁落ち対策をしてみた。
 *
 *  コンパイル: cc -o kadai1-4 kadai1-4.c
 *
 */

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

typedef double ddfunction(double);

double pi, halfpi;

//  tanh(t)-1 の計算 (t ≫ 1 の場合用)
double tanhm1(double t)
{
  if (t <= 354)
    return - 2.0 / (1.0 + exp(2.0 * t));
  else {
    printf("tahnm1(): %g は大きすぎる\n", t);
    return 0;
  }
}

//  tanh(t)+1 の計算 (t ≪-1 の場合用)
double tanhp1(double t)
{
  if (t >= - 354)
    return 2.0 / (1.0 + exp(- 2.0 * t));
  else {
    printf("tahnp1(): %g は小さ過ぎる\n", t);
    return 0;
  }
}

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

/* ξ (t)=φ (t)-1 */
double xi(double t)
{
  return tanhm1(halfpi * sinh(t));
}

/* η (t)=φ (t)+1 */
```

```
double eta(double t)
{
  return tanhp1(halfpi * sinh(t));
}

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

/* φ' */
double dphi1(double t)
{
  // printf("dphi1: %g\n", halfpi * cosh(t) / sqr(cosh(halfpi * sinh(t))));
  return halfpi * cosh(t) / sqr(cosh(halfpi * sinh(t)));
}

/* DE 公式による (a,b) における定積分の計算, x=a,b で桁落ち対策 */
double de3(ddfunction f, ddfunction g, ddfunction H,
   double a, double b, double h, double N)
{
  int n;
  double p, q;
  double t, S, dS;
  p = (b - a) / 2.0; q = (a + b) / 2.0;
  S = 0.0;
  for (n = -N; n <= N; n++) {
    t = n * h;
    if (t > 0.5)
      dS = dphi1(t) * g(p * xi(t));
    else if (t < - 0.5)
      dS = dphi1(t) * H(p * eta(t));
    else
      dS = dphi1(t) * f(p * phi1(t) + q);
    S += dS;
  }
  return p * h * S;
}

/* テスト用の被積分関数 その2 */
double f(double x)
{
  return 1.0 / sqrt(1.0 - sqr(x));
}

double g(double y)
{
  return 1.0 / sqrt(- y * (y + 2.0));
}

double h(double y)
{
  return 1.0 / sqrt(y * (2.0 - y));
}

void test(ddfunction f, ddfunction g, ddfunction H, 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 = de3(f, g, H, - 1.0, 1.0, h, N);
```

```
    printf("h=%f, I_h=%25.16f, I_h-I=%e\n", h, IhN, IhN - I);
    h /= 2; N *= 2;
  }
}

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

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

  return 0;
}
```

```
$ cc kadai1-4.c
$ ./a.out
test (1/sqrt(1-x^2) の積分)
h=1.000000, I_h=       3.1435079789309328, I_h-I=1.915325e-03
h=0.500000, I_h=       3.1415926733057051, I_h-I=1.971591e-08
h=0.250000, I_h=       3.1415926535897940, I_h-I=8.881784e-16
h=0.125000, I_h=       3.1415926535897940, I_h-I=8.881784e-16
h=0.062500, I_h=       3.1415926535897936, I_h-I=4.440892e-16
h=0.031250, I_h=       3.1415926535897927, I_h-I=-4.440892e-16
h=0.015625, I_h=       3.1415926535897918, I_h-I=-1.332268e-15
h=0.007812, I_h=       3.1415926535897976, I_h-I=4.440892e-15
h=0.003906, I_h=       3.1415926535897958, I_h-I=2.664535e-15
h=0.001953, I_h=       3.1415926535897905, I_h-I=-2.664535e-15
$
```

J　補正台形公式についてのメモ

$I = \int_a^b f(x)\,dx$ に対する数値積分公式では、f の値のみ用い、f の導関数の値は使わないのが普通であるが、f' の値を使って良いならば、**補正台形公式**と呼ばれる

$$T_{N,\,\text{補}} := T_N - \frac{h^2}{12}\left(f'(b) - f'(a)\right)$$

がある。台形公式 T_N と比べて、$T_{N,\,\text{補}}$ では精度がどれくらい改善されるか、適当な被積分関数を選んで実験して調べよ。中点公式 M_N はどう補正すれば良いか。
理論的なことは、Euler-Maclaurin 展開がヒントになる。

台形公式を補正したものなので、台形公式や、それとほぼ同等の結果を与える中点公式、シンプソン公式と比べる実験をするのが良いだろう。例えば `example2.c` のようなプログラムをたたき台にしてみる。

Euler-Maclaurin 展開を見ると、補正台形公式を更に補正した

$$T_{N,\,\text{補補}} := T_N - \frac{h^2}{12}\left(f'(b) - f'(a)\right) + \frac{h^4}{720}\left(f'''(b) - f'''(a)\right)$$

を得ることも出来る。この場合は、被積分関数 f の 3 階導関数が必要になる。これで試したレポートを書いた人もいた。

例えば次のような関数 trapezoidal2() を作る (df() は f' を計算する関数)。

nc2.c で nc.c から加えた関数

```
/* 補正複合台形公式 */
double trapezoidal2(ddfunction f, ddfunction df, double a, double b, int N)
{
  double h;
  h = (b - a) / N;
  return trapezoidal(f, a, b, N) - h * h * (df(b) - df(a)) / 12.0;
}

/* 補正補正複合台形公式 */
double trapezoidal3(ddfunction f, ddfunction df, ddfunction dddf,
                    double a, double b, int N)
{
  double h;
  h = (b - a) / N;
  return trapezoidal(f, a, b, N) - h * h * (df(b) - df(a)) / 12.0 +
    (h*h*h*h) * (dddf(b) - dddf(a))/ 720.0;
}
```

中点公式の補正は分からなかった人も多いようだが、発見できた人もいて、

$$M_{N,\,\text{補}} := M_N + \frac{h^2}{24}\left(f'(b) - f'(a)\right)$$

とすれば良い。

ここでは、$I = \displaystyle\int_1^2 \log x \; dx = \log 4 - 1$ について、複合台形公式、複合中点公式、複合 Simpson 公式、補正複合台形公式、2 回補正した複合台形公式の結果を見てみよう。

```
/*
 * kadai1-3.c -- \int_0^1 log x dx
 */

#include <stdio.h>
#include <math.h>
#include "nc2.c"

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

double df(double x)
{
  return 1.0 / x;
}

double dddf(double x)
{
  return 2 / (x * x * x);
}

int main(void)
{
  int N;
  double a, b, I;
  double  M, T, S, T2, T3;
  a = 1.0; b = 2.0; I = log(4.0) - 1.0;
  printf("#   N       I-M_N           I-T_N         I-S_N       I-補  T_N        I-補 補
T_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);
    T2 = trapezoidal2(f, df, a, b, N);
    T3 = trapezoidal3(f, df, dddf, a, b, N);
    printf("%5d  %14e %14e %14e %14e %14e\n",
   N, I - M, I - T, I - S, I - T2, I - T3);
  }
  return 0;
}
```

```
./kadai1-3
#   N       I-M_N           I-T_N         I-S_N       I-補 T_N        I-補補 T_N
    2   -5.085309e-03   1.027501e-02   4.597590e-04  -1.416547e-04   1.025498e-05
    4   -1.293949e-03   2.594852e-03   3.479831e-05  -9.314956e-06   1.794014e-07
    8   -3.250044e-04   6.504512e-04   2.317654e-06  -5.904989e-07   2.898481e-09
   16   -8.134780e-05   1.627234e-04   1.474441e-07  -3.704164e-08   4.569012e-11
   32   -2.034302e-05   4.068779e-05   9.257558e-09  -2.317243e-09   7.153722e-13
   64   -5.086136e-06   1.017238e-05   5.792660e-10  -1.448613e-10   1.110223e-14
  128   -1.271558e-06   2.543122e-06   3.621459e-11  -9.054424e-12   1.110223e-16
  256   -3.178909e-07   6.357823e-07   2.263467e-12  -5.658807e-13   0.000000e+00
  512   -7.947283e-08   1.589457e-07   1.414979e-13  -3.530509e-14   5.551115e-17
 1024   -1.986821e-08   3.973643e-08   8.881784e-15  -2.109424e-15   1.110223e-16
 2048   -4.967053e-09   9.934107e-09   6.106227e-16  -2.775558e-16  -1.665335e-16
 4096   -1.241763e-09   2.483527e-09   2.220446e-16  -2.775558e-16  -2.775558e-16
 8192   -3.104412e-10   6.208816e-10   5.551115e-17  -1.110223e-16  -1.110223e-16
16384   -7.760981e-11   1.552213e-10  -2.775558e-16   8.881784e-16   8.881784e-16
32768   -1.940148e-11   3.880235e-11   6.106227e-16  -2.775558e-15  -2.775558e-15
65536   -4.851064e-12   9.704404e-12  -2.220446e-16   3.108624e-15   3.108624e-15
```

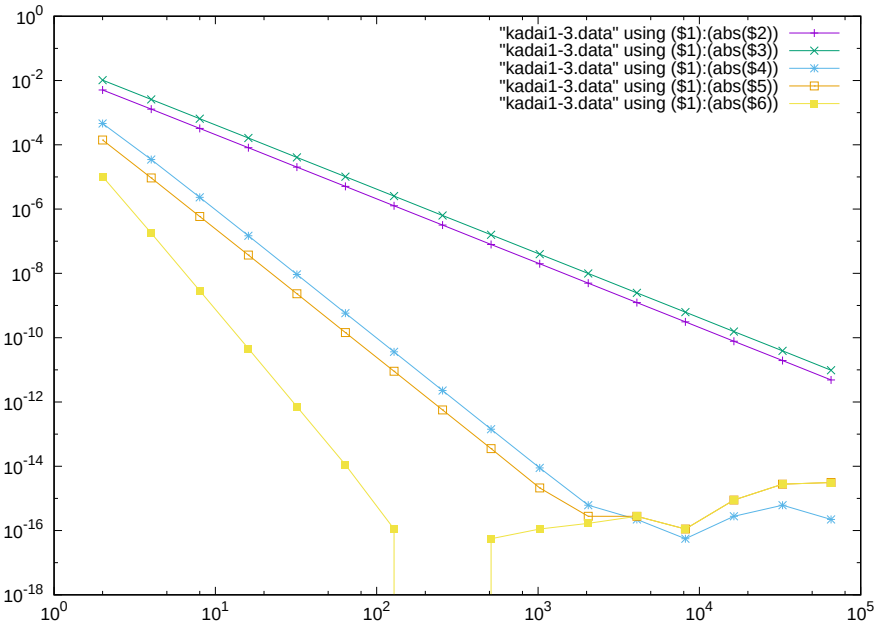

図 19: $\int_1^2 \log x\, dx$ を中点公式、台形公式、Simpson 公式、補正台形公式、補正を 2 回行った台形公式で計算したときの誤差

補正台形公式は、Simpson 公式と同様に、4 位の公式になっていることが分かる。

余談 J.1 N を大きくしたときに、誤差がこのように減衰するグラフは既に見せたはずだが、自分で計算すると、頭を使って考えてくれるらしくて、N が 10^3 を超えるあたりからグラフが直線にならず、ジグザグするのはなぜか？という質問を複数の人から受けた。百聞百見は一験にしかず。 ■

K　ガンマ関数 Γ の数値計算

$$\Gamma(x) = \int_0^\infty e^{x-1}e^{-t}\, dt.$$

変数 t についての積分である。積分範囲が $(0,\infty)$ で、$t \to \infty$ のとき、一重指数関数的に減衰する。

(66)
$$t = \varphi_4(s) := \exp(s - \exp(-s)) \quad (s \in \mathbb{R})$$

を使えば良い、ということだが。

素朴に計算すると、やはり x が 0 に近かったり、大きいと難しいみたい。

$\sum_{n=-N}^{N}$ でなく、$\sum_{n=-N_1}^{N_2}$ か？

関数等式 $\Gamma(x) = (x-1)\Gamma(x-1)$ を使うのか？

x が 0 に近いと大変。そうか $0 < x < 1$ のときは、

$$\Gamma(x) = \frac{\Gamma(x+1)}{x}$$

という関係を使って計算すれば良いか。

$f(x,t) = e^{-t}t^{x-1}$ という関数は、x を止めて t の関数と見たとき、$t = x - 1$ で最大となる。つまり x が大きくなると、重心が右に移動する感じだ。変数変換後の $f(x, \varphi_4(s))\varphi_4'(s)$ は、$s = s_p := \log x$ くらいにピークがある。

$I = \int_{-\infty}^{\infty} F(s)\,ds$ を普通は

$$I_{h,N} = h \sum_{n=-N}^{N} F(nh)$$

で計算するよりは

$$I_{h,N} = h \sum_{n=-N}^{N} F(s_p + nh)$$

と計算する方が良いのでは？

lgamma() はどうしているの？ソースを glibc[13] から持って来て、読んでみるとか。

```
/*
 * kadai1-2.c --- DE 公式
 */

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

typedef double ddfunction(double);

/* φ */
double phi4(double s)
{
  return exp(s - exp(- s));
}
/* φ' */
double dphi4(double s)
{
  double expms = exp(-s);
  return exp(s - expms) * (1.0 + expms);
}
/* テスト用の被積分関数 */
double x_gamma;
double f(double t)
{
  return exp(-t) * pow(t, x_gamma-1);
}

double mygamma(double x, double h)
{
  int n;
  double s1, s2, sp, S, dS;
  if (x < 1.0) {
    if (x <= 0.0) {
      fprintf(stderr, "error\n");
      return 0.0;
    }
    else
      return mygamma(x + 1.0, h) / x;
  }
  x_gamma = x;
  S = 0.0;
  sp = log(x);
```

[13] http://ftp.gnu.org/gnu/libc/

```
  S = f(phi4(sp)) * dphi4(sp);
  for (n = 1; n <= 10 / h; n++) {
    s1 = sp + n * h; s2 = sp - n * h;
    dS = f(phi4(s1)) * dphi4(s1) + f(phi4(s2)) * dphi4(s2);
    S += dS;
    if (fabs(dS / S) < 1.0e-20) {
      printf("n=%d\n", n);
      return h * S;
    }
  }
}

int main(void)
{
  int m;
  double x, I, h, IhN, error;

  printf("x: "); scanf("%lf", &x);
  I = tgamma(x);
  printf("Gamma(%g)=%20.16e\n", x, I);
  h = 1.0;
  /* h を半分にして double exponential formula で計算してゆく */
  for (m = 1; m <= 10; m++) {
    IhN = mygamma(x, h);
    error = fabs((I - IhN) / I);
    printf("h=%f, I_h=%20.16e, 相対誤差=%e\n", h, IhN, error);
    h /= 2;
  }
  return 0;
}
```

割と上手く計算できている。

x が大きいと、ピークがするどくなるのか、むしろ N が小さくて済むみたい。

$x = 120$ ($\Gamma(x) \fallingdotseq 5.6 \times 10^{196}$) でも、$h = \frac{1}{16}$, $N = 16$ で相対誤差 10^{-14} くらい。

あれ？`tgamma()` よりも精度が良かったりする？

L Euler の定数

$$\gamma = -\int_0^1 \log\log\frac{1}{x}\,dx.$$

$h = 1$, $N = 3$ ($|x| \leq 3$ で評価) から始めて、$H = 1/4$ で 5×10^{-14} 程度。

実は N が小さくて精度が出ない。しかし $h = 1$, $N = 4$ とすると、計算が破綻する。

$f(x) = -\log\log\frac{1}{x}$ $(0 < x < 1)$ の $x = 1$ の近くの評価を

```
double g(double y)
{
  return - log(-log1p(y));
}
```

により行うようにした。ここで `log1p(x)` は、$x \fallingdotseq 0$ のときに、$\log(1+x)$ を高精度に計算できるようにした関数である。

$x = 0$ の近くの評価は `-log(log(1.0/x))` で行って構わないわけである。原点だから。

```
/*
 * kadai1-1.c --- DE 公式
```

```
 */

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

typedef double ddfunction(double);

double pi, halfpi;

//  tanh(t)-1 の計算 (t ≫ 1 の場合用)
double tanhm1(double t)
{
  if (t <= 354)
    return - 2.0 / (1.0 + exp(2.0 * t));
  else {
    printf("tahnm1(): %g は大きすぎる\n", t);
    return 0;
  }
}

//  tanh(t)+1 の計算 (t ≪-1 の場合用)
double tanhp1(double t)
{
  if (t >= - 354)
    return 2.0 / (1.0 + exp(- 2.0 * t));
  else {
    printf("tahnp1(): %g は小さ過ぎる\n", t);
    return 0;
  }
}

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

/* ξ (t)=φ (t)-1 */
double xi(double t)
{
  return tanhm1(halfpi * sinh(t));
}

/* η (t)=φ (t)+1 */
double eta(double t)
{
  return tanhp1(halfpi * sinh(t));
}

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

/* φ' */
double dphi1(double t)
{
  // printf("dphi1: %g\n", halfpi * cosh(t) / sqr(cosh(halfpi * sinh(t))));
  return halfpi * cosh(t) / sqr(cosh(halfpi * sinh(t)));
}

/* DE 公式による (a,b) における定積分の計算, x=a,b で桁落ち対策 */
double de3(ddfunction f, ddfunction g, ddfunction H,
   double a, double b, double h, double N)
```

```
{
  int n;
  double p, q;
  double t, S, dS;
  p = (b - a) / 2.0; q = (a + b) / 2.0;
  S = 0.0;
  for (n = -N; n <= N; n++) {
    t = n * h;
    if (t > 0.5)
      dS = dphi1(t) * g(p * xi(t));
    else if (t < - 0.5)
      dS = dphi1(t) * H(p * eta(t));
    else
      dS = dphi1(t) * f(p * phi1(t) + q);
    S += dS;
  }
  return p * h * S;
}

/* テスト用の被積分関数 その 2 */
double f(double x)
{
  return - log(log(1.0/x));
}

// g(y)=f(y+b)
double g(double y)
{
  // return f(y + 1.0);
  return - log(- log1p(y));
}

// H(y)=f(y+a)
double h(double y)
{
  return f(y);
}

void test(ddfunction f, ddfunction g, ddfunction H, 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 = de3(f, g, H, 0.0, 1.0, h, N);
    printf("h=%f, I_h=%25.16f, I_h-I=%e\n", h, IhN, IhN - I);
    h /= 2; N *= 2;
  }
}

int main(void)
{
  double gamma = 0.57721566490153286061;

  pi = 4.0 * atan(1.0);
  halfpi = pi / 2.0;
  test(f, g, h, gamma);

  return 0;
}
```

L.1 DE 色々実験した後の悟り

端点に特異性がある場合も確かに計算できるが、まともな精度を出そうと思ったら、被積分関数に、端点が原点に来るような変数変換を施した関数を用意すべきである (昔から良く言われていることがようやく腑に落ちた感じ)。

その上でどういう範囲で和を取るかは慎重に考える必要がある。自動化出来ることが望ましいと考えるが、十分な完成度で出来るかどうか。そのあたりは大浦さんのコードを研究するのかもしれない。

参考文献

[1] 森正武：数値解析 第 2 版, 共立出版 (2002/2/25), 第 1 版は 1973 年に出版された。

[2] 杉原正顯, 室田一雄：数値計算法の数理, 岩波書店 (1994).

[3] 山本哲朗：数値解析入門 [新訂版], サイエンス社 (2003/6/1), 1976 年初版発行の定番本の待望の改訂版.

[4] 一松信：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。

[5] 篠原能材：数値解析の基礎, 日新出版 (初版 1978, 5 版 1997).

[6] 一松信：初等関数の数値計算, 教育出版 (1974).

[7] 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.

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

[9] 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.

[10] 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.

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

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

[13] 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.

[14] 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).

[15] 荒川恒男, 伊吹山知義, 金子昌信：ベルヌーイ数とゼータ関数, 牧野書店 (2001).

[16] 桂田祐史：応用複素関数, 現象数理学科での講義科目「応用複素関数」の講義ノート. http://nalab.mind.meiji.ac.jp/~mk/complex2/ouyou.pdf (2015〜).

[17] E. ハイラー, G. ヴァンナー：解析教程 上, 下, シュプリンガーフェアラーク東京 (1997), “Analysis by its History” の邦訳. Wanner は元は「ワナー」と綴っていた.

[18] 杉原正顯：DE 変換公式の最適性について, 京都大学数理解析研究所講究録, Vol. 585, pp. 150–175 (1986), http://www.kurims.kyoto-u.ac.jp/~kyodo/kokyuroku/contents/pdf/0585-09.pdf.

[19] Sugihara, M.: Optimality of the double exponential formula – functional analysis approach –, *Numerische Mathematik*, Vol. 75, pp. 379–395 (1997).

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

[21] 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.

[22] 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。

[23] 志村五郎：数学をいかに使うか, ちくま学芸文庫, 筑摩書房 (2010).

[24] 桂田祐史：IEEE754 倍精度浮動小数点数のフォーマット, http://nalab.mind.meiji.ac.jp/~mk/labo/text/ieee_format.pdf (2002).