数値積分

桂田 祐史

2014 年 5 月 27 日

目 次

第 1 章　数値積分についての概説 **3**

第 2 章　1 次元の数値積分法 **4**

2.1 概説 4
2.2 補間型公式 5
2.3 等間隔分点を用いる補間型積分公式 5
2.3.1 Newton-Cotes の公式 5
2.3.2 Maclaurin の公式 6
2.4 複合則 6
2.4.1 複合台形則 7
2.4.2 複合中点則 7
2.4.3 複合 Simpson 則 8
2.4.4 簡単な誤差解析 8
2.4.5 台形則、シンプソン則、中点則の関係 10
2.5 Gauss 型公式 10
2.6 Euler-Maclaurin 展開 12
2.6.1 Darboux の公式 13
2.6.2 Bernoulli 多項式 13
2.6.3 Euler-Maclaurin 展開 16
2.7 周期関数の 1 周期上の数値積分 16
2.8 解析関数の R 全体における定積分 17
2.9 2 重指数関数型公式 18
2.9.1 基本的なアイディア 18
2.9.2 具体的な積分公式 20
2.9.3 基本的な性質 23
2.10 計算例 24
2.10.1 以下のプログラムで共通して用いる関数の定義 24
2.10.2 台形則と中点則の関係 26
2.10.3 台形則, 中点則, Simpson 則の比較 28

2.10.4 数直線上の解析関数の数値積分 36
2.10.5 DE 公式 38

第 3 章 山本第 7 章「数値積分」から抜き書き **42**
3.1 数値積分公式 42

第 4 章 TO DO LIST **44**

第1章　数値積分についての概説

よく知られているように積分の計算はむつかしい。定積分を数値的に近似計算することを数値積分という。

この文書では、主に 1 次元の定積分

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

の計算法について説明する。

微分については、合成関数の微分法により、例えば初等関数の導関数は初等関数になるなど、機械的に計算できてしまうことが多い。特に高速微分法 (自動微分法とも言う) という効率的なアルゴリズムの存在が知られている。

多次元の積分に関しては、紙と鉛筆の計算においては、Fubini の定理

$$\iint_{A\times B} f(x,y)\,dxdy = \int_B \left(\int_A f(x,y)\,dx \right) dy$$

に基づく累次積分への帰着が使われることが多いが、この手順をそのまま応用して 1 次元の数値積分を繰り返すのは良くないとされている (らしい)。

(『応用数理』に多次元の数値積分についての論説があったように思う。探し出して参考文献表に加え、できれば内容の紹介を書くこと。)

第2章　1 次元の数値積分法

(この章の内容はまだかなり不完全である。必要な場合は、杉原・室田 [1] や森 [2] などで補うこと。やるべきこととして、

(1) 補間 (ただ今書きかけ)

(2) 直交多項式

(3) ベルヌイ数と Euler-Maclaurin (少し書いてあるが、ひどい出来)

(4) 高橋・森の理論 (ただ今書きかけ)

がある。... 何だ、全然足りない。重要性から言うと、高橋・森か。)

2.1　概説

Riemann 積分では積和の極限として積分を定義するわけであるが、

$$a \leq a_0 < a_1 < \cdots < a_n \leq b$$

のような分点 (標本点とも言う) と、重み $\{w_i\}_{i=0}^n$ を取って

$$I_{n+1} = \sum_{i=0}^{n} w_i f(a_i)$$

のような和を作って積分 I を近似するのが普通である。

2.2 補間型公式

相異なる $n+1$ 点 a_i $(i=0,\,1,\,\cdots,\,n)$ を標本点とする、関数 f の補間多項式 $f_n(x)$ は[1] 次式で与えられる (Lagrange の補間公式):

$$f_n(x)=\sum_{k=0}^{n}\frac{F_n(x)}{(x-a_k)F_n'(a_k)}f(a_k),\quad F_n(x)=(x-a_0)(x-a_1)\cdots(x-a_n).$$

このとき

$$\int_a^b f_n(x)\,dx=\sum_{k=0}^{n}w_k f(a_k),\quad w_k=\frac{1}{F_n'(a_k)}\int_a^b\frac{F_n(x)}{x-a_k}dx.$$

この値を $I=\int_a^b f(x)dx$ の近似値として採用する公式を補間型積分公式と呼ぶ。

2.3 等間隔分点を用いる補間型積分公式

ここで述べる公式は、高等学校の数学の教科書でも説明されているポピュラーなものである (実際に参考文献にいれるべし)。

2.3.1 Newton-Cotes の公式

補間型積分公式のうちで

$$a_k=a+\frac{b-a}{n}k\quad(k=0,1,\cdots,n)$$

とするのを **Newton-Cotes の公式**と総称する。

Newton-Cotes の公式で $n=1$ である

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

とするのを台形則 (**trapezoidal rule**) と呼ぶ。

Newton-Cotes の公式で $n=2$ である

$$S=I_3=\frac{b-a}{6}\left(f(a)+4f\left(\frac{a+b}{2}\right)+f(b)\right)$$

[1] f の補間多項式 $f_n(x)$ は $f_n(a_i)=f(a_i)$ $(i=0,1,\cdots,n)$ を満たす n 次多項式として特徴づけられる。

とするのを **Simpson** 則と呼ぶ。

次の命題は明らかである。

定理 2.3.1 被積分関数 f が n 次までの多項式のときは、n 次の Newton-Cotes の公式 I_{n+1} は正確な積分値を与える。

系 2.3.2 (1) 台形則は 1 次以下の多項式関数に関して正しい値を与える。

(2) Simpson 則は 2 次以下の多項式関数に関して正しい値を与える。

2.3.2 Maclaurin の公式

補間型積分公式のうちで

$$a_k = a + \frac{b-a}{n}\left(k - \frac{1}{2}\right) \quad (k = 1, \cdots, n)$$

とするのを **Maclaurin** の公式と総称する。

Maclaurin の公式で $n = 1$ である

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

とするのを中点則 (**midpoint rule**) とよぶ。

台形則、シンプソン則、中点則の間には、$S = (T + 2M)/3$ という関係がある (すぐに確かめられる)。

定理 2.3.3

系 2.3.4 中点則は...

2.4 複合則

Newton-Cotes の公式にしろ、Maclaurin の公式にしろ、n が大きい公式は今ではあまり使われない (等間隔標本点に基づく補間多項式は、次数が高いとき、もとの関数をあまり良く近似しないという **Runge** の現象がある — 後述)。

実際にこれらの公式を使用するときには、与えられた積分区間 $[a,b]$ をまず m 等分して、各小区間をさらに n 等分して、前節の公式を適用することが多い。これを複合則と呼ぶ。

2.4.1 複合台形則

$[a,b]$ を N 等分し、小区間の端点を分点とする。すなわち

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

で分点 x_j を定める。そして各小区間 $[x_{j-1}, x_j]$ で台形則を用いて計算したものを加えると、

$$\begin{aligned} T_N &= \frac{h}{2}\left[(f(x_0)+f(x_1)) + (f(x_1)+f(x_2)) + \cdots + (f(x_{N-2})+f(x_{N-1})) + (f(x_{N-1})+f(x_N))\right] \\ &= h\left[\frac{1}{2}f(x_0) + \sum_{j=1}^{N-1} f(x_j) + \frac{1}{2}f(x_N)\right]. \end{aligned}$$

後述の Euler-Maclaurin の公式を用いると

$$T_N - I \sim \frac{h^2}{12}(f'(b) - f'(a)) - \frac{h^4}{720}(f'''(b) - f'''(a)) + \cdots$$

2.4.2 複合中点則

$[a,b]$ を N 個の小区間に等分し、各小区間の中点を分点とする。すなわち

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

で分点 x_j を定める。そして各小区間で中点則を用いて計算したものを加えると、

$$M_N = h\left(f(x_1) + f(x_2) + \cdots + f(x_{N-1}) + f(x_N)\right) = h\sum_{j=1}^{N} f(x_j).$$

後述の Euler-Maclaurin の公式を用いると

$$M_N - I \sim -\frac{1}{2}\cdot\frac{h^2}{12}(f'(b) - f'(a)) + \frac{7}{8}\cdot\frac{h^4}{720}(f'''(b) - f'''(a)) - \cdots$$

2.4.3 複合 Simpson 則

$[a,b]$ を m 個の小区間に分割し、各小区間で Simpson 則を用いる。$[a,b]$ の $N=2m$ 等分点を x_j $(j=0,1,\cdots,N)$ とする。すなわち

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

で分点 x_j を定める。そして小区間 $[x_{2(j-1)},x_{2j}]$ $(j=1,2,\cdots,m)$ で Simpson 則を用いて計算したものを加えると、

$$\begin{aligned} S_{2m} &= \frac{2h}{6}\left\{[f(x_0)+4f(x_1)+f(x_2)]+\cdots+\left[f(x_{2(m-1)})+4f(x_{2m-1})+f(x_{2m})\right]\right\} \\ &= \frac{h}{3}\left[f(x_0)+2\sum_{j=1}^{m-1}f(x_{2j})+4\sum_{j=1}^{m}f(x_{2j-1})+f(x_{2m})\right]. \end{aligned}$$

(漸近公式は？)

2.4.4 簡単な誤差解析

より一般的な誤差解析の公式が山本 [6] に載っている。

補題 2.4.1 (台形則の誤差) $f\colon[a,b]\to\mathbf{R}$ を C^2-級の関数とするとき

$$h\overset{\text{def.}}{=}b-a,\quad \varepsilon=\int_a^b f(x)\,dx-\frac{b-a}{2}\left(f(a)+f(b)\right)$$

とおけば、以下の (1), (2) が成り立つ。

(1) $\varepsilon=-\dfrac{1}{2}\displaystyle\int_a^b f''(x)(x-a)(b-x)\,dx.$

(2) $-\dfrac{1}{12}h^3\max_{y\in[a,b]}f''(y)\le\varepsilon\le-\dfrac{1}{12}h^3\min_{y\in[a,b]}f''(y).$

(3) $\exists\xi\in(a,b)$ s.t. $\varepsilon=-\dfrac{1}{12}h^3f''(\xi).$

証明　(1) 証明すべき式の右辺を 2 回部分積分すればよい。また (2), (3) については $(x-a)(b-x)\ge0$ に注意すればよい。 ■

命題 2.4.2 (複合台形則の誤差) $f\colon [a,b] \to \mathbf{R}$ が C^2-級ならば、$\exists \xi \in (a,b)$ s.t.

$$I - T_n = -\frac{b-a}{12} h^2 f''(\xi), \quad h = \frac{b-a}{n}.$$

ただし

$$T_n = h\left(\frac{1}{2}f(a) + \sum_{j=1}^{n-1} f(a+jh) + \frac{1}{2}f(b)\right).$$

証明 $x_j = a + jh$ $(j = 0, 1, \cdots, n)$, さらに

$$\varepsilon_j \overset{\text{def.}}{=} \int_{x_j}^{x_{j+1}} f(x)\,dx - \frac{h}{2}\left(f(x_j) + f(x_{j+1})\right).$$

とおくと

$$I - T_n = \sum_{j=1}^{n} \varepsilon_j.$$

ゆえに上の補題から

$$-\frac{b-a}{12} h^2 \max_{y\in[a,b]} f''(\xi) \leq I - T_n \leq -\frac{b-a}{12} h^2 \min_{y\in[a,b]} f''(\xi).$$

これから明らかである。 ■

以下の二つの命題もほぼ同様に証明できる。

命題 2.4.3 (複合中点則の誤差) $f\colon [a,b] \to \mathbf{R}$ が C^2-級ならば、$\exists \xi \in (a,b)$ s.t.

$$I - M_n = \frac{b-a}{24} h^2 f''(\xi), \quad h = \frac{b-a}{n}.$$

ただし

$$M_n = h\sum_{j=1}^{n} f\left(a + \left(j - \frac{1}{2}\right)h\right).$$

命題 2.4.4 (複合 Simpson 則の誤差) $f\colon [a,b] \to \mathbf{R}$ が C^4-級ならば、$\exists \xi \in (a,b)$ s.t.

$$I - S_{2m} = -\frac{b-a}{180} h^4 f^{(4)}(\xi), \quad h = \frac{b-a}{2m}.$$

ただし

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

2.4.5 台形則、シンプソン則、中点則の関係

次の関係式は簡単だが、計算の手間を軽減するのに極めて有用である。

$$T_{2m} = \frac{T_m + M_m}{2}, \quad S_{2m} = \frac{T_m + 2M_m}{3} = \frac{4T_{2m} - T_m}{3}.$$

2.5 Gauss 型公式

(注意: この節は書きかけ。)

重みも分点もすべてパラメーターとした Gauss 型の公式について説明する。これは直交多項式の零点として分点が定められ、精度が極めて高い。

ここでは区間 $[-1,1]$ の Gauss-Legendre の公式のみ。$[-1,1]$ 上の実数値連続関数の空間 $X = C[-1,1]$ に

$$\langle f, g \rangle = \int_{-1}^{1} f(x)g(x)\,dx$$

で内積を導入する。

補題 2.5.1 (Legendre 多項式の直交性) n 次の Legendre 多項式を

$$p_n(x) \overset{\text{def.}}{=} \frac{1}{2^n n!} \frac{d^n}{dx^n}\left[(x^2-1)^n\right]$$

で定義すると、

$$\langle p_n, p_m \rangle = 0 \quad (n \neq m), \quad \langle p_n, p_n \rangle = \frac{2}{2n+1}.$$

系 2.5.2 (Legendre 多項式の漸化式)

$$(n+1)p_{n+1}(x) = (2n+1)xp_n(x) - np_{n-1}(x).$$

補題 2.5.3 (Christoffel-Darboux の定理)

$$\sum_{k=0}^{n}(2k+1)p_k(x)p_k(y) = \frac{n+1}{x-y}\begin{vmatrix} p_{n+1}(x) & p_n(x) \\ p_{n+1}(y) & p_n(y) \end{vmatrix}. \tag{2.1}$$

補題 2.5.4 (選点直交性) $p_n(x) = 0$ の根を $x_1, \cdots, x_n$ とすると、$i \neq j$ ならば、

$$\sum_{k=0}^{n-1}(2k+1)p_k(x_i)p_k(x_j) = 0.$$

定理 2.5.5 (Gauss の積分公式) $p_n(x) = 0$ の零点を節点 $x_1, \cdots, x_n$ とし、重みを

$$w_j = \frac{2}{np'_n(x_j)p_{n-1}(x_j)} \quad (j = 1, 2, \cdots, n)$$

と取ると、公式

$$\sum_{i=0}^{n} w_i f(a_i)$$

は $2n-1$ 次までの任意の多項式に対して正しい積分値を与える。

n	節点	重み
1	0	2
2	$-1/\sqrt{3}$, $1/\sqrt{3}$	1, 1
3	$-\sqrt{3/5}$, 0, $\sqrt{3/5}$	5/9, 8/9, 5/9

定理 2.5.6 () n 次 Gauss 公式

について以下のことが成り立つ。

(i) f が楕円

$$\mathcal{E}(\rho) = \{z \in \mathbf{C}; |z+1| + |z-1| = \rho + 1/\rho\}$$

(ただし $\rho > 1$) およびその内部を含む複素領域で正則のときには

$$|\text{誤差}| \leq \frac{\pi(\rho + 1/\rho)}{\rho^{2n+1}} \max_{z \in \mathcal{E}(\rho)} |f(z)| \quad (\text{十分大きい } n).$$

(ii) f が $\mathcal{E}(\rho)$ $(\rho > 1)$ およびその内部を含む複素領域で有理型であって、特異点は $\mathcal{E}(\rho)$ 上にある 1 位の極のみであるときには、

$$|\text{誤差}| \leq C/\rho^{2n} \quad (\text{十分大きい } n).$$

ただし C は f と ρ に依存する定数である。

証明

杉原・室田 [1] を見よ。 ■

2.6 Euler-Maclaurin 展開

(この節はまだたくさんの作業が必要。)

2.6.1 Darboux の公式

補題 2.6.1 (Darboux の公式) u は $\mathbf{C}$ 内の $a,\ z \in \mathbf{C}$ を結ぶ線分の近傍上で正則な関数、$p_n(t)$ は t の n 次多項式とするとき、

$$\begin{aligned} p_n^{(n)}(u(z)-u(a)) &= \sum_{r=1}^{n}(-1)^{r-1}(z-a)^r\left(p_n^{(n-r)}(1)u^{(r)}(z)-p_n^{(n-r)}(0)u^{(r)}(a)\right) \\ &\quad +(-1)^n(z-a)^{n+1}\int_0^1 p_n(t)u^{(n+1)}(a+t(z-a))\,dt. \end{aligned}$$

証明　積の微分法から

$$\frac{d}{dt}p_n^{(n-r)}(t)u^{(r)}(a+t(z-a)) = p_n^{(n-r+1)}(t)u^{(r)}(a+t(z-a))+(z-a)p_n^{(n-r)}(t)u^{(r+1)}(a+t(z-a))$$

が成り立つが、この両辺に $(-1)^r(z-a)^r$ をかけて、$r=1,\cdots,n$ まで加えると

$$\begin{aligned} &\frac{d}{dt}\sum_{r=1}^{n}(-1)^r(z-a)^r p_n^{(n-r)}(t)u^{(r)}(a+t(z-a)) \\ &\quad = -(z-a)p_n^{(n)}(t)u'(a+t(z-a))+(-1)^n(z-a)^{n+1}p_n(t)u^{(n+1)}(a+t(z-a)). \end{aligned}$$

$p_n^{(n)}(t)$ が定数 (ゆえに特に $p_n^{(n)}(0)$ に等しい) であることに注意して $t=0$ から $t=1$ まで積分すると結果を得る。 ■

2.6.2 Bernoulli 多項式

この小節の内容は森 [2] による。
実パラメーター t を持つ形式的冪級数

$$f_t(z) = \frac{ze^{tz}}{e^z-1}$$

の係数を $B_n(t)/n!$ とおこう[2]:

$$\frac{ze^{tz}}{e^z-1} = \sum_{n=0}^{\infty}\frac{B_n(t)}{n!}z^n \quad (|z|<2\pi).$$

[2] 分母は $z=2n\pi i$ $(n\in\mathbf{Z})$ で 0 になるが、分子に z があるので、$z=0$ は除去可能な特異点であることに注意しよう。

この $B_n(t)$ は t についての n 次多項式になるが、これを n 次の **Bernoulli** 多項式と呼び、

$$B_n \overset{\text{def.}}{=} B_n(0) \quad (n = 0, 1, \cdots)$$

を **Bernoulli** 数と呼ぶ。

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

$$B_0(x) = 1, \quad B_1(x) = x - \frac{1}{2}, \quad B_2(x) = x^2 - x + \frac{1}{6}, \quad B_3(x) = x^3 - \frac{3}{2}x^2 + \frac{1}{2}x, \cdots$$

注意 2.6.2 (異なる流儀) 上に書いた Beronoulli 数の定義は、関孝和 (1642–1708) や Jakob Bernoulli (1654–1705) の定義と一致するが、それとは異なる流儀もある。

1. B_1 の符号だけが異なるもの ($B_1 = -1/2$)。
   実はこの流儀で書いてある本の方が多いとか。どちらを採用しても B_n を $(-1)^n B_n$ で置き換えることで、他方に移る。

2. 奇数番目を飛ばす。

補題 2.6.3 (荒川・伊吹山・金子 [7] から)

(1) (B_j を求めるために使える漸化式)

$$\sum_{j=0}^{k} \binom{k+1}{j} B_j = k+1 \quad (k = 0, 1, 2, \cdots).$$

最初のいくつかを書いておくと、

$$\begin{aligned} 1 &= B_0, \\ 2 &= B_0 + 2B_1, \\ 3 &= B_0 + 3B_1 + 3B_2, \\ 4 &= B_0 + 4B_1 + 6B_2 + 4B_3, \\ 5 &= B_0 + 5B_1 + 10B_2 + 10B_3 + 5B_4, \\ &\vdots \end{aligned}$$

(2) $B_1 = 1/2$ を除き、奇数番目は 0 である: $B_{2\ell+1} = 0$ ($\ell \in \mathbf{N}$).

(3) $B_n(1) = B_n(0) = B_n$ ($n \geq 0$). (ただし B_1 については流儀に依存する。)

(4) (関・Bernoulli の公式)

$$\sum_{i=1}^{n} i^k = \sum_{j=0}^{k} \binom{k}{j} B_j \frac{n^{k+1-j}}{k+1-j}.$$

(5) $S_k(n)$ を

$$S_k(n) = \sum_{i=1}^{n} i^k$$

で定めると n の $k+1$ 次多項式になるので、自然に実変数関数に拡張できるが、

$$B_k(x) = S_k'(x-1).$$

さらに

$$B_k(x+1) - B_k(x) = kx^{k-1}, \quad B_k'(x) = B_{k-1}(x).$$

(6) $B_n(x)$ は

$$\int_x^{x+1} B_n(y)dy = x^n$$

を満たす唯一の多項式。

(7) (Bernoulli 数による表示)

$$B_n(x) = \sum_{j=0}^{n} (-1)^j \binom{n}{j} B_j x^{n-j}.$$

(8) k を 2 以上の偶数とするとき

2.6.3 Euler-Maclaurin 展開

定理 2.6.4 (Euler-Maclaurin 展開) f が $[a,b]$ で C^{2m}-級であれば、

$$\begin{aligned}\int_a^b f(x)dx &= h\left(\frac{1}{2}f(a)+\sum_{k=1}^{n-1}f(a+kh)+\frac{1}{2}f(b)\right)\\ &\quad -\sum_{r=1}^{m}\frac{h^{2r}B_{2r}}{(2r)!}\left(f^{(2r-1)}(b)-f^{(2r-1)}(a)\right)+R_m,\\ R_m &= \frac{h^{2m+1}}{(2m)!}\int_0^1 B_{2m}(t)\left(\sum_{k=0}^{n-1}f^{(2m)}(a+kh+ht)\right)dt,\quad h=(b-a)/n.\end{aligned}$$

ただし B_n は Bernoulli の数である。

2.7 周期関数の 1 周期上の数値積分

この節では「複合」という接頭辞を省いて単に、台形則、中点則、Simpson 則と呼ぶ。

$f\colon \mathbf{R}\to\mathbf{R}$ が周期 $b-a$ の周期関数である場合、1 周期にわたる積分

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

を台形則で計算することを考える。台形則の公式は

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

であったが、$f(a)=f(b)$ に注意すると

$$T_n=h\sum_{j=0}^{n-1}f(a+jh)=h\sum_{j=1}^{n}f(a+jh)$$

とも表せる。(この式からも容易にわかるように、周期関数の 1 周期にわたる積分を計算する場合、台形則と中点則は本質的には同じものである。)

f が C^{2m}-級ならば Euler-Maclaurin の公式から、

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

であるから、高精度であることが期待できる。
　実際に被積分関数の計算回数をそろえて比較すると、台形則は Simpson 則よりもはるかに高精度の値が得られることが多い。

2.8　解析関数の $\mathbf{R}$ 全体における定積分

　$f\colon \mathbf{R} \to \mathbf{R}$ が解析関数であるとする (つまり $\mathbf{R} \subset \mathbf{C}$ と見なしたとき、f は $\mathbf{C}$ における $\mathbf{R}$ の近傍で正則な関数に拡張できる)。このとき

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

を数値積分することを考える。
　$h > 0$ を刻み幅とする台形公式を

$$T_h \overset{\text{def.}}{=} h \sum_{n=-\infty}^{\infty} f(nh)$$

で定義する。実際の計算では十分大きな N を取って

$$T_{h,N} \overset{\text{def.}}{=} h \sum_{n=-N}^{N} f(nh)$$

で T_h の代用とする。
　ある意味で台形公式は最適な公式であることが証明できる。
　次の二つの命題の証明は杉原・室田 [1]

定理 2.8.1 () $\mathcal{D}(d) = \{z \in \mathbf{C}; |\Im z| < d\}$ で正則な関数 f が、次の二条件を満足する。

(i) $\forall c \in (0, d)$ に対して、

$$\Lambda(f, c) \stackrel{\text{def.}}{=} \int_{\mathbf{R}} (|f(x - ic)| + |f(x + ic)|)\, dx$$

が存在し、極限

$$\Lambda(f, d - 0) \stackrel{\text{def.}}{=} \lim_{c \uparrow d} \Lambda(f, c)$$

が有限確定であるとする。

(ii) $\forall c \in (0, d)$ 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、

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

(ここに π を 240 億桁近似するという、一見不思議な公式を！)

2.9 2 重指数関数型公式

2.9.1 基本的なアイディア

高橋秀俊, 森正武の研究として名高い **2** 重指数関数型積分公式 (**double exponential formula**) を解説する。

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

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 $\varphi\colon \mathbf{R} \to (a, b)$ を用いて変数変換

$$x = \varphi(t)$$

を施すと

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

この積分に台形公式を適用すると

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

φ をどう選択するのが良いだろうか？話を簡単にするために h は固定しておくことにする。I_h は無限和なので、実際の計算では

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

で代用することを考えると、

$$\varepsilon_{\mathrm{t}} \overset{\text{def.}}{=} I_h - I_{h,N}$$

という誤差 (「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、$\varphi(t)$ 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 $\varphi(t)$ が減衰すると、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

$$\Delta I_h \overset{\text{def.}}{=} I - I_h$$

と ε_{t} 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、

$$\varphi'(t) \fallingdotseq \exp(-C\exp|t|) \quad (|t| \to \infty) \tag{2.2}$$

の形であるときにある意味で最適な公式が得られる (高橋秀俊&森正武) [3]。

念のため: (2.2) 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。

[3] その後、杉原正顯氏によって、この最適性は定理の形で厳密に述べられるようになった。

2.9.2　具体的な積分公式

有限区間上の積分

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

に対しては

$$\varphi(t) \overset{\text{def.}}{=} \tanh\left(\frac{\pi}{2}\sinh t\right) \quad (t \in \mathbf{R})$$

とおいて、変数変換 $x = \varphi(t)$ を施す。

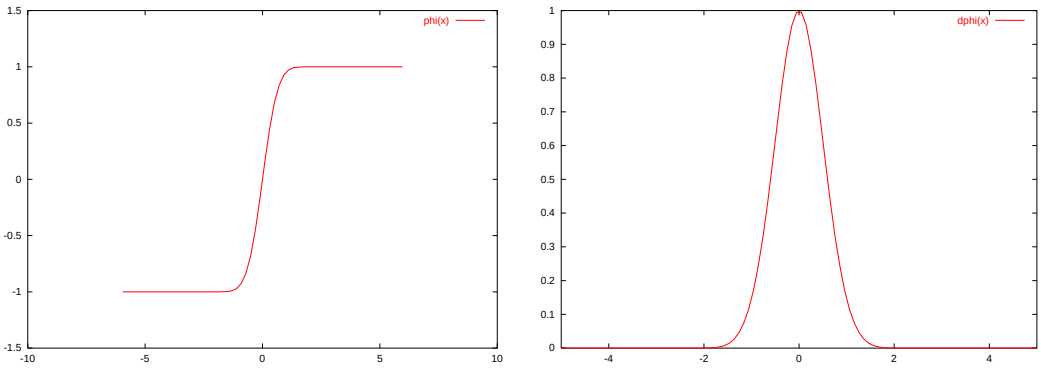
$$\varphi'(t) = \frac{\pi}{2}\frac{\cosh t}{\cosh^2(\pi/2\sinh t)}$$

であり、もちろん

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

念のため公式を書いておくと

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



$\mathbf{R}$ 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、

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

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

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

この場合は、$t \to \pm\infty$ のとき $\varphi'(t)$ は 減衰しないが、$f(\varphi(t))\varphi'(t)$ 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

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

半無限区間上の減衰の緩い関数の積分

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

において f の減衰が代数的な場合は、

$$\varphi(t) \stackrel{\text{def.}}{=} \exp(\pi \sinh t) \quad (t \in \mathbf{R})$$

とおいて、変数変換 $x = \varphi(t)$ を施す。

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

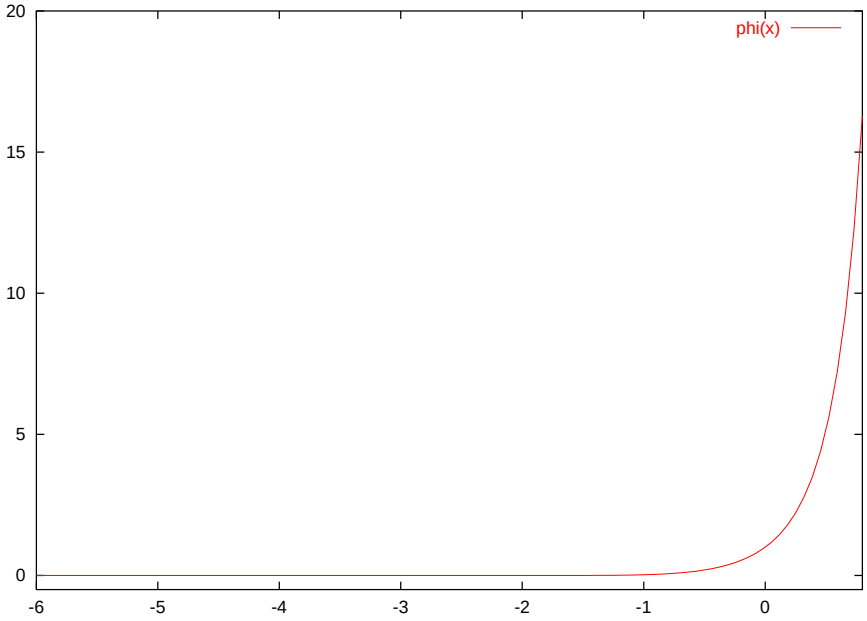
であり、

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

$t \to \infty$ のとき $\varphi'(t)$ は 減衰しないが、$f(\varphi(t))\varphi'(t)$ 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

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



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、

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

のような場合、

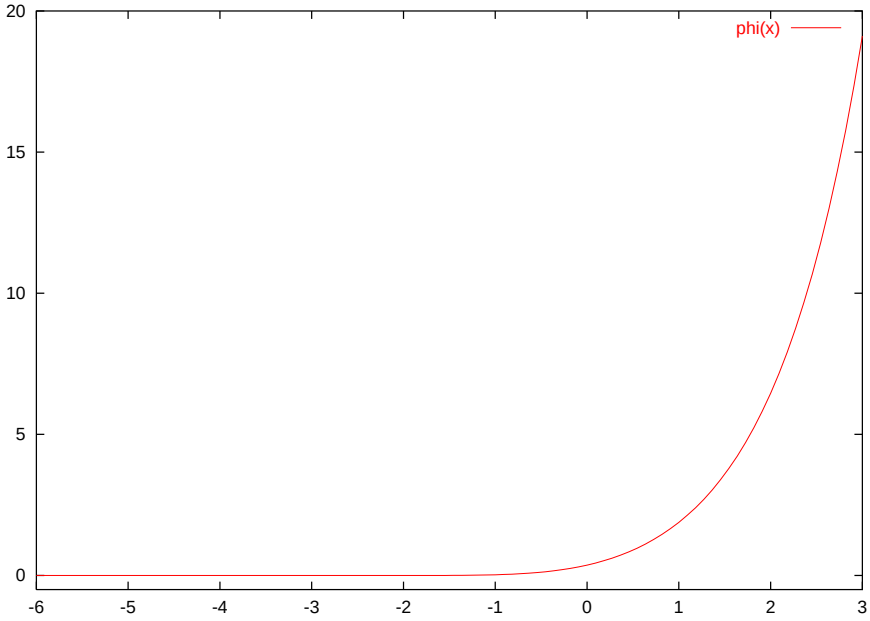
$$\varphi(t) = \exp(t - \exp(-t))$$

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

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

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

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



2.9.3 基本的な性質

- 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
  - 同じ手間で何桁も良い
  - 同じ桁数を得るのに手間が桁違い
- Gauss 型公式と比べても

  - 自動積分が出来るのは有利
  - 分点や重みが計算しやすい
- 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、使用上の注意が必要

2.10 計算例

以下 C 言語で書いたプログラムとその実行結果を示す。

2.10.1 以下のプログラムで共通して用いる関数の定義

nint.h

```
/*
 * nint.h --- ^^ca^^a3喉□□□§ , ^^ca^^a3喉□□□□§ , ^^ca^^a3喉 Simpson §
 */

typedef double rrfunction(double);

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

nint.c

```
/*
 * nint.c --- ^^ca^^a3喉□ □ □§ , ^^ca^^a3喉□ □ □ □§ , ^^ca^^a3喉 Simpson §
 *
 *  コア^^d1^^a5ぅ□: gcc -c nint.c
 */

#include <math.h>
#include "nint.h"

/* ^^d8^^bf □ f あ [a,b] ^^cb^^a4 □韻□摺^^ca^^acあ^^ca^^a3喉□ □ □§ ^^cb^^a4
□ □ □ □ □摺^^ca^^ac T_m */
double trapezoidal(rrfunction f, double a, double b, int m)
{
  int j;
  double h = (b - a)/ m, T = (f(a) + f(b)) / 2;
  for (j = 1; j < m; j++) T += f(a + j * h);
  T *= h;
  return T;
}

/* ^^d8^^bf □ f あ [a,b] ^^cb^^a4 □韻□摺^^ca^^ac あ^^ca^^a3 喉□ □ □ □§
^^cb^^a4 □ □ □ □ □摺^^ca^^ac M_m */
double midpoint(rrfunction f, double a, double b, int m)
{
  int j;
  double h = (b - a) / m, C = 0.0;
  for (j = 1; j <= m; j++) C += f(a + (j - 0.5) * h);
  C *= h;
  return C;
}

/* ^^d8^^bf □ f あ [a,b] ^^cb^^a4 □韻□摺^^ca^^ac あ^^ca^^a3 喉 Simpson §
^^cb^^a4 □ □ □ □ □摺^^ca^^ac S_{2m} */
double simpson(rrfunction f, double a, double b, int m)
{
  return (trapezoidal(f, a, b, m) + 2 * midpoint(f, a, b, m)) / 3;
}
```

nint.c をコンパイルして nint.o を作っておく。

nint.o の作り方

```
gcc -O -c nint.c
```

2.10.2　台形則と中点則の関係

例 2.10.1

$$\int_1^2 \frac{dx}{x} = \log 2$$

の計算を通して、$T_{2m} = (T_m + M_m)/2$ という関係が成り立つことの確認をしてみよう。

nint0.c

```
/*
 * nint0.c --- ^^ca^^a3    喉 □  □  □§  T_m, ^^ca^^a3    喉 □  □  □  □§
M_m ^^cb^^a4^^c4^^a4い□
 *               T_{2m} = (T_m+M_m)/2
 *             が  成  あ  □  犬^^c4^^a4       海^^c8^^a4       γ  □
^^c7^^a7 (f(x)=1/x あ [1,2] ^^cb^^a4 □韻□ □ □摺^^ca^^ac)
 *
 *  コア^^d1^^a5ぅ□: gcc -o nint0 nint0.c nint.o
 *    ただし nint.o あ gcc -c nint.c ^^c8^^a4靴^^c6^^bd □ □ □靴^^c6^^a4 □ □
 */

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

rrfunction f;

int main()
{
  int m;
  double Tm, Mm, T2m, mean;

  printf("#   m\t □ □繊Tm\t □ □ □澄Mm\t(Tm+Mm)/2\t\tT_{2m}\t\t差\n");
  for (m = 1; m <= (1 << 16); m *= 2) {
    /* Tm: □ □繊, Mm: □ □ □澄, S: Simpson §  */
    Tm = trapezoidal(f, 1.0, 2.0, m);
    T2m = trapezoidal(f, 1.0, 2.0, 2 * m);
    Mm = midpoint(f, 1.0, 2.0, m);
    mean = (Tm + Mm) / 2;
    printf("%5d %18.15f%18.15f %18.15f%18.15f %e\n",
           m, Tm, Mm, T2m, mean, fabs(T2m - mean));
  }
  return 0;
}

/* □ □摺^^ca^^ac^^d8^^bf □  */
double f(double x)
{
  return 1 / x;
}
```

nint0 の実行結果

```
#    m    □□繊 Tm          □□□澄 Mm          (Tm+Mm)/2               T_{2m}              差
     1  0.750000000000000  0.666666666666667  0.708333333333333  0.708333333333333  0.000000e+00
     2  0.708333333333333  0.685714285714286  0.697023809523809  0.697023809523809  0.000000e+00
     4  0.697023809523809  0.691219891219891  0.694121850371850  0.694121850371850  0.000000e+00
     8  0.694121850371850  0.692660554043203  0.693391202207527  0.693391202207527  1.110223e-16
    16  0.693391202207527  0.693025214330971  0.693208208269249  0.693208208269249  1.110223e-16
    32  0.693208208269249  0.693116669497558  0.693162438883403  0.693162438883403  1.110223e-16
    64  0.693162438883403  0.693139551572812  0.693150995228108  0.693150995228108  3.330669e-16
   128  0.693150995228108  0.693145273236777  0.693148134232443  0.693148134232442  8.881784e-16
   256  0.693148134232443  0.693146703724379  0.693147418978410  0.693147418978411  1.332268e-15
   512  0.693147418978410  0.693147061350755  0.693147240164583  0.693147240164582  6.661338e-16
  1024  0.693147240164583  0.693147150757630  0.693147195461108  0.693147195461106  1.887379e-15
  2048  0.693147195461108  0.693147173109364  0.693147184285235  0.693147184285236  1.332268e-15
  4096  0.693147184285235  0.693147178697300  0.693147181491270  0.693147181491268  2.442491e-15
  8192  0.693147181491270  0.693147180094283  0.693147180792774  0.693147180792776  1.887379e-15
 16384  0.693147180792774  0.693147180443531  0.693147180618150  0.693147180618153  2.886580e-15
 32768  0.693147180618150  0.693147180530836  0.693147180574505  0.693147180574493  1.176836e-14
 65536  0.693147180574505  0.693147180552671  0.693147180563600  0.693147180563588  1.265654e-14
```

当然のことながら、丸め誤差の程度で一致する (実験に用いた C 言語処理系の double の精度は 10 進 16 桁程度)。

2.10.3 台形則, 中点則, Simpson 則の比較

例 2.10.2 (滑らかな関数の積分) やはり

$$I = \int_1^2 \frac{dx}{x} = \log 2 = 0.69314718\cdots$$

を、3 つの方法で計算して、結果を比較する。

誤差の表

```
#   N    □ □ 繊 □ θ ^^ed^^ba^^b9   □ □ □ 澄 □ θ ^^ed^^ba^^b9  Simpson § う
^^ed^^ba^^b9
    1   -5.685282e-02   2.648051e-02    -1.297264e-03
    2   -1.518615e-02   7.432895e-03    -1.067877e-04
    4   -3.876629e-03   1.927289e-03    -7.350095e-06
    8   -9.746698e-04   4.866265e-04    -4.722595e-07
   16   -2.440216e-04   1.219662e-04    -2.972988e-08
   32   -6.102771e-05   3.051106e-05    -1.861510e-09
   64   -1.525832e-05   7.628987e-06    -1.163973e-10
  128   -3.814668e-06   1.907323e-06    -7.275514e-12
  256   -9.536725e-07   4.768356e-07    -4.551914e-13
  512   -2.384185e-07   1.192092e-07    -2.797762e-14
 1024   -5.960464e-08   2.980232e-08    -2.220446e-15
 2048   -1.490116e-08   7.450581e-09    -2.220446e-16
 4096   -3.725290e-09   1.862645e-09     2.220446e-16
 8192   -9.313247e-10   4.656625e-10     2.220446e-16
16384   -2.328292e-10   1.164144e-10    -1.110223e-16
32768   -5.820444e-11   2.910883e-11     4.329870e-15
65536   -1.455958e-11   7.274403e-12    -3.663736e-15
```

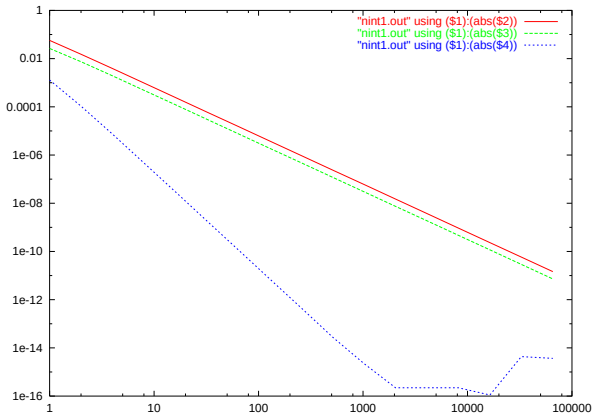
数表では今一つ分かりにくいでの、両側対数グラフにプロットしてみる。ここで用いているグラフ描画ソフト gnuplot については、

http://nalab.mind.meiji.ac.jp/~mk/labo/howto/index.html#gnuplot

特に

『gnuplot 入門』 by 桂田祐史
http://nalab.mind.meiji.ac.jp/~mk/labo/howto/intro-gnuplot/

を参照せよ。



gnuplot 用プログラム show_error1.gp

```
# show_nint1.gp --- gnuplot □^^d1^^a5^^d7^^a5^^ed^^a5^^b0ア□
#   ./nint1 > nint1.out ^^c8^^a4 靴^^c6^^ba □^^c3^^a4 織^^c7^^a1 璽燭□□
^^c9^^a4□^^c7^^a5哀□^^d5^^a4□□舛
#   窟^^cc^^a4□ nint1.eps ^^cb^^bd□□□
set logscale xy
plot "nint1.out" using ($1):(abs($2)) with lines, \
     "nint1.out" using ($1):(abs($3)) with lines, \
     "nint1.out" using ($1):(abs($4)) with lines
pause -1 "終入 すあ^^cb^^a4^^cf^^a5^^ea^^a5^^bf ーア^^f2^^b2^^a1^^a4靴^^c6^^a4
□世気"
set term postscript eps color
set output "nint1.eps"
replot
```

nint1.c

```
/*
 * nint1.c --- 1/x あ [1,2] ^^cb^^a4 □韻□ □ □摺^^ca^^ac あ
 *           ^^ca^^a3 喉□ □ □§ , ^^ca^^a3 喉□ □ □ □§ , ^^ca^^a3 喉 Simpson §
^^c7^^b7^^d7^^bb 擦靴^^c6^^b8^^ed^^ba^^b9 あ□ □ □
 *
 *  コア^^d1^^a5う□: gcc -o nint1 nint1.c nint.o -lm
 *    ただし nint.o あ gcc -c nint.c ^^c8^^a4 靴^^c6^^bd □ □ □靴^^c6^^a4 □ □
 */

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

rrfunction f;

int main()
{
  int N;
  double I = log(2.0), Tm, Mm, S2m;

  printf("#   N    □ □繊□θ ^^ed^^ba^^b9   □ □ □澄□θ ^^ed^^ba^^b9   Simpson §
う^^ed^^ba^^b9\n");
  for (N = 1; N <= (1 << 16); N *= 2) {
    /* Tm: □ □繊, Mm: □ □ □澄, S2m: Simpson §  */
    Tm = trapezoidal(f, 1.0, 2.0, N);
    Mm = midpoint(f, 1.0, 2.0, N);
    S2m = (Tm + 2 * Mm) / 3;
    printf("%5d\t%e\t%e\t%e\n", N, I - Tm, I - Mm, I - S2m);
  }
  return 0;
}

/* □ □摺^^ca^^ac^^d8^^bf □  */
double f(double x)
{
  return 1 / x;
}
```

これから

$$|I - S_{2m}| \ll |I - T_m|, |I - M_m|.$$

より詳しくは

$$I - T_m = O\left(\frac{1}{m^2}\right), \quad I - M_m = O\left(\frac{1}{m^2}\right), \quad I - S_{2m} = O\left(\frac{1}{m^4}\right)$$

という挙動を示していることが分る。また

$$(I - T_m) : (I - M_m) \fallingdotseq 2 : (-1)$$

となっていることも分かる。 ■

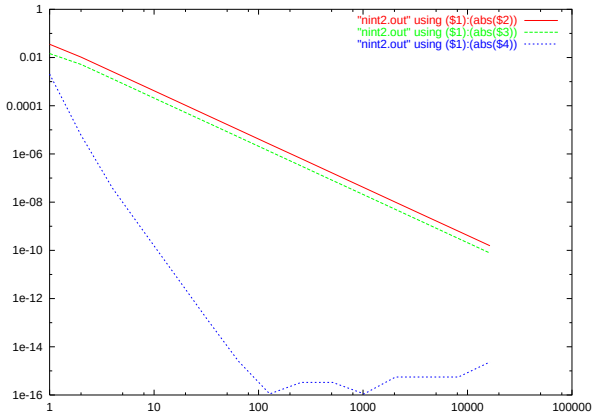
例 2.10.3 (Simpson 則がやけに高精度な例)

$$I = \int_0^1 \frac{dx}{1+x^2}$$

に対して $|I - S_n|$ は異常に小さくなることが知られている。

誤差の表

# N	□ □ 繊 □ θ ^^ed^^ba^^b9	□ □ □ 澄 □ θ ^^ed^^ba^^b9	Simpson § う ^^ed^^ba^^b9
1	3.539816e-02	-1.460184e-02	2.064830e-03
2	1.039816e-02	-5.190072e-03	6.006535e-06
4	2.604046e-03	-1.301966e-03	3.778277e-08
8	6.510398e-04	-3.255190e-04	5.912427e-10
16	1.627604e-04	-8.138018e-05	9.239165e-12
32	4.069010e-05	-2.034505e-05	1.442180e-13
64	1.017253e-05	-5.086263e-06	2.553513e-15
128	2.543132e-06	-1.271566e-06	-1.110223e-16
256	6.357829e-07	-3.178914e-07	3.330669e-16
512	1.589457e-07	-7.947286e-08	-3.330669e-16
1024	3.973643e-08	-1.986821e-08	1.110223e-16
2048	9.934108e-09	-4.967053e-09	5.551115e-16
4096	2.483527e-09	-1.241763e-09	5.551115e-16
8192	6.208802e-10	-3.104410e-10	-5.551115e-16
16384	1.552228e-10	-7.760781e-11	2.331468e-15



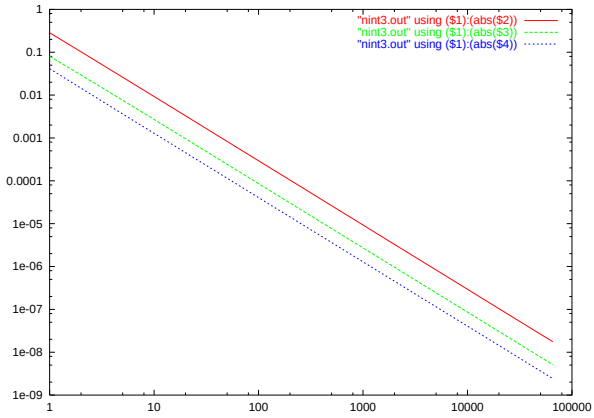
例 2.10.4 (積分区間に特異点が含まれる例) 積分区間に特異点を含む定積分

$$I = \int_0^1 \sqrt{1-x^2}\, dx$$

に対しては、複合台形則、複合中点則、複合 Simpson 則のいずれも低い精度しかでない。

誤差の表

```
#   N    □ □ 繊 □ θ ^^ed^^ba^^b9  □ □ □ 澄 □ θ ^^ed^^ba^^b9  Simpson § う
^^ed^^ba^^b9
    1   2.853982e-01    -8.062724e-02   4.138123e-02
    2   1.023855e-01    -2.944367e-02   1.449937e-02
    4   3.647090e-02    -1.058414e-02   5.100871e-03
    8   1.294338e-02    -3.773569e-03   1.798746e-03
   16   4.584904e-03    -1.339789e-03   6.351089e-04
   32   1.622558e-03    -4.746873e-04   2.243944e-04
   64   5.739352e-04    -1.680046e-04   7.930866e-05
  128   2.029653e-04    -5.942998e-05   2.803511e-05
  256   7.176766e-05    -2.101722e-05   9.911071e-06
  512   2.537522e-05    -7.431692e-06   3.503944e-06
 1024   8.971763e-06    -2.627674e-06   1.238805e-06
 2048   3.172045e-06    -9.290536e-07   4.379792e-07
 4096   1.121496e-06    -3.284755e-07   1.548482e-07
 8192   3.965100e-07    -1.161346e-07   5.474696e-08
16384   1.401877e-07    -4.105994e-08   1.935595e-08
32768   4.956389e-08    -1.451691e-08   6.843354e-09
65536   1.752349e-08    -5.132508e-09   2.419491e-09
```



例 **2.10.5** 解析的周期関数の数値積分を見てみよう。n 次の第 1 種 Bessel 関数 $J_n(x)$ は

$$J_n(x) = \frac{1}{2\pi}\int_{-\pi}^{\pi} \cos(nt - x\sin t)\,dt$$

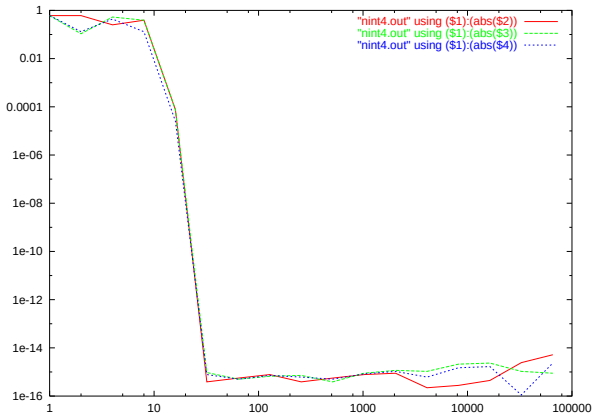
と積分表示できるが、これは解析的周期関数の一周期に渡る積分だから、台形則で非常に精密に計算できるはずである。$n = 4$, $x = 5$ のときの値、すなわち $J_4(5)$ の値を見てみよう。

nint4.c

```
/*
 * nint4.c --- 旺□□^^c5^^aa周期^^d8^^bf □o □□□摺^^ca^^ac
 *
 *  n 次あ□□□□ Bessel ^^d8^^bf □ J_n(x) あ
 *
 *            1   B
 *    J_n(x)=---□  cos (n t - x sin t) dt
 *           2B  -B
 *
 * あ□□^^ca^^ac^^c9^^bd示^^c7^^a4□^^eb^^a4^^ac、こあ^^cf^^b2□摺^^c5^^aa周
期^^d8^^bf □世□^^e9^^a1^^a2□□繊□□□□□□
 * 精^^cc^^a9^^cb^^b7^^d7^^bb擦^^c7^^a4□□^^cf^^a4困^^c7^^a4△^^eb^^a1^^a3
 *
 * UNIX う連^^d8^^b4^^d8^^bf  □^^e9^^a5^^a4^^d6^^a5  □^^ea^^a5^^a3^^cb^^a4
□ jn(int,double) ^^c8^^a4い^^cc^^be前^^c7^^b4^^d8^^bf □
 * □^^d1^^b0^^d5^^a4 気□^^c6^^a4 い□T ^^c7^^a1 △海□□□□^^d3^^a4 靴
^^c6^^a4^^df^^a4^^eb^^a1^^a3
 *
 *  コア^^d1^^a5ぅ□: gcc -o nint4 nint4.c nint.o -lm
 *    ただし nint.o あ gcc -c nint.c ^^c8^^a4靴^^c6^^bd□□□靴^^c6^^a4□□
 */

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

rrfunction f;

/* n 次あ Bessel ^^d8^^bf □□ x ^^c7^^a4□□^^cd^^a4□□□□□ */
int n;
double x;

/* □□摺^^ca^^ac^^d8^^bf □ */
double f(double t)
{
  return cos(n * t - x * sin(t));
}

int main()
{
  int m;
  double pi = 4.0 * atan(1.0), I, T, M, S;

  /* J4(5) */
  n = 4; x = 5.0;

  /* ^^e9^^a5^^a4^^d6^^a5□^^ea^^a5^^a3^^d8^^bf □ */
  I = jn(n, x);

  printf("# 旺□□^^c5^^aa 周期^^d8^^bf □ cos(%d t - %g sin t) あ [-B , B
] ^^cb^^a4□韻□摺^^ca^^ac\n",
 n, x);
  printf("# ^^ca^^a3 喉□□□§ T_m, ^^ca^^a3 喉□□□□§ M_m, ^^ca^^a3
喉 Simpson § S_{2m} う^^ed^^ba^^b9\n");
  printf("    m         I-T_m          I-M_m         I-S_{2m}\n");
  for (m = 1; m <= (1 << 16); m *= 2) {
    /* T: □□繊, M: □□□澄, S: Simpson § */
    T = trapezoidal(f, - pi, pi, m) / (2 * pi);
    M = midpoint(f, - pi, pi, m) / (2 * pi);
    S = (T + 2 * M) / 3;
    printf("  %5d\t%14e\t%14e\t%14e\n", m, I - T, I - M, I - S);
```

誤差の表

```
# 旺□□^^c5^^aa周期^^d8^^bf□ cos(4 t - 5 sin t) あ [-B , B ] ^^cb^^a4□韻□摺^^ca^^ac
# ^^ca^^a3 喉□□□§  T_m, ^^ca^^a3 喉□□□□§  M_m, ^^ca^^a3 喉 Simpson § S_{2m} う^^ed^^ba^^b9
    m          I-T_m           I-M_m          I-S_{2m}
    1  -6.087676e-01   -6.087676e-01   -6.087676e-01
    2  -6.087676e-01    1.075702e-01   -1.312091e-01
    4  -2.505987e-01   -5.321711e-01   -4.383136e-01
    8  -3.913849e-01    3.912324e-01    1.303599e-01
   16  -7.627816e-05    7.627816e-05    2.542605e-05
   32  -3.885781e-16   -9.436896e-16   -7.771561e-16
   64  -5.551115e-16   -4.996004e-16   -4.996004e-16
  128  -7.771561e-16   -6.661338e-16   -7.216450e-16
  256  -3.885781e-16   -7.216450e-16   -6.106227e-16
  512  -5.551115e-16   -3.885781e-16   -4.996004e-16
 1024  -7.771561e-16   -8.881784e-16   -8.326673e-16
 2048  -8.881784e-16   -1.165734e-15   -1.054712e-15
 4096   2.220446e-16   -1.054712e-15   -6.106227e-16
 8192  -2.775558e-16   -2.109424e-15   -1.498801e-15
16384  -4.440892e-16   -2.331468e-15   -1.665335e-15
32768  -2.442491e-15    1.054712e-15   -1.110223e-16
65536  -5.162537e-15   -8.881784e-16   -2.331468e-15
```



かなり小さな分割数 m に対して非常に高精度の値が得られている。$m = 32$ のとき、複合台形則 T_{32}, 複合中点則 M_{32} ともに 10^{-16} 程度と double 型の計算精度一杯の値が得られている (実は丸め誤差がなければ 3.7×10^{-19} 程度の値であるとか)。一方で複合 Simpson 則 S_{32} については $I - S_{32} \fallingdotseq 2.5 \times 10^{-5}$ であまり精度が出ていない。(グラフでは同じ m について T_m, M_m, S_{2m} を比較しているので同程度の精度に見えるが、本来は被積分関数の計算回数を揃えて比較すべきであろう。そうすると Simpson 則は他の二つの方法に比べて見劣りすることが分る。)

2.10.4　数直線上の解析関数の数値積分

例 2.10.6 確率積分

$$\int_{-\infty}^{\infty} e^{-x^2}\,dx = \sqrt{\pi}$$

は数直線上の解析関数だから台形則で高精度に計算できるはず。

nint5.c

```
/*
 * nint5.c --- 晦□契□^^ca^^ac
 *          、
 *    I =□   exp(-x^2)dx = □ □
 *        -、
 * あ R 匠□蔵□^^cf^^b4^^d8^^bf □ □摺^^ca^^ac だか^^e9^^a1^^a2 □ □繊
 *            、
 *    T_h = h Σ    f(n h)              (ただし f あ□ □ □ □^^ca^^ac^^d8^^bf □)
 *           n=-、
 * あ^^eb^^a4^^a4^^cf^^a1△ 修□ □^^c7^^a4 狙^^da^^a4 □
 *                 N
 *    T_{h,N} = h Σ    f(n h)
 *                n=-N
 * あ □ □ □ □ □^^cc^^a9^^cb^^b7^^d7^^bb 擦^^c7^^a4 □ □^^cf^^a4 困^^c7^^a4 △
^^eb^^a1^^a3
 *
 *  コア^^d1^^a5う□: gcc -o nint5 nint5.c -lm
 */

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

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

/* □ □摺^^ca^^ac^^d8^^bf □ */
double f(double x)
{
  return exp(- x * x);
}

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

  /* ^^df^^bc □, 晦□契□^^ca^^ac う録□ □ */
  pi = 4.0 * atan(1.0); I = sqrt(pi);

  printf(" m      h              I-T\n");
  for (m = 1; m <= (1 << 2); m *= 2) {
    h = 1.0 / m;
    /* [-6,6] あ□^^c7^^a4 狙^^da^^a4 □ */
    N = m * 6;
    T = trapezoidal2(f, h, N);
    printf("%2d\t%g\t%14e\n", m, h, I - T);
  }
  return 0;
}
```



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

誤差の表

```
m       h             I-T
1       1         -1.833539e-04
2       0.5       -2.220446e-16
4       0.25      -4.440892e-16
```

2.10.5 DE 公式

例 2.10.7 (端点の特異性くらい何でもない)

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

誤差の表

```
test1 (sqrt(1-x^2) あ □ □ ^^ca^^ac)
 h=1.000000, I_h=       1.7125198292703636, I_h-I=1.417235e-01
 h=0.500000, I_h=       1.5709101233831164, 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=       1.5707963267948970, I_h-I=4.440892e-16
 h=0.031250, I_h=       1.5707963267948954, I_h-I=-1.110223e-15
 h=0.015625, I_h=       1.5707963267948979, I_h-I=1.332268e-15
 h=0.007812, I_h=       1.5707963267948957, I_h-I=-8.881784e-16
 h=0.003906, I_h=       1.5707963267948959, I_h-I=-6.661338e-16
 h=0.001953, I_h=       1.5707963267948954, I_h-I=-1.110223e-15
test2 (1/sqrt(1-x^2) あ □ □ ^^ca^^ac)
 h=1.000000, I_h=       3.1435079763395435, 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=       3.1415926343278695, I_h-I=-1.926192e-08
 h=0.031250, I_h=       3.1415926326210664, I_h-I=-2.096873e-08
 h=0.015625, I_h=       3.1415926323669550, I_h-I=-2.122284e-08
 h=0.007812, I_h=       3.1415926327540102, I_h-I=-2.083578e-08
 h=0.003906, I_h=       3.1415926312582481, I_h-I=-2.233155e-08
 h=0.001953, I_h=       3.1415926319069580, I_h-I=-2.168284e-08
```

nint6.c

```
/*
 * nint6.c --- DE 公式
 *
 *          1      2 (1/2)
 *  I1 = □    (1-x )     dx = B /2
```

```
 *        -1
 *
 *         1      2 (-1/2)
 *  I2 = □    (1-x )      dx = B
 *        -1
 *
 *  いずあ□^^c3^^bc□世□□^^c3^^b0□性があ^^c3^^a4^^c6^^a1□□^^c5^^b5^^c5^^aa^^ca^^bf
□□□摺^^ca^^ac公式^^cf^^a4Δ^^de^^a4□^^d4^^a4□^^ca^^a4ぁ
 *  double exponential formula (DE公式) ^^ca^^a4□ΦΔ^^de^^a4□^^d7^^bb擦^^c7^^a4
□^^eb^^a1^^a3
 *
 *  コア^^d1^^a5ぅ□: gcc -o nint6 nint6.c -lm
 *
 *  晦□ (横山□□正君) う泡^^c5^^a6^^cb^^a4□^^ea^^be^^afし修正あ^^c3^^a4┐□ (2004/1/15)。

 */

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

typedef double rrfunction(double);

double debug = 0;
double pi, halfpi;

/* B  */
double phi(double t)
{
  return tanh(halfpi * sinh(t));
}

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

/* B' */
double dphi(double t)
{
  return halfpi * cosh(t) / sqr(cosh(halfpi * sinh(t)));
}

/* DE 公式^^cb^^a4□□ (-1,1) ^^cb^^a4□韻□□□摺^^ca^^acう^^d7^^bb */
double de(rrfunction f, double h, double N)
{
  int n;
  double t, S, dS;
  S = f(phi(0.0)) * dphi(0.0);
  for (n = 1; n <= N; n++) {
    t = n * h;
    dS = f(phi(t)) * dphi(t) + f(phi(-t)) * dphi(- t);
```

```
    S += dS;
    if (fabs(dS) < 1.0e-16) {
      if (debug)
        printf("\tde(): n=%d, |t|=%g, fabs(dS)=%g\n", n, t, fabs(dS));
      break;
    }
  }
  return h * S;
}

/* ^^c6^^a5好□□^^d1^^a4□□□摺^^ca^^ac^^d8^^bf□ そあ 1 */
double f1(double x)
{
  return sqrt(1 - x * x);
}

/* ^^c6^^a5好□□^^d1^^a4□□□摺^^ca^^ac^^d8^^bf□ そあ 2 */
double f2(double x)
{
  if (x >= 1.0 || x <= -1.0) return 0; else return 1 / sqrt(1 - x * x);
}

void test(rrfunction f, double exact)
{
  int m, N;
  double h, IhN;

  /* |t|、10 ^^de^^a4^^c7^^b7^^d7^^bb擦垢^^eb^^a4^^b3^^c8^^a4^^cb^^a4垢□ */
  h = 1.0; N = 10;
  /* h あ^^c8^^be^^ca^^ac, N あ□^^dc^^a4^^cb^^a4靴□ double exponential formula ^^c7^^b7^^d7^^bb
擦靴^^c6^^a4^^e6^^a4^^af */
  for (m = 1; m <= 10; m++) {
    IhN = de(f, h, N);
    printf("h=%f, I_h=%25.16f, I_h-I=%e\n", h, IhN, IhN - exact);
    h /= 2; N *= 2;
  }
}

int main(int argc, char **argv)
{
  if (argc >= 2 && strcmp(argv[1], "-d") == 0)
      debug = 1;
  pi = 4.0 * atan(1.0); halfpi = pi / 2;

  printf("test1 (sqrt(1-x^2) あ□□^^ca^^ac)\n");
  test(f1, halfpi);

  printf("test2 (1/sqrt(1-x^2) あ□□^^ca^^ac)\n");
  test(f2, pi);
```

```
  return 0;
}
```

第3章　山本第7章「数値積分」から抜き書き

3.1　数値積分公式

積分

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

を求めるための近似公式[1]は、通常次の形である。

$$I_n(f) = \alpha_1 f(x_1) + \cdots + \alpha_n f(x_n) \quad (n \text{ 点公式}). \tag{3.1}$$

ここで α_j は定数, x_j は区間 $[a, b]$ における相異なる分点である。この公式による誤差を $E_n(f) \overset{\text{def.}}{=} I(f) - I_n(f)$ で表わし、条件

$$E_n(x^k) = 0 \quad (k = 0, 1, 2, \cdots, m)$$

が成り立つとき、公式 (3.1) は少なくとも m 次の精度を持つという。また

$$E_n(x^k) = 0 \quad (k = 0, 1, 2, \cdots, m), \quad E_n(x^{m+1}) \neq 0$$

が成り立つとき、公式 (3.1) は (ちょうど) m 次の精度を持つ、(ちょうど) m 次の積分公式であるという。

明らかに、ちょうど m 次の積分公式を用いるとき、高々 m 次の多項式 $f(x)$ について $E_n(f) = 0$, また $m + 1$ 次の任意の多項式 $g(x)$ に対して、$E_n(g) \neq 0$ である。

[1] ある程度一般の f が与えられるものであると仮定している。

n 点近似公式の作り方

$[a,b]$ 上に n 個の分点 $a \le x_1 < x_2 < \cdots < x_n \le b$ を取り、それらに関する $n-1$ 次補間多項式を $p_{n-1}(x)$ とする。Lagrange の公式を用いて表わせば、

$$p_{n-1}(x) = \sum_{j=1}^{n} \ell_j^*(x) f(x_j), \quad \ell_j^*(x) = \prod_{i=1, i\neq j}^{n} \left(\frac{x - x_i}{x_j - x_i} \right).$$

多項式であることから、$I(p_{n-1})$ は簡単に表現できる。

$$I(p_{n-1}) = \int_a^b p_{n-1}(x)\, dx = \sum_{j=1}^{n} \alpha_j f(x_j), \quad \alpha_j = \int_a^b \ell_j^*(x)\, dx.$$

そこで、$I(f)$ の近似として $I(p_{n-1})$ を採用してみると、これは確かに (3.1) の形をしていて、

$$\begin{aligned} F_n(f) &= I(f) - I_n(f) = \int_a^b \left(f(x) - p_{n-1}(x) \right)\, dx \\ &= \int_a^b (x - x_1) \cdots (x - x_n) f[x_1, \cdots, x_n, x] dx. \end{aligned}$$

さらに f が $[a,b]$ において C^n-級ならば、

$$\exists \xi = \xi(x) \in (a,b) \quad \text{s.t.} \quad E_n(f) = \frac{1}{n!} \int_a^b (x - x_1) \cdots (x - x_n) f^{(n)}(\xi(x))\, dx.$$

特に $f(x) = x^k$ $(0 \le k \le n-1)$ ならば $f^{(n)}(x) \equiv 0$ であるから、$E_n(f) = 0$. つまり、これは少なくとも $n-1$ 次の公式である。

第4章　TO DO LIST

- 直交多項式の理論
- Gauss 型公式
- 高橋・森の誤差解析理論
- 杉原の DE 公式の最適性定理
- 多次元数値積分

関連図書

[1] 杉原 正顯(まさあき), 室田(むろた) 一雄, 数値計算法の数理, 岩波書店 (1994).

[2] 森 正武(まさたけ), 数値解析, 共立出版 (1973).

[3] 森 正武, FORTRAN 77 数値計算プログラミング, 岩波書店 (1986).

[4] 森 正武, 名取 亮(まこと), 鳥居 達生, 数値計算 第 4 章 (森正武執筆), 岩波講座 情報科学, 岩波書店 (1982).

[5] P.J.Davis and P.Rabinowitz, Methods of Numerical Integration, Academic Press, 1975.
森 正武 訳, 計算機による数値積分法, 日本コンピュータ協会 (1981).

[6] 山本哲朗, 数値解析入門, サイエンス社 (1976).

[7] 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).