# 関数の補間と数値積分

# 目次


1 はじめに 2

2 課題 4.2 2
2.1 4.2.1 . . . 2
2.1.1 Lagrange 補間 . . . 2
2.1.2 Lagrange 補間の適用 . . . 3
2.1.3 結果の考察 . . . 4
2.2 4.2.2 . . . 5
2.2.1 Newton-Cotes 公式 . . . 5
2.2.2 計算結果 . . . 6
2.3 4.2.3 . . . 7
2.3.1 計算結果 . . . 7
2.3.2 結果の考察 . . . 8

3 課題 4.3 9
3.1 4.3.1 . . . 9
3.1.1 Gauss 型積分公式 . . . 9
3.1.2 計算結果 . . . 11
3.1.3 結果の考察 . . . 12
3.2 4.3.2 . . . 13
3.2.1 計算結果 . . . 13
3.2.2 結果の考察 . . . 15

4 課題 4.4 16
4.1 有限要素法 . . . 16
4.1.1 弱形式 . . . 17
4.1.2 弱形式の離散化 . . . 17
4.2 4.4.1 . . . 18
4.3 4.4.2 . . . 18
4.4 4.4.3 . . . 19
4.5 4.4.4 . . . 19
4.6 4.4.5 . . . 20
4.6.1 計算結果 . . . 20
4.6.2 結果の考察 . . . 21

5 付録 22

6 参考文献 38

# 1　はじめに

本レポートでは、関数補間による数値積分について述べる。関数補間に基づく手法は、高々 3,4 次元での数値積分で広く用いられている。本レポートでは特に、一次元上の区間 $(a, b)$ における積分 $\int_a^b f(x)dx$ について扱う。なお、実験で用いたプログラムは付録に記す。

# 2　課題 4.2

Lagrange 補間を使用してみる。また、Lagrange 補間を利用した積分公式である Newton-Cotes 公式により、関数の定積分を計算する。なお、ここでは区間 $[a, b]$ における $n+1$ 個の分点を

$$x_i = a + \frac{b-a}{n} i \quad (i = 0, 1, \cdots, n) \tag{1}$$

と定める。

## 2.1　4.2.1

以下に示す 3 つの関数について、分点 $x_i$ における値を求める。その後、$n = 2, 4, 8, 16$ として Lagrange 補間で $n$ 次多項式 $P(x)$ を求める。

$$(a)\ f_1(x) = \ln(x) \quad x \in [1, 2]$$
$$(b)\ f_2(x) = 1/x^3 \quad x \in [0.1, 2]$$
$$(c)\ f_3(x) = 1/(1 + 25x^2) \quad x \in [-1, 1]$$

### 2.1.1　Lagrange 補間

相違なる $n$ 個の点 $x_1 < \cdots < x_n$ のそれぞれに関数 $f$ の値 $f(x_i)$ が与えられたとき、

$$P(x_i) = f(x_i) \ \ (i = 1, \cdots, n) \tag{2}$$

を満たす $n-1$ 次多項式 $P(x)$ を作り、$x \in [x_1, x_n]$ における $f(x)$ の値を $P(x)$ を推定することを Lagrange 補間 (多項式補間) と呼ぶ。このような $n-1$ 次の補間多項式 $P(x)$ は一意に存在する。実際、$P(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1}$ とおくと、与えられた条件は

$$a_0 + a_1 x_i + \cdots + a_{n-1} x_i^{n-1} = f(x_i) \ \ (i = 1, \cdots, n) \tag{3}$$

となり、この式をまとめて書くと

$$\begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ & \cdots & & \cdots & \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \cdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} f(x_1) \\ f(x_2) \\ \cdots \\ f(x_n) \end{pmatrix} \tag{4}$$

となる。係数行列の行列式は、Vandermonde の行列式であるから

$$\begin{Vmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ & \cdots & & \cdots & \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{Vmatrix} = \prod_{i>j} (x_i - x_j) \neq 0$$

となるため、式 (4) の解は一意に存在する。
また、式 (4) を満たす $n-1$ 次の補間多項式 $P(x)$ は

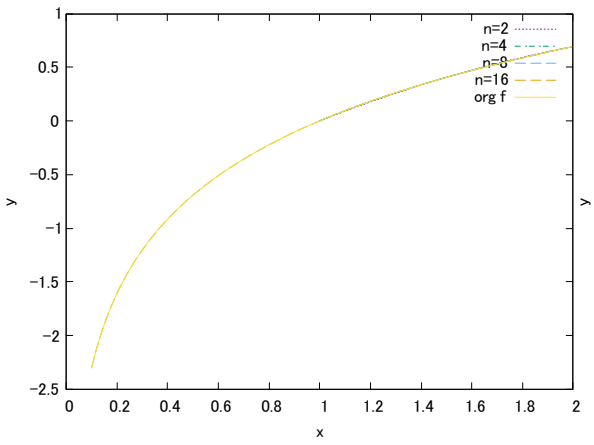
$$P(x) = \sum_{i=1}^{n} f(x_i) l_i(x) \tag{5}$$

$$l_i(x) = \frac{\prod_{j \neq i}(x - x_j)}{\prod_{j \neq i}(x_i - x_j)} \tag{6}$$

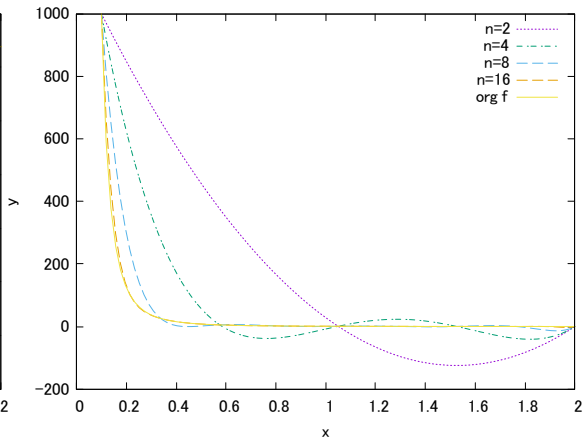
で与えられる。実際、$l_i(x_j) = \delta_{ij}$ より $P(x_i) = f(x_i)$ となる。また、$P(x)$ は $n-1$ 次多項式 $l_i(x)$ の線形結合であるので、高々 $n-1$ 次である。よってこの $P(x)$ は式 (4) を満たし、補間多項式の一意性からこの $P(x)$ は $f(x)$ の $n-1$ 次補間多項式である。

### 2.1.2 Lagrange 補間の適用

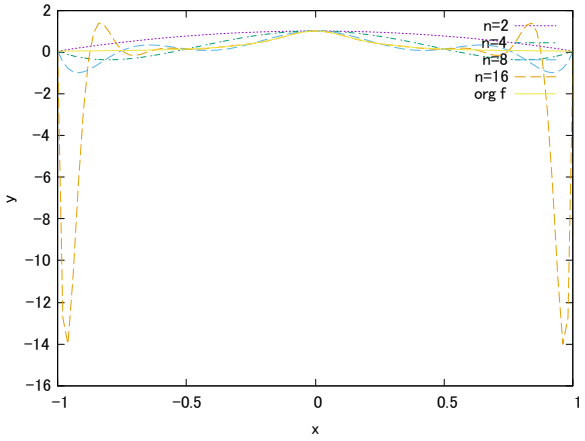
先に示した式 $(a), (b), (c)$ について、分点 $x_i$ における値を求める。その後、$n = 2, 4, 8, 16$ として Lagrange 補間で $n$ 次多項式 $P(x)$ を求める。
結果を表 1 に示す。関数ごとに、各 $n$ でのグラフをまとめている。




**(a)** $f_1$




**(b)** $f_2$




**(c)** $f_3$

**図 1:** P(x) の値

### 2.1.3 結果の考察

図 1 を見ると、$f_1$ ではどの $n$ についても元の関数とほぼ一致しており、$f_2$ では $n$ が大きくなるほど元の関数に近づいている様子が見て取れる。しかし、$f_3$ では、$x = 0$ 近傍では $n$ の増加に従って精度が良くなっているものの、$x$ の絶対値が大きくなるにつれて誤差が大きくなり、発散のような挙動が見られる。以下では、この原因を考察する。

まず、Lagrange 補間と Taylor 展開の関係について述べる。Taylor 展開は無限次の多項式による $f$ の表現である。したがって、Lagrange 補間で得られる補間多項式は、$n \to \infty$ では Taylor 展開により得られる多項式に近づいていく。

つまり、Lagrange 補間による補間多項式は Taylor 展開の近似と見なすことができる。そのため、Taylor 展開における収束半径の内側では補間は上手く機能するものの、収束半径の外側では補間多項式の値が発散することが予測できる。実際、$f_3$ を考えると、$x = 0$ における Taylor 展開の収束半径は $|x| < 0.2$ であり、図 2 から、この内側では補間が上手く機能していることがわかる。

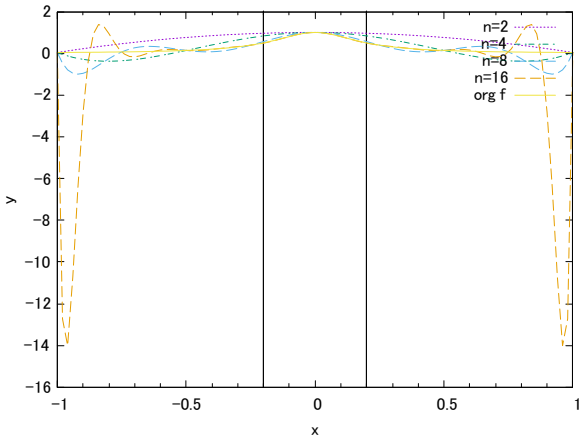



**図 2:** 収束半径を図示した様子

## 2.2 4.2.2

以下に示す 4 つの関数について、区間内での定積分の理論値、および分割数 $n = 2^i \quad (i = 0, 1, \cdots, 10)$ の (複合) 中点公式、台形公式、Simpson 公式による値を求める。

$$
\begin{aligned}
&(a)\ f_4(x) = 1/x \quad x \in [1, 2] \\
&(b)\ f_5(x) = e^{5x} \quad x \in [-1, 1] \\
&(c)\ f_6(x) = 1 + \sin(x) \quad x \in [0, \pi] \\
&(c)\ f_7(x) = 1 + \sin(x) \quad x \in [0, 2\pi]
\end{aligned}
$$

### 2.2.1 Newton-Cotes 公式

定積分

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

を近似的に計算するにあたり、Lagrange 補間で関数 $f$ を近似する。$x_i$ を $n$ 個の等間隔分点

$$x_i = x_1 + (i-1)h \quad (1 \leq i \leq n, h > 0)$$

とし、この分点における関数 $f(x)$ の Lagrange 補間を用いると、

$$
\begin{aligned}
I(f) &= \int_a^b f(x)dx \\
&\simeq \int_a^b P(x)dx \\
&= \int_a^b \sum_{i=1}^{n} f(x_i) l_i(x) \\
&= \sum_{i=0}^{n} c_i f(x_i)
\end{aligned}
$$

となる。ここで

$$c_i = \int_a^b l_i(x)dx$$

である。この数値積分公式を n 点 Newton-Cotes 公式という。この公式で、

$$n = 1, \quad x_1 = \frac{a+b}{2}$$

としたものが中点公式、

$$n = 2, \quad x_1 = a, \quad h = b - a$$

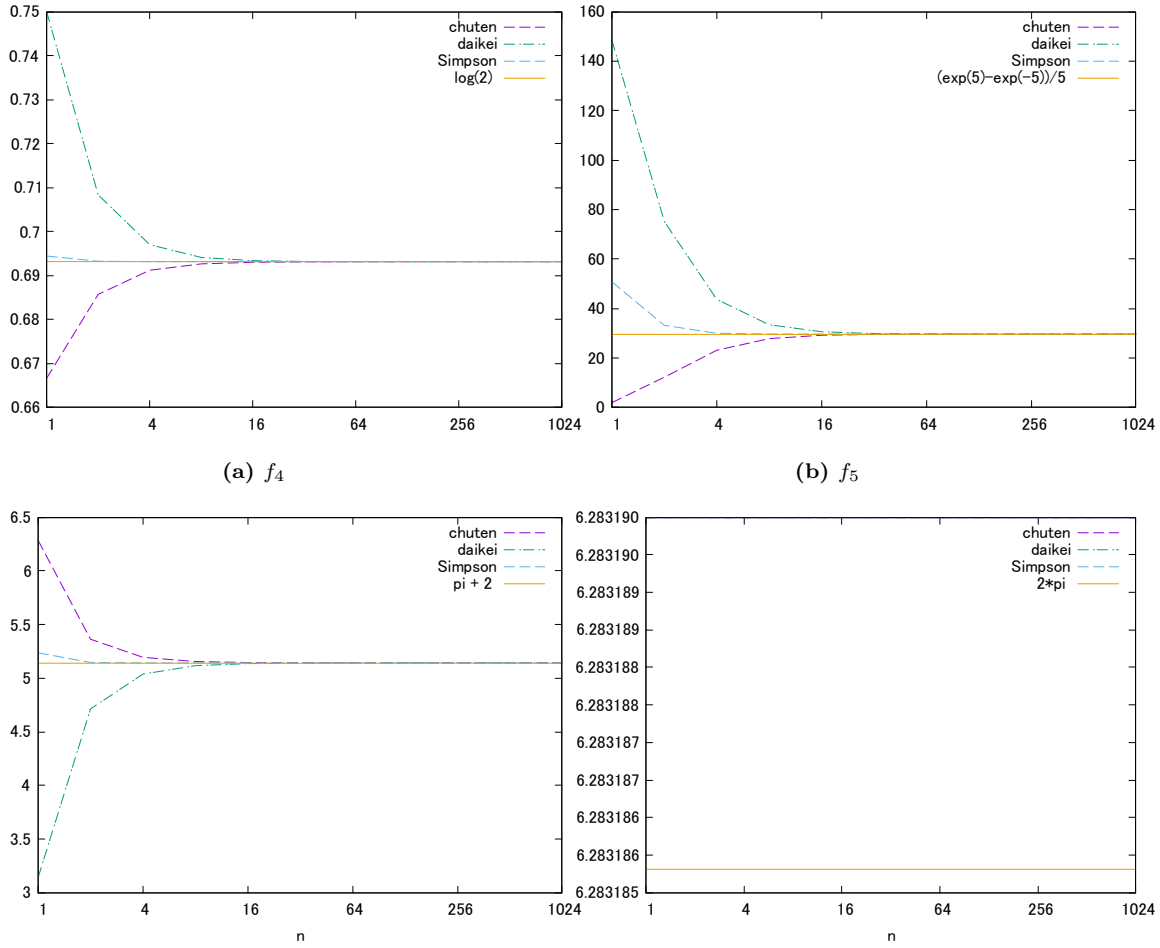
としたものが台形公式、

$$n = 3, \quad x_1 = a, \quad h = \frac{b-a}{2}$$

としたものが Simpson の公式である。実際の計算では、積分区間を等分して小区間を作り、各区間に対して上記のような数値積分公式を適用する。このようにして得られる積分公式を複合公式と言い、小区間で用いる積分公式の頭に「複合」を付けて「複合中点公式」などと呼ぶ。「複合」を略して、単に中点公式などと呼ぶことも多い。

### 2.2.2 計算結果

計算結果を図 3 に示す。関数ごとに、各積分公式と理論値をまとめている。




**(a)** $f_4$ **(b)** $f_5$

**(c)** $f_6$ **(d)** $f_7$

**図 3:** 定積分の値。pi は $\pi$ を表す。

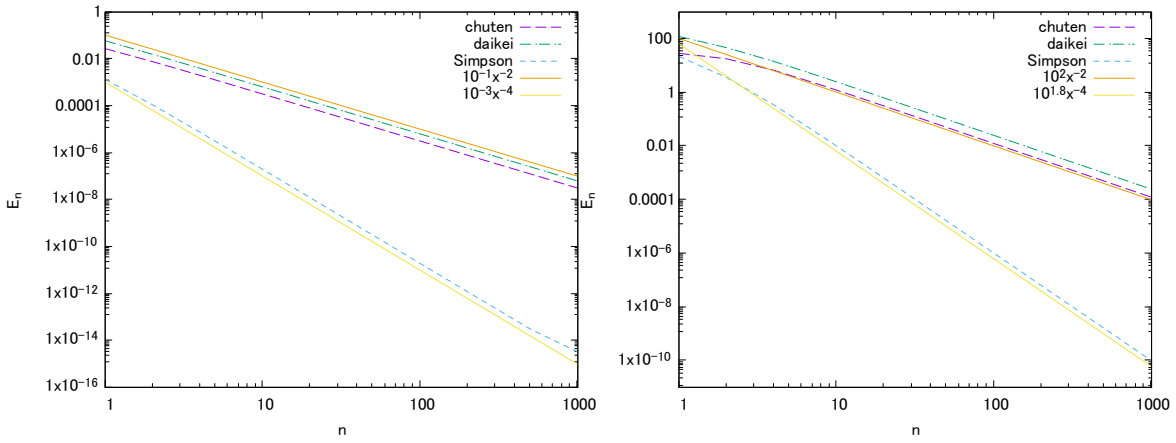
図 3 から、どの関数についても、全ての積分公式で収束することがわかった。ここで、$f_7$ は他と異なり、いかなる $n$ についても理論値と一致しているが、その理由の考察は次の問題で述べる。

## 2.3 4.2.3

4.2.2 において、関数 $f$ に対する、分割数 $n$ での数値計算結果と理論値との誤差 (絶対値) を $E_n(f)$ とする。関数 $f_i$ に対して、積分公式ごとの誤差の $n$ 依存性を調べる。

### 2.3.1 計算結果

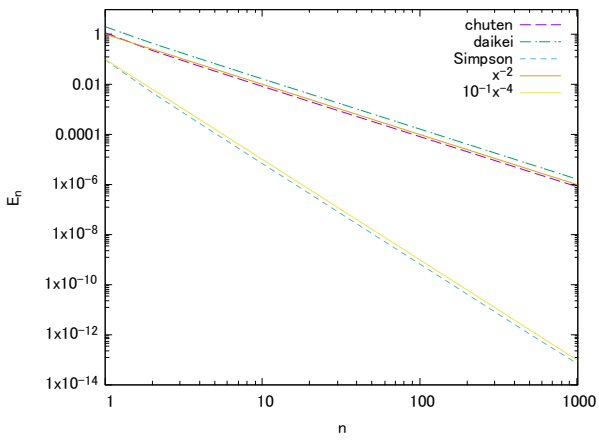
誤差をまとめて図 4 に示す。



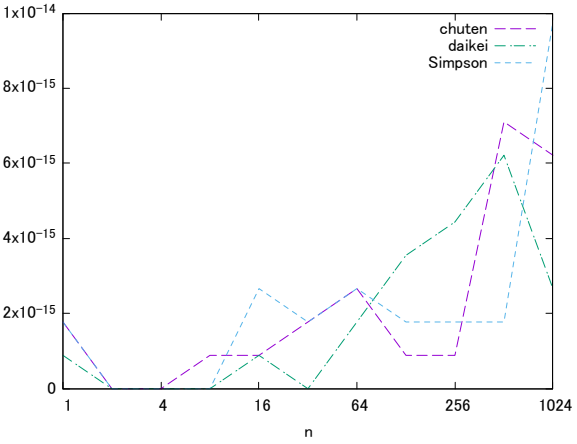

**(a)** $f_4$




**(b)** $f_5$




**(c)** $f_6$




**(d)** $f_7$

**図 4:** 関数ごとの誤差。

図 4 から、$f_4, f_5, f_6$ については、積分公式ごとに誤差の $n$ 依存性が同一であることがわかる。すべての公式が $n$ のべきに依存し、その指数は中点公式と台形公式で-2,Simpson 公式で-4 である。

### 2.3.2 結果の考察

図 4 において、$f_7$ のみ関数形が異なっている。前問の図 3 を見ても、$f_7$ の特殊性が見て取れる。以下では、$f_7$ の特殊性の原因を考察する。

まず、複合公式でない中点公式の誤差を考える。中点公式

$$\int_a^b f(x)dx \simeq (b-a)f(x_1)$$

の左辺において $f(x)$ を $x = x_1 := (a+b)/2$ まわりで Taylor 展開すると、

$$\begin{aligned}\int_a^b f(x)dx &\simeq \int_a^b (f(x_1) + f'(x_1)(x-x_1) + \frac{1}{2}f''(x_1)(x-x_1)^2)dx \\ &= (b-a)f(x_1) + \frac{(b-a)^3}{24}f''(x_1)\end{aligned}$$

となる。したがって中点公式の誤差は $\frac{(b-a)^3}{24}f''(x_1)$ となる。これを用いると n 点複合中点公式の誤差は

$$\int_a^b f(x)dx - h\sum_{i=0}^{n-1} f\left(\frac{x_i + x_{i+1}}{2}\right) \tag{7}$$

$$=\sum_{i=0}^{n-1}\left\{\int_{x_i}^{x_{i+1}} f(x)dx - hf\left(\frac{x_i + x_{i+1}}{2}\right)\right\} \tag{8}$$

$$=\sum_{i=0}^{n-1}\frac{(x_{i+1}-x_i)^3}{24}f''\left(\frac{x_i + x_{i+1}}{2}\right) \tag{9}$$

$$=\frac{n}{24}f''(\eta)h^3 = \frac{(b-a)}{24}f''(\eta)h^2 \tag{10}$$

となる。ただし、$\eta$ は次式を満たすように選んだ。

$$\sum_{i=0}^{n-1} f''(\frac{x_i + x_{i+1}}{2}) = nf''(\eta)$$

同様に、他の積分公式についても、誤差は $\sum_{i=0}^{n-1} f''(\frac{x_i+x_{i+1}}{2})$ に比例する。
ここで、$f_7$ では $f''(x) = -\sin(x)$ であり、対称性と積分範囲から $\sum_{i=0}^{n-1} f''(\frac{x_i+x_{i+1}}{2}) = 0$ となる。したがって誤差は理論的には 0 となり、これが $f_7$ の特殊性の原因である。また、図 4 で $f_7$ の誤差が 0 でないのは、浮動小数点演算における丸め誤差など、計算機の性質によって誤差が生じるためであると思われる。

# 3 課題 4.3

課題 4.2.2 における 4 つの関数 $f_i$ $(i = 4, 5, 6, 7)$ の定積分を考える。

## 3.1 4.3.1

分割数 $n = 2^i$ $(i = 0, 1, \cdots, 10)$ の複合 $M$ 次 Gauss 型積分公式 $(M = 2, 3)$ による数値積分値を $I_n(f_i)$ とし、真値との誤差を $E_n(f_i) = |I_n(f_i) - I(f_i)|$ と定義する。$E_n(f_i)$ の $n$ 依存性と、その指数の $M$ 依存性を調べる。

### 3.1.1 Gauss 型積分公式

Newton-Cotes 公式は、$n$ 個の等間隔分点を用いて、少なくとも $n-1$ 次までの多項式に対して誤差なく積分を行うことができた。Gauss 型積分公式は、分点の取り方を適切に決めることで、より高次の多項式について誤差を無くすことを実現した公式である。以下でその公式を構成する。
まず、分点の定め方を考える。$n$ 個の重み $w_i$ と分点 $x_i$ に対して積分公式

$$\int_{-1}^{1} f(x)dx = \sum_{i=1}^{n} w_i f(x_i) \tag{11}$$

が $m$ 次までの多項式に対して正確であるとする (簡単のため積分範囲は $(-1, 1)$ とする.)
このとき分点 $x_i$ を零点とする $n$ 次多項式を $G(x) = \prod_{i=1}^{n}(x - x_i)$, $r$ 次の Legendre 多項式を $P_r$ とすると, $0 \le r \le m - n$ に対して

$$\int_{-1}^{1} G(x)P_r(x)dx = \sum_{i=1}^{n} w_i G(x_i)P_r(x_i) = 0 \tag{12}$$

が成り立つ ($G(x)P_r(x)$ は高々 $m$ 次多項式). つまり $G(x)$ と $P_r(x)(0 \leq r \leq m-n)$ は直交する. これと Legendre 多項式の直交性より、$G(x)$ は $m-n+1$ 次以上の多項式でなければならない (もし $G(x)$ が $m-n$ 次以下の多項式ならば, $G(x)$ は $P_r(x)(0 \leq r \leq m-n)$ の線型結合で表されるため, いずれかの $P_r(x)$ と直交しない.) すなわち

$$n \geq m-n+1 \tag{13}$$

より

$$m \leq 2n-1 \tag{14}$$

積分公式 (4.11) は $m$ 次までの多項式に対して正確であるとしたので, $m$ を最大化するために $m = 2n-1$ とする. このとき $n$ 次多項式 $G(x)$ に対して式 (12) は, $0 \leq r \leq n-1$ で成り立たなければならない. すなわち $G(x)$ は $n$ 次の Legendre 多項式の定数倍である必要がある. さらにこの事実と式 (12) より $x_i$ は $n$ 次 Legendre 多項式の零点となることがわかる. したがって、分点 $x_i$ は $n$ 次 Legendre 多項式の零点とすればよい。
次に、$m = 2n-1$ を実現する重み $w_i$ を求める。$f(x)$ を $2n-1$ 次以下の多項式とするとき、$f(x)$ を $P_n(x)$ で割った商を $Q(x)$, 余りを $R(x)$ とする。すなわち

$$f(x) = P_n(x)Q(x) + R(x).$$

ただし $Q(x), R(x)$ は $n-1$ 次以下の多項式である。$P_n(x)$ は $n-1$ 次以下の任意の多項式と直交するので、

$$\int_{-1}^{1} f(x)dx = \int_{-1}^{1} (P_n(x)Q(x) + R(x))dx \tag{15}$$

$$= \int_{-1}^{1} R(x)dx \tag{16}$$

となる。一方、分点 $x_i$ は Legendre 多項式の零点であったから

$$\sum_{i=1}^{n} w_i f(x_i) = \sum_{i=1}^{n} (P_n(x_i)Q(x_i) + R(x_i)) \tag{17}$$

$$= \sum_{i=1}^{n} w_i R(x_i) \tag{18}$$

となる。以上より、数値積分公式が $n-1$ 次以下の多項式 $R(x)$ に対して正確な値を与えるように重み $w_i$ を定めれば、$2n-1$ 次以下の多項式 $f(x)$ に対しても正確な値を与えることがわかる。$n-1$ 次多項式 $R(x)$ の Lagrange 補間は $R(x)$ に等しいから、$R(x) = \sum_{i=1}^{n} R(x_i)l_i(x)$ である。これを積分すると

$$\int_{-1}^{1} R(x)dx = \sum_{i=1}^{n} R(x_i) \int_{-1}^{1} l_i(x)dx \tag{19}$$

となるので、重みは

$$w_i = \int_{-1}^{1} l_i(x)dx \tag{20}$$

で与えられる。$n$ 次の Legendre 多項式の零点 $x_i$ と重み (20) による積分公式 (11) を $n$ 次の Gauss 型積分公式と呼ぶ。

また、Gauss 型積分公式も、Newton-Cotes 公式と同様に、積分区間を小区間に分割し、それぞれの小区間に低次の積分公式を用いる複合公式が一般的である。積分区間 $(a,b)$ を分点 $x_i = a + i(b-a)$ $(i = 0, \cdots, N)$ で $N$ 等分すると

$$\int_a^b f(x)dx = \sum_{i=0}^{N-1} \int_{x_i}^{x_{i+1}} f(x)dx$$

区間 $(x_i, x_{i+1})$ の積分は変数変換

$$y = 2\frac{x - x_i}{x_{i+1} - x_i} - 1$$

で $(-1,1)$ での積分に変換できるので、$n$ 次 Gauss 型積分公式の積分点と重みをそれぞれ $y_m, w_m$ $(m = 1, \cdots, n)$ とすると、

$$\begin{aligned}
&\sum_{i=0}^{N-1} \int_{x_i}^{x_{i+1}} f(x)dx \\
&\qquad = \sum_{i=0}^{N-1} \int_{-1}^{1} f\left\{\frac{(y+1)(x_{i+1}-x_i)}{2} + x_i\right\} \frac{x_{i+1}-x_i}{2} dy \\
&\qquad = \sum_{i=0}^{N-1} \sum_{m=1}^{n} w_m f\left\{\frac{(y_m+1)(x_{i+1}-x_i)}{2} + x_i\right\} \frac{x_{i+1}-x_i}{2}
\end{aligned}$$

となり、Gauss 型積分公式に対する複合公式を得る。

### 3.1.2 計算結果

計算結果を図 5 に示す。なお、$y, w$ の値は表 1 に示す通りである。

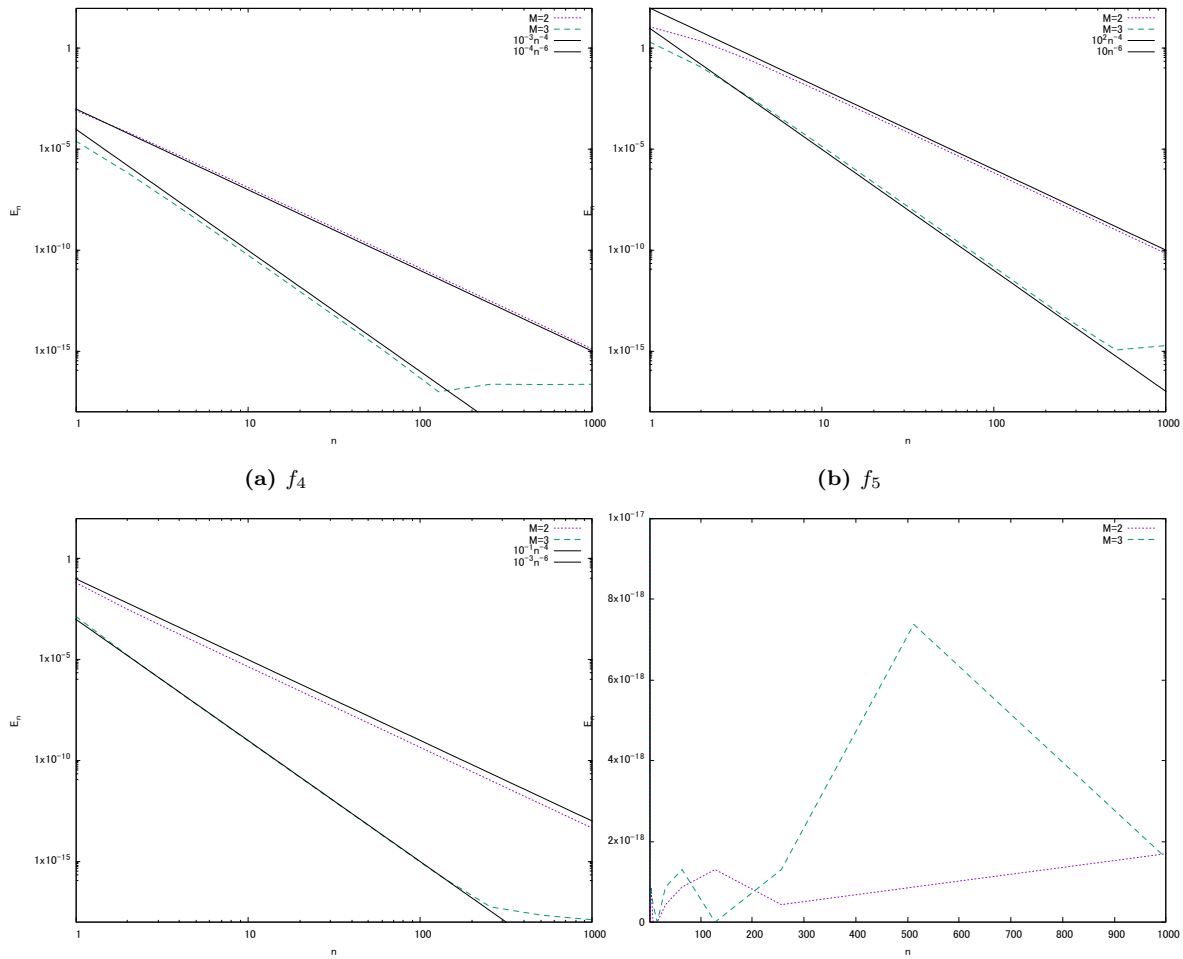
**表 1:** $M$ の値ごとの $y_m, w_m$ の値.(a)$M = 2$,(b)$M = 3$

**(a)**

| $m$ | 1 | 2 |
|---|---|---|
| $y_m$ | $\sqrt{-1/3}$ | $\sqrt{1/3}$ |
| $w_m$ | 1 | 1 |

**(b)**

| $m$ | 1 | 2 | 3 |
|---|---|---|---|
| $ym$ | $-\sqrt{3/5}$ | 0 | $\sqrt{3/5}$ |
| $wm$ | 5/9 | 8/9 | 5/9 |




**(a)** $f_4$




**(b)** $f_5$




**(c)** $f_6$

**(d)** $f_7$

**図 5:** 関数ごとの誤差。

図 5 より、$f_4, f_5, f_6$ については、積分公式ごとに誤差の $n$ 依存性が同一であることがわかる。Gauss 型積分公式は $n$ のべきに依存し、その指数は $M = 2$ で-4,$M = 3$ で-6 である。$f_7$ については、4.2.3 と同様に計算機の性質による誤差が発生しているものと思われる。

### 3.1.3 結果の考察

Gauss 型積分公式の誤差について考える。区間 $(-1, 1)$ における積分の、Gauss 型積分公式の誤差は、ある $\xi \in (a, b)$ に対して

$$\int_{-1}^{1} f(x)dx - \sum_{m=1}^{n} w_m f(y_m) \simeq \frac{f^{(2n)}(\xi)}{(2n)!} \int_{-1}^{1} \phi_n(x)^2 dx$$

$$\phi_n(x) = (x - y_1)^2 \cdots (x - y_m)^2$$

となることが知られている。これを用いると、複合 $n$ 次 Gauss 型積分公式の誤差は

$$\begin{aligned}
&\int_a^b f(x)dx - \sum_{i=0}^{N-1}\sum_{m=1}^{n} w_m f(x(y_m))\frac{x_{i+1}-x_i}{2} \\
&= \sum_{i=0}^{N-1}\int_{x_i}^{x_{i+1}} f(x)dx - \sum_{i=0}^{N-1}\sum_{m=1}^{n} w_m f(x(y_m))\frac{x_{i+1}-x_i}{2} \\
&= \sum_{i=0}^{N-1}\left\{\int_{-1}^{1} f(x(y))dy - \sum_{m=1}^{n} w_m f(x(y_m))\right\}\frac{x_{i+1}-x_i}{2} \\
&\simeq \sum_{i=0}^{N-1}\frac{f^{(2n)}(\xi_i)}{(2n)!}\int_{-1}^{1}\phi_n(x)^2dx\left(\frac{x_{i+1-x_i}}{2}\right)^{2n+1} \\
&= N\frac{f^{(2n)}(\eta)}{(2n)!}\int_{-1}^{1}\phi_n(x)^2dx\left(\frac{x_{i+1-x_i}}{2}\right)^{2n+1}
\end{aligned}$$

となる。したがって複合公式の刻み幅を $h=(b-a)/N=x_{i+1}-x_i$ とすると、この式は

$$\frac{(b-a)f^{(2n)}(\eta)}{(2n)!2^{2n+1}}\int_{-1}^{1}\phi_n(x)^2dx\cdot h^{2n}$$

となり、誤差は分割数 $N$ の $-2n$ 乗に比例することが理論的に裏付けられた。

## 3.2 4.3.2

以下の定積分を考える。

$$A=\int_0^1\frac{e^{-x}}{\sqrt{x}}dx\simeq 1.49364826562$$

以下では積分区間を $n=2^i\ \ (i=0,\cdots,10)$ 等分して、$M$ 次 Gauss 型積分公式を用いる。得られた数値積分値を $A_n$ とし、真値との誤差を $E_n=A_n-A$ とする。$M=2,3$ に対して、以下に示す 3 つの方法で $E_n$ を求める。また、それぞれの方法で中点公式や台形公式、Simpson 公式が適用できるかについても考察する。さらに、Gauss 型積分公式と Newton-Cotes 公式との精度の違いについても考察する。

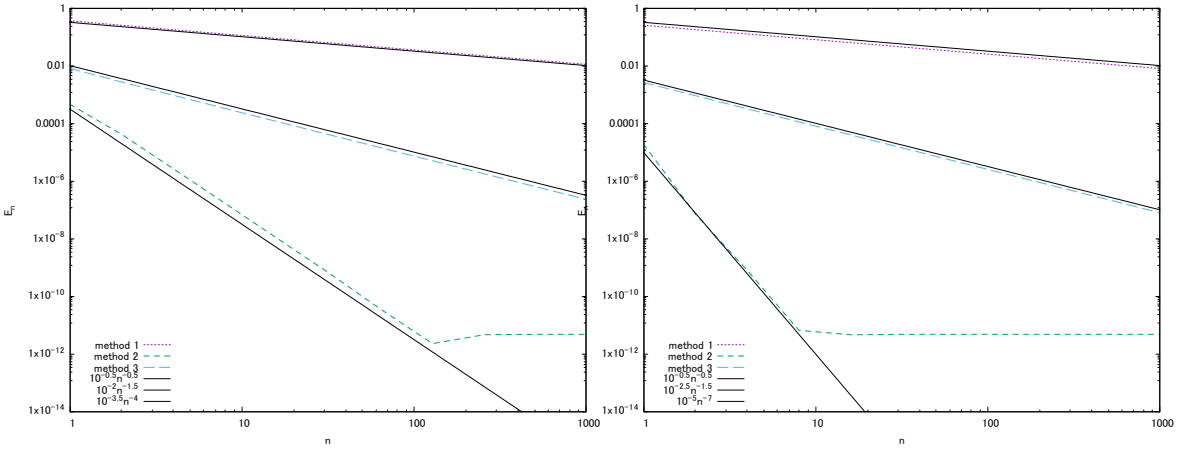
(方法 1) 積分表式 $\int_0^1\frac{e^{-x}}{\sqrt{x}}$ を用いる。
(方法 2) 積分表式 $\int_0^1\frac{e^{-x}}{\sqrt{x}}$ から $t=\sqrt{x}$ と変数変換した積分表式 $\int_0^1 2e^{-t^2}dt$ を用いる。
(方法 3) 定積分 A を

$$A=\int_0^1\frac{1}{\sqrt{x}}dx+\int_0^1\frac{e^{-x}-1}{\sqrt{x}}dx$$

と変形し、右辺第 1 項の積分は理論的に求め、第 2 項を数値積分で求める。

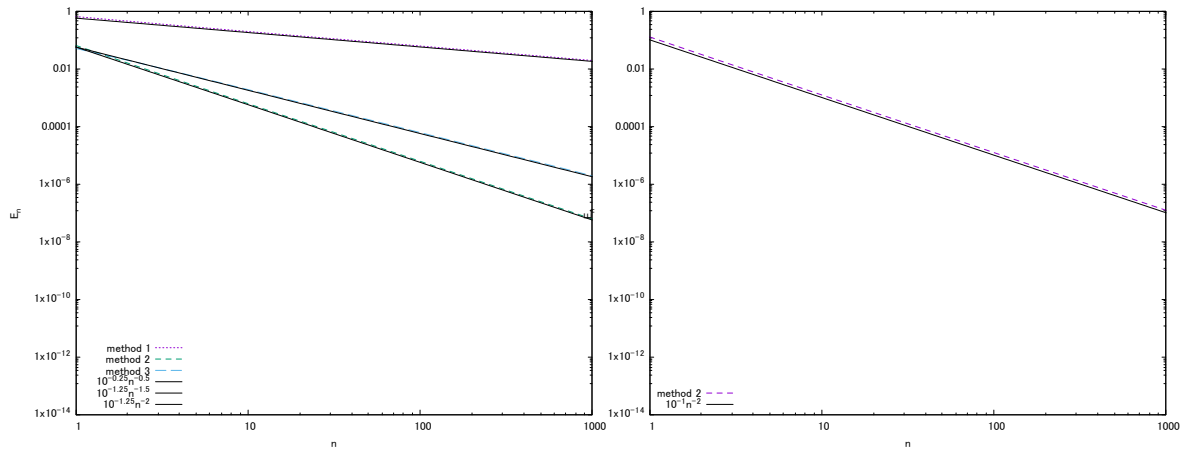
### 3.2.1 計算結果

Gauss 型積分公式の計算結果を図 6 に、中点公式、台形公式と Simpson 公式の適用結果を図 7 に示す。

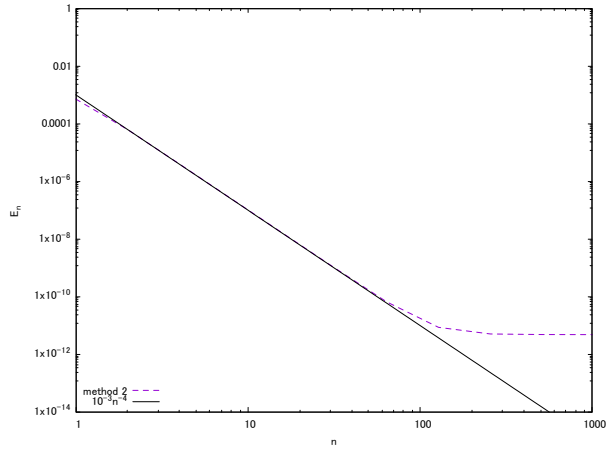



**(a)** $M = 2$ **(b)** $M = 3$

**図 6:** 方法ごとの誤差。




**(a)** 中点公式　　**(b)** 台形公式




**(c)** Simpson 公式

**図 7:** 他の積分公式での誤差。

### 3.2.2　結果の考察

**■Gauss 型積分公式**　まず、図 6 で、方法 1,3 に比べて方法 2 で収束が速くなっている理由を考える。これは、方法 1,3 では積分区間に特異点があるのに対して方法 2 では変数変換により特異点が除かれているためであると考えられる。実際、方法 2 では理論的に予測される精度に近くなっている。一方、方法 1 に比べて方法 3 で収束が速くなっている理由は、$x \to 0$ で分子も 0 に収束するために特異点付近での発散が抑えられているためであると考えられる。

**■他の積分公式**　台形公式および Simpson 公式では、方法 1,3 で結果が nan や inf になった。これは、4.2.2 で示したように、台形公式と Simpson 公式では $x_0 = a$ での $f$ の値を用いるが、ここで 0 除算が発生するためである。このように端点で被積分関数が発散する場合には、台形公式や Simpson 公式は適用できない。

**■Gauss 型積分公式と Newton-Cotes 公式の比較**　図 8 に、方法 2 での各公式の誤差を示した。これを見ると、どの方法でも誤差は $1.0 \times 10^{-12}$ より小さくなっていない。一方、収束の速度は $M = 3$ の Gauss 型積分

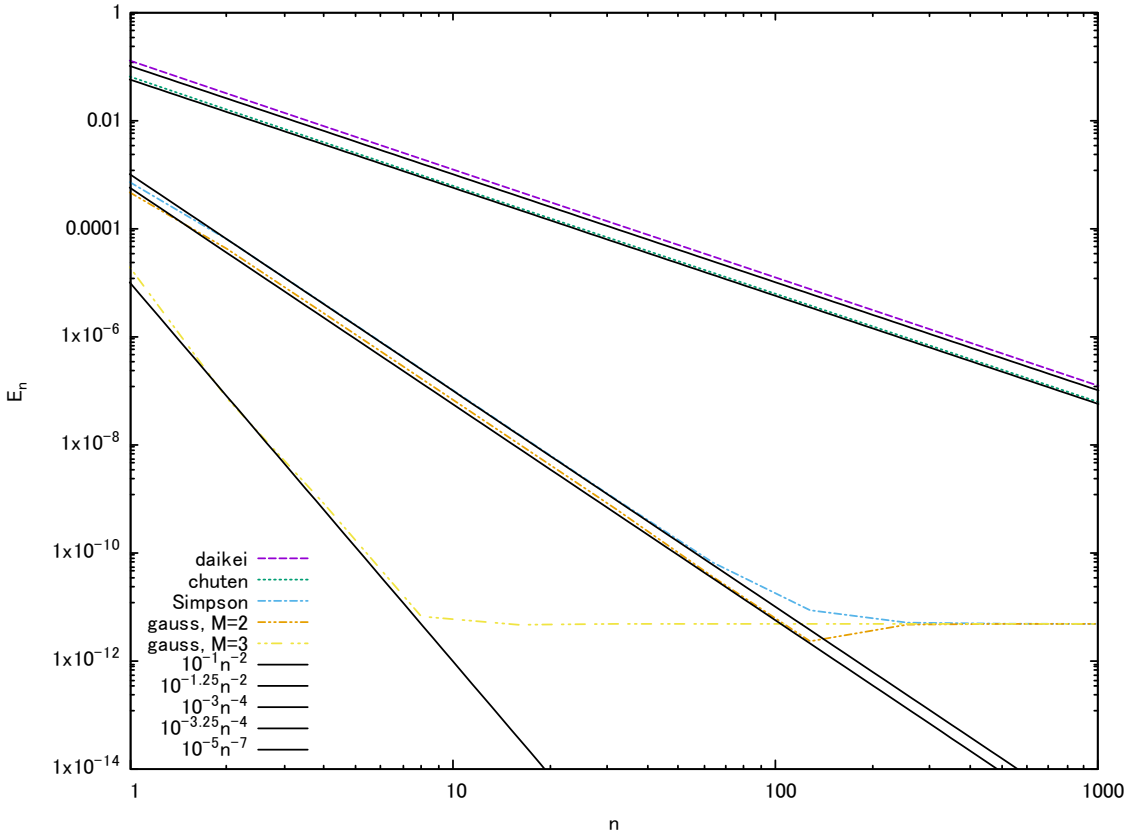



**図 8:** $n$ ごとの $E_n$ の値。

公式が最も速く、$n = 8$ ですでに収束している。このことから、Gauss 型積分公式は分点の取り方を工夫することで少ない分点数でも高い精度の近似が行える方法であると言える。

# 4　課題 4.4

1 次元 Strum-Liouville 型境界値問題

$$\begin{aligned} &-\frac{\mathrm{d}}{\mathrm{d}x}\left(e^{-x^2}\frac{\mathrm{d}\tilde{u}}{\mathrm{d}x}\right) - 6e^{-x^2}\tilde{u}(x) = 0, \quad (0 < x < 1) \\ &\tilde{u}(0) = 0, \;\; \tilde{u}(1) = -4 \end{aligned} \tag{21}$$

を考える。区間 $[0, 1]$ を $x_i \;\; (i = 0, \cdots, 10); x_0 = 0, x_n = 1$ で $n$ 等分割し、有限要素法で解く。

## 4.1　有限要素法

有限要素法は変分問題に基づく数値解法であり、差分法と並んで代表的な変微分方程式の数値解法である。ここでは、1 次元の Strum-Liouville 型微分方程式に有限要素法を適用した結果について述べる。

### 4.1.1 弱形式

Strum-Liouville 型の 2 点境界値問題,

$$\begin{aligned} &-\frac{d}{dx}\left(p(x)\frac{du}{dx}(x)\right)+q(x)u(x)=f(x) \\ &a<x<b \\ &u(a)=u(b)=0 \end{aligned} \tag{22}$$

を考える. ただし, $p(x), q(x), f(x)$ は与えられた滑らかな関数とする. $v(a)=v(b)=0$ を満たす関数 $v$ を式 (22) の両辺にかけ, 区間 $(a,b)$ で積分すると, 左辺第一項は

$$\begin{aligned} &\int_a^b v(x)\left\{-\frac{d}{dx}\left(p(x)\frac{du}{dx}\right)\right\}dx \\ =&-\left[v(x)\left(p(x)\frac{du}{dx}\right)\right]_a^b+\int_a^b \frac{dv}{dx}(x)p(x)\frac{du}{dx}(x)dx \\ =&\int_a^b p(x)\frac{dv}{dx}(x)\frac{du}{dx}(x)dx \end{aligned}$$

となり、以下の式を得る。

$$\begin{aligned} &a(u,v) \\ :=&\int_a^b \left(p(x)\frac{du}{dx}(x)\frac{dv}{dx}(x)+q(x)u(x)v(x)\right)dx \\ =&\int_a^b f(x)v(x)dx \end{aligned} \tag{23}$$

式 (23) は弱形式や変分形式と呼ばれる。有限要素法では問題 (22) を解く代わりに、任意の関数 $v$ に対して弱形式 (23) を満たす関数 $u$ を求めることで解を求める。

### 4.1.2 弱形式の離散化

式 (23) では $u, v$ の取りうる自由度が無限にあるため、計算機で扱うことは困難である。そこで離散化を行い、関数 $u, v$ が属する空間を有限次元空間に制限し、(23) を計算機で扱える式に帰着する。
例えば関数 $u(x)$ が $n$ 次元ベクトル空間に属する場合, $u$ は基底関数 $t_i(x), (i=1,\cdots n)$ と係数 $c_i \in \mathbb{R}$ を用いて,

$$u(x)=\sum_{i=1}^{n} c_i t_i(x) \tag{24}$$

と書ける. 基底 $t_i$ の選び方は様々なものが考えられるが, ここでは折れ線関数による近似を考える. 区間 $(a,b)$ の分割 $a=x_0<x_1<\cdots<x_{n-1}<x_n=b$ に対して, $h_i=x_i-x_{i-1}(i=1,\cdots n)$ とし, $t_i(x)$ を

$$t_i(x)=\begin{cases} \frac{x-x_{i-1}}{h_i} & (x_{i-1}\le x<x_i) \\ \frac{x_{i+1}-x}{h_{i+1}} & (x_i\le x<x_{i+1}) \\ 0 & (\text{others}) \end{cases} \tag{25}$$

とすると, 式 (24) で表される関数 $u(x)$ は点 $u\left(x_i\right)=c_i$ を満たす折れ線関数になる. 関数 $u(x)$ に式 (25) で表される折れ線関数を代入し, さらに $v=t_i$ を代入すると, 式 (23) の左辺は

$$
\begin{aligned}
& a\left(\sum_{j=1}^n u_j t_j, t_i\right) \\
& =\int_a^b\left\{p(x) \frac{d}{d x}\left(\sum_{j=1}^n u_j t_j(x)\right) \frac{d t_i}{d x}(x)+q(x)\left(\sum_{j=1}^n u_j t_j(x)\right) t_i(x)\right\} d x \\
& =\sum_{j=1}^n u_j \int_a^b\left(p(x) \frac{d t_j}{d x}(x) \frac{d t_i}{d x}(x)+q(x) t_i(x) t_j(x)\right) d x \quad (i=1, \cdots, n)
\end{aligned}
$$

となり、右辺は

$$
\int_a^b f(x) t_i(x) d x \quad (i=1, \cdots, n)
$$

となるので、これらをまとめて行列で表すと

$$
A \boldsymbol{u}=\boldsymbol{b} \tag{26}
$$

となる。ここで $A=\left(a_{i j}\right), \boldsymbol{u}=\left(u_i\right), \boldsymbol{b}=\left(b_i\right)$ とし、

$$
a_{i j}=\int_a^b\left(p \frac{d t_i}{d x} \frac{d t_j}{d x}+q t_i t_j\right) d x \tag{27}
$$

$$
b_i=\int_a^b f(x) t_i(x) d x \tag{28}
$$

である。

## 4.2 4.4.1

$\tilde{u}(x)=8 x^3-12 x$ が解であることを確かめる。

$$
\frac{\mathrm{d} \tilde{u}}{\mathrm{d} x}(x)=24 x^2-12 \tag{29}
$$

より (21) の左辺は

$$
\begin{aligned}
& -\frac{\mathrm{d}}{\mathrm{d} x}\left(e^{-x^2}\left(24 x^2-12\right)\right)-6 e^{-x^2}\left(8 x^3-12 x\right) && (30)\\
& =-\left(48 x e^{-x^2}-48 x^3 e^{-x^2}+24 x e^{-x^2}\right)-48 x^3 e^{-x^2}+72 x e^{-x^2} && (31)\\
& =0 && (32)
\end{aligned}
$$

また、$\tilde{u}(0)=0, \tilde{u}(1)=-4$ だから、$\tilde{u}(x)=8 x^3-12 x$ は (21) の解である。

## 4.3 4.4.2

$u=\tilde{u}+4 x$ と変数変換する。すると、$\frac{\mathrm{d} u}{\mathrm{d} x}=\frac{\mathrm{d} \tilde{u}}{\mathrm{d} x}+4$ より、(21) は

$$
\begin{aligned}
-\frac{\mathrm{d}}{\mathrm{d} x}\left(e^{-x^2} \frac{\mathrm{d} u}{\mathrm{d} x}\right)-6 e^{-x^2} u(x)=-16 x e^{-x^2}, \quad (0<x<1) \\
u(0)=u(1)=0
\end{aligned} \tag{33}
$$

となる。いま、

$$-\frac{\mathrm{d}}{\mathrm{d}x}\left(p(x)\frac{\mathrm{d}u}{\mathrm{d}x}\right)+q(x)u(x)=f(x)$$

とすれば、

$$\begin{aligned} f(x) &= -16xe^{-x^2} \\ p(x) &= e^{-x^2} \\ q(x) &= -6e^{-x^2} \end{aligned}$$

である。

## 4.4 4.4.3

$h_i = x_i - x_{i-1}$ として、関数 $t_i(x)$ $(i = 1, \cdots, n-1)$ を

$$t_i(x) = \begin{cases} \frac{x-x_{i-1}}{h_i} & (x_{i-1} \le x < x_i) \\ \frac{x_{i+1}-x}{h_{i+1}} & (x_i \le x < x_i + 1) \\ 0 & (\text{others}) \end{cases}$$

で定義したときの、$\frac{\mathrm{d}t_i}{\mathrm{d}x}(x)$ の値は以下のようになる。

$$\frac{\mathrm{d}t_i}{\mathrm{d}x}(x) = \begin{cases} \frac{1}{h_i} & (x_{i-1} \le x < x_i) \\ \frac{-1}{h_{i+1}} & (x_i \le x < x_i + 1) \\ 0 & (\text{others}) \end{cases} \tag{34}$$

## 4.5 4.4.4

関数がなすベクトル空間の部分空間 $Spant_1, \cdots, t_{n-1}$ において境界値問題 (33) の弱解 $u$ を求める。$u$ を $t_i$ の線形結合で

$$u(x) = \sum_{i=1}^{n-1} c_i t_i(x)$$

と書く。$\boldsymbol{c} = (c_1, \cdots, c_{n-1})^T$ を決定する方程式は

$$A\boldsymbol{c} = \boldsymbol{b}$$

であり、サイズ $n-1$ の正方行列 $A = (a_{ij})$ とベクトル $\boldsymbol{b} = (b_i)$ は

$$\begin{aligned} a_{ij} &= \int_0^1 \left[p(x)\frac{\mathrm{d}t_i}{\mathrm{d}x}\frac{\mathrm{d}t_j}{\mathrm{d}x} + q(x)t_i t_j\right] dx, \quad (i, j = 1, \cdots, n-1) \\ b_i &= \int_0^1 f(x)t_i(x)dx \quad (i = 1, \cdots, n-1) \end{aligned}$$

と定義される。行列 $A$ は対称三重対角行列であり、行列 $A$ とベクトル $\boldsymbol{b}$ の要素を求める際、積分区間を $[0, 1]$ から縮小できることを示す。

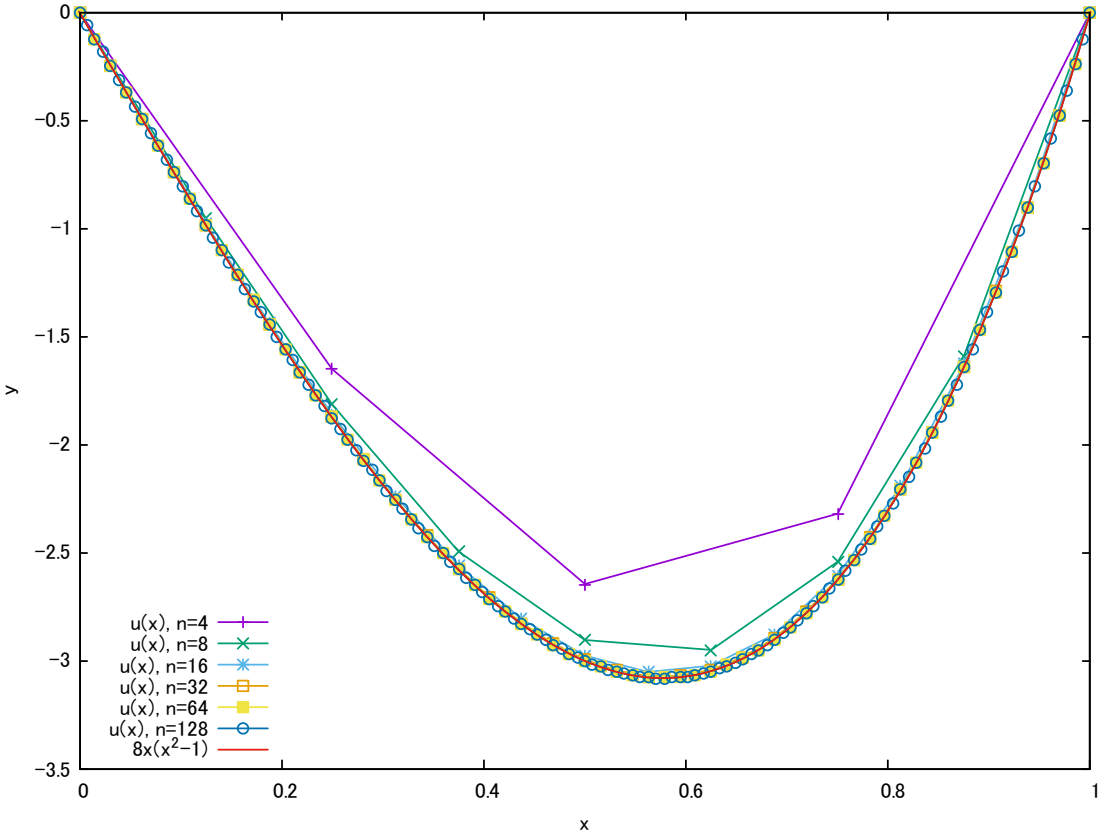



**図 9:** $n$ ごとの $u(x)$ の値。

**■$A$ の性質**　行列 $A$ は対称三重対角行列であることを示す。
式 (25) から、$t_i t_j$ が 0 でないためには、$t_{j-1} \leq t_{i-1} < t_{j+1}$ または $t_{j-1} < t_{i+1} \leq t_j$, すなわち $i-1 \leq j \leq i+1$ が必要である。逆に、$j \leq i-2$ および $j \geq i+2$ では $t_i t_j = 0$ になる。$\frac{\mathrm{d}t_i}{\mathrm{d}x}\frac{\mathrm{d}t_j}{\mathrm{d}x}$ についても同様であるから、$A$ は対称三重三角行列となる。

**■積分区間の縮小**　式 (4.5) において、$a_{ij}$ の被積分関数は $x \notin [x_{i-1}, x_{i+1})$ または $x \notin [x_{j-1}, x_{j+1})$ のとき 0 になる。したがって、積分区間を $[max\{x_{i-1}, x_{j-1}\}, min\{x_{i+1}, x_{j+1}\}]$ に縮小できる。$b_i$ の計算においても同様に、積分区間を $[x_{i-1}, x_{i+1}]$ に縮小できる。

## 4.6　4.4.5

有限要素法を実装し、解を求める。$a_{ij}, b_i$ の計算には $M = 3$ の Gauss 型積分公式を用いる。分割数 n として、得られた結果と厳密解 $u = 8x(x^2 - 1)$ との差の絶対値を $E_n$ とする。

### 4.6.1　計算結果

図 9 に各 $n$ での $u(x)$ の値と厳密解を、図 10 に各 $n$ での $E_n$ の値を示す。

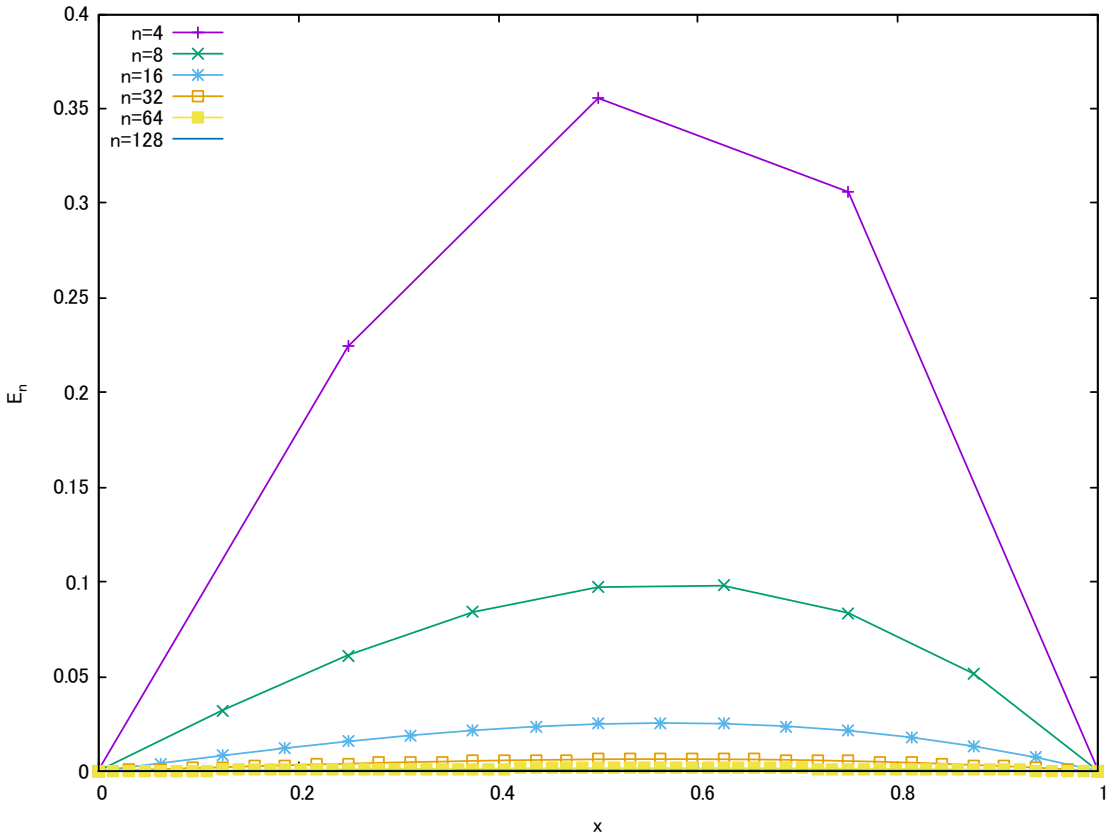



**図 10:** $n$ ごとの $E_n$ の値。

#### 4.6.2 結果の考察

図 10 を見ると、$n$ の増加とともに誤差が減少している様子が見て取れる。また、全体的に端点から遠いほど誤差が大きくなっており、$n$ の値に関わらず $x = 0.5$ から $x = 0.6$ の辺りで誤差が最大になっている。

# 5 付録

実験で用いたプログラムを記す。

**コード 1:** 問題 4.2.1

```
#include<iostream>
#include<cmath>
#include<vector>
#include<fstream>

using namespace std;

double f1(double x){ //関数の定義
    return log(x);
}
double f2(double x){
    return 1/(x*x*x);
}
double f3(double x){
    return 1/(1.0+25*x*x);
}

vector<double> l(double x, vector<double> xarr){ //の計算 l
    int xsize = xarr.size();
    vector<double> l(xsize);
    for (int i=0;i<xsize;i++){
        l[i] = 1.0;
        for (int j=0;j<xsize;j++){
            if (j != i){
                l[i] *= (x-xarr[j])/(xarr[i]-xarr[j]);
            }
        }
    }
    return l;
}

double P1(double x, vector<double> xarr){ //の計算 P
    int lsize = xarr.size();
    vector<double> larr = l(x, xarr);
    double P;
    for (int i=0;i<lsize;i++){
        P += f1(xarr[i])*larr[i];
    }
    return P;
}
double P2(double x, vector<double> xarr){
    int lsize = xarr.size();
    vector<double> larr = l(x, xarr);
    double P;
    for (int i=0;i<lsize;i++){
        P += f2(xarr[i])*larr[i];
    }
    return P;
}
double P3(double x, vector<double> xarr){
```

```
51      int lsize = xarr.size();
52      vector<double> larr = l(x, xarr);
53      double P;
54      for (int i=0;i<lsize;i++){
55          P += f3(xarr[i])*larr[i];
56      }
57      return P;
58  }
59
60  void calc(int n, double a, double b,
61          double(*P)(double,vector<double>), string FILENAME){
62      ofstream outputfile(FILENAME);
63      vector<double> xarr(n+1);
64      double n1 = n;
65      for (int i=0;i<n+1;i++){
66          xarr[i] = a + (b-a)*i/n1;
67      }
68      for(int i=0;i<=100;i++){
69          double x = a + (b-a)*i/100.0;
70          outputfile << x << "␣" << P(x,xarr) << endl;
71      }
72  }
73
74
75  int main(void){
76      calc(2,1,2,P1,"f1_2.txt");
77      calc(4,1,2,P1,"f1_4.txt");
78      calc(8,1,2,P1,"f1_8.txt");
79      calc(16,0.1,2,P1,"f1_16.txt");
80      calc(2,0.1,2,P2,"f2_2.txt");
81      calc(4,0.1,2,P2,"f2_4.txt");
82      calc(8,0.1,2,P2,"f2_8.txt");
83      calc(16,0.1,2,P2,"f2_16.txt");
84      calc(2,-1,1,P3,"f3_2.txt");
85      calc(4,-1,1,P3,"f3_4.txt");
86      calc(8,-1,1,P3,"f3_8.txt");
87      calc(16,-1,1,P3,"f3_16.txt");
88      return 0;
89  }
```

**コード 2:** 問題 4.2.2

```
1   #include<iostream>
2   #include<cmath>
3   #include<vector>
4   #include<fstream>
5
6   using namespace std;
7   double f4(double x){ //の定義 f
8       return 1/x;
9   }
10  double f5(double x){
11      return exp(5*x);
12  }
13  double f6(double x){
14      return 1+sin(x);
15  }
```

```
16 void chuten(double a, double b, double(*f)(double), string FILENAME){ //中点公式
17     ofstream outputfile(FILENAME);
18     for (int k=0;k<11;k++){
19         double sum = 0;
20         double n = pow(2,k);
21         double h = (b-a)/(double)n;
22         for (int i=0;i<n;i++){
23             double xi = a + h*i;
24             double xinext = a + h*(i+1);
25             sum += h*f((xi+xinext)/2.0);
26         }
27         outputfile << n << "␣" << sum <<endl;
28     }
29 }
30
31 void daikei(double a, double b, double(*f)(double), string FILENAME){ //台形公式
32     ofstream outputfile(FILENAME);
33     for (int k=0;k<11;k++){
34         double sum = 0;
35         double n = pow(2,k);
36         double h = (b-a)/(double)n;
37         for (int i=1;i<n;i++){
38             double xi = a + h*i;
39             sum += h*f(xi);
40         }
41         sum += h*(f(a)+f(b))/2.0;
42         outputfile << n << "␣" << sum <<endl;
43     }
44 }
45
46 void simp(double a, double b, double(*f)(double), string FILENAME){ //公式 Simpson
47     ofstream outputfile(FILENAME);
48     for (int k=0;k<11;k++){
49         double sum = 0;
50         double n = pow(2,k);
51         double h = (b-a)/(double)n;
52         for (int i=1;i<n;i++){
53             double xi = a + h*i;
54             double xinext = a + h*(i+1);
55             sum += h*f(xi)/3.0 + h*2*f((xi+xinext)/2.0)/3.0;
56         }
57         sum += h*2*f((a+a+h)/2.0)/3.0;
58         sum += h*(f(a)+f(b))/6.0;
59         outputfile << n << "␣" << sum <<endl;
60     }
61 }
62
63 int main(void){
64     chuten(1,2,f4,"chuten_4.txt");
65     chuten(-1,1,f5,"chuten_5.txt");
66     chuten(0,M_PI,f6,"chuten_6.txt");
67     chuten(0,2*M_PI,f6,"chuten_7.txt");
68
69     daikei(1,2,f4,"daikei_4.txt");
70     daikei(-1,1,f5,"daikei_5.txt");
71     daikei(0,M_PI,f6,"daikei_6.txt");
72     daikei(0,2*M_PI,f6,"daikei_7.txt");
```

```
73
74      simp(1,2,f4,"simp_4.txt");
75      simp(-1,1,f5,"simp_5.txt");
76      simp(0,M_PI,f6,"simp_6.txt");
77      simp(0,2*M_PI,f6,"simp_7.txt");
78
79      return 0;
80  }
```

**コード 3:** 問題 4.2.3

```
1   #include<iostream>
2   #include<cmath>
3   #include<vector>
4   #include<fstream>
5
6   using namespace std;
7   double f4(double x){ //大枠は前問と同じ
8       return 1/x;
9   }
10  double f5(double x){
11      return exp(5*x);
12  }
13  double f6(double x){
14      return 1+sin(x);
15  }
16
17  void chuten(double a, double b, double(*f)(double), double ideal, string FILENAME){
18      ofstream outputfile(FILENAME);
19      for (int k=0;k<11;k++){
20          double sum = 0;
21          double n = pow(2,k);
22          double h = (b-a)/(double)n;
23          for (int i=0;i<n;i++){
24              double xi = a + h*i;
25              double xinext = a + h*(i+1);
26              sum += h*f((xi+xinext)/2.0);
27          }
28          outputfile << n << "␣" << fabs(sum-ideal) <<endl; //理論値との差を取る
29      }
30  }
31
32  void daikei(double a, double b, double(*f)(double), double ideal, string FILENAME){
33      ofstream outputfile(FILENAME);
34      for (int k=0;k<11;k++){
35          double sum = 0;
36          double n = pow(2,k);
37          double h = (b-a)/(double)n;
38          for (int i=1;i<n;i++){
39              double xi = a + h*i;
40              sum += h*f(xi);
41          }
42          sum += h*(f(a)+f(b))/2.0;
43          outputfile << n << "␣" << fabs(sum-ideal) <<endl;
44      }
45  }
46
```

```
47 void simp(double a, double b, double(*f)(double), double ideal, string FILENAME){
48     ofstream outputfile(FILENAME);
49     for (int k=0;k<11;k++){
50         double sum = 0;
51         double n = pow(2,k);
52         double h = (b-a)/(double)n;
53         for (int i=1;i<n;i++){
54             double xi = a + h*i;
55             double xinext = a + h*(i+1);
56             sum += h*f(xi)/3.0 + h*2*f((xi+xinext)/2.0)/3.0;
57         }
58         sum += h*2*f((a+a+h)/2.0)/3.0;
59         sum += h*(f(a)+f(b))/6.0;
60         outputfile << n << "␣" << fabs(sum-ideal) <<endl;
61     }
62 }
63
64 int main(void){
65     double F4 = log(2); //理論値
66     double F5 = (exp(5)-exp(-5))/5.0;
67     double F6 = M_PI + 2;
68     double F7 = 2*M_PI;
69     chuten(1,2,f4,F4,"chuten_4g.txt");
70     chuten(-1,1,f5,F5,"chuten_5g.txt");
71     chuten(0,M_PI,f6,F6,"chuten_6g.txt");
72     chuten(0,2*M_PI,f6,F7,"chuten_7g.txt");
73
74     daikei(1,2,f4,F4,"daikei_4g.txt");
75     daikei(-1,1,f5,F5,"daikei_5g.txt");
76     daikei(0,M_PI,f6,F6,"daikei_6g.txt");
77     daikei(0,2*M_PI,f6,F7,"daikei_7g.txt");
78
79     simp(1,2,f4,F4,"simp_4g.txt");
80     simp(-1,1,f5,F5,"simp_5g.txt");
81     simp(0,M_PI,f6,F6,"simp_6g.txt");
82     simp(0,2*M_PI,f6,F7,"simp_7g.txt");
83
84     return 0;
85 }
```

**コード 4:** 問題 4.3.1

```
1 #include<iostream>
2 #include<cmath>
3 #include<vector>
4 #include<fstream>
5
6 using namespace std;
7
8 void gauss(long double(*f)(long double), long double ideal, long double a, long double b,
      vector<long double> yarr, vector<long double> warr, int size, string FILENAME){ //型積
      分公式 gauss
9     ofstream outputfile(FILENAME);
10    for (int k=0; k<11; k++){
11        long double sum = 0;
12        long double n = pow(2,k);
13        vector<long double> xarr(n+1);
```

```
14          for (int i=0; i<=n; i++){
15              xarr[i] = a + (b-a)*(long double) i/n;
16          }
17
18          for (int i=0; i<n; i++){
19              for (int m=0; m<size; m++){
20                  sum += warr[m]*f((yarr[m]+1)*(xarr[i+1]-xarr[i])/2.0 + xarr[i])*(xarr[i+1]-
                        xarr[i])/2.0;
21              }
22          }
23          outputfile << n << "␣" << fabs(ideal - sum) << endl;
24      }
25  }
26
27  long double f4(long double x){ //の定義 f
28      return 1/x;
29  }
30  long double f5(long double x){
31      return exp(5*x);
32  }
33  long double f6(long double x){
34      return 1+sin(x);
35  }
36
37  int main(void){
38      long double F4 = log(2);
39      long double F5 = (exp(5)-exp(-5))/5.0;
40      long double F6 = M_PI + 2;
41      long double F7 = 2*M_PI;
42
43      vector<long double> yarr1(2);
44      vector<long double> yarr2(3);
45      vector<long double> warr1(2);
46      vector<long double> warr2(3);
47
48      yarr1[0] = -1/sqrt(3);
49      yarr1[1] = 1/sqrt(3);
50      warr1[0] = 1;
51      warr1[1] = 1;
52
53      yarr2[0] = -sqrt(0.6);
54      yarr2[1] = 0;
55      yarr2[2] = sqrt(0.6);
56      warr2[0] = 5/9.0;
57      warr2[1] = 8/9.0;
58      warr2[2] = 5/9.0;
59
60      gauss(f4,F4,1,2,yarr1,warr1,2,"gaussf4_2.txt");
61      gauss(f4,F4,1,2,yarr2,warr2,3,"gaussf4_3.txt");
62      gauss(f5,F5,-1,1,yarr1,warr1,2,"gaussf5_2.txt");
63      gauss(f5,F5,-1,1,yarr2,warr2,3,"gaussf5_3.txt");
64      gauss(f6,F6,0,M_PI,yarr1,warr1,2,"gaussf6_2.txt");
65      gauss(f6,F6,0,M_PI,yarr2,warr2,3,"gaussf6_3.txt");
66      gauss(f6,F7,0,2*M_PI,yarr1,warr1,2,"gaussf7_2.txt");
67      gauss(f6,F7,0,2*M_PI,yarr2,warr2,3,"gaussf7_3.txt");
68      return 0;
69  }
```

**コード 5:** 問題 4.3.2 の Gauuss 型積分公式

```
#include<iostream>
#include<cmath>
#include<vector>
#include<fstream>

using namespace std;

void gauss12(long double(*f)(long double), long double a, long double b, vector<long double> yarr, vector<long double> warr, int size, string FILENAME){ //大枠は前問と同じ
    ofstream outputfile(FILENAME);
    long double ideal = 1.49364826562;
    for (int k=0; k<11; k++){
        long double sum = 0;
        long double n = pow(2,k);
        vector<long double> xarr(n+1);
        for (int i=0; i<=n; i++){
            xarr[i] = a + (b-a)*(long double) i/n;
        }

        for (int i=0; i<n; i++){
            for (int m=0; m<size; m++){
                sum += warr[m]*f((yarr[m]+1)*(xarr[i+1]-xarr[i])/2.0 + xarr[i])*(xarr[i+1]-xarr[i])/2.0;
            }
        }
        outputfile << n << "␣" << fabs(ideal - sum) << endl; //理論値との差を取る
    }
}

void gauss3(long double(*f)(long double), long double a, long double b, vector<long double> yarr, vector<long double> warr, int size, string FILENAME){
    ofstream outputfile(FILENAME);
    long double ideal = 1.49364826562;
    for (int k=0; k<11; k++){
        long double sum = 0;
        long double n = pow(2,k);
        vector<long double> xarr(n+1);
        for (int i=0; i<=n; i++){
            xarr[i] = a + (b-a)*(long double) i/n;
        }

        for (int i=0; i<n; i++){
            for (int m=0; m<size; m++){
                sum += warr[m]*f((yarr[m]+1)*(xarr[i+1]-xarr[i])/2.0 + xarr[i])*(xarr[i+1]-xarr[i])/2.0;
            }
        }
        sum += 2;
        outputfile << n << "␣" << fabs(ideal - sum) << endl;
    }
}

long double f1(long double x){
    return exp(-x)/sqrt(x);
}
```

```
long double f2(long double x){
    return 2*exp(-x*x);
}
long double f3(long double x){
    return (exp(-x)-1)/sqrt(x);
}

int main(void){

    vector<long double> yarr1(2);
    vector<long double> yarr2(3);
    vector<long double> warr1(2);
    vector<long double> warr2(3);

    yarr1[0] = -1/sqrt(3);
    yarr1[1] = 1/sqrt(3);
    warr1[0] = 1;
    warr1[1] = 1;

    yarr2[0] = -sqrt(0.6);
    yarr2[1] = 0;
    yarr2[2] = sqrt(0.6);
    warr2[0] = 5/9.0;
    warr2[1] = 8/9.0;
    warr2[2] = 5/9.0;

    gauss12(f1,0,1,yarr1,warr1,2,"gaussm1_2.txt");
    gauss12(f2,0,1,yarr1,warr1,2,"gaussm2_2.txt");
    gauss3(f3,0,1,yarr1,warr1,2,"gaussm3_2.txt");
    gauss12(f1,0,1,yarr2,warr2,3,"gaussm1_3.txt");
    gauss12(f2,0,1,yarr2,warr2,3,"gaussm2_3.txt");
    gauss3(f3,0,1,yarr2,warr2,3,"gaussm3_3.txt");
    return 0;
}
```

**コード 6:** 問題 4.3.2 の他の積分公式

```
#include<iostream>
#include<cmath>
#include<vector>
#include<fstream>

using namespace std;
double m1(double x){ //大枠はと同じ 4.2.2
    return exp(-x)/sqrt(x);
}
double m2(double x){
    return 2*exp(-x*x);
}
double m3(double x){
    return (exp(-x)-1)/sqrt(x);
}

void chuten(double a, double b, double(*f)(double), double ideal, string FILENAME){
    ofstream outputfile(FILENAME);
    for (int k=0;k<11;k++){
        double sum = 0;
```

```
21          double n = pow(2,k);
22          double h = (b-a)/(double)n;
23          for (int i=0;i<n;i++){
24              double xi = a + h*i;
25              double xinext = a + h*(i+1);
26              sum += h*f((xi+xinext)/2.0);
27          }
28          outputfile << n << "␣" << fabs(sum-ideal) <<endl;
29      }
30  }
31
32  void daikei(double a, double b, double(*f)(double), double ideal, string FILENAME){
33      ofstream outputfile(FILENAME);
34      for (int k=0;k<11;k++){
35          double sum = 0;
36          double n = pow(2,k);
37          double h = (b-a)/(double)n;
38          for (int i=1;i<n;i++){
39              double xi = a + h*i;
40              sum += h*f(xi);
41          }
42          sum += h*(f(a)+f(b))/2.0;
43          outputfile << n << "␣" << fabs(sum-ideal) <<endl;
44      }
45  }
46
47  void simp(double a, double b, double(*f)(double), double ideal, string FILENAME){
48      ofstream outputfile(FILENAME);
49      for (int k=0;k<11;k++){
50          double sum = 0;
51          double n = pow(2,k);
52          double h = (b-a)/(double)n;
53          for (int i=1;i<n;i++){
54              double xi = a + h*i;
55              double xinext = a + h*(i+1);
56              sum += h*f(xi)/3.0 + h*2*f((xi+xinext)/2.0)/3.0;
57          }
58          sum += h*2*f((a+a+h)/2.0)/3.0;
59          sum += h*(f(a)+f(b))/6.0;
60          outputfile << n << "␣" << fabs(sum-ideal) <<endl;
61      }
62  }
63
64  void chuten2(double a, double b, double(*f)(double), double ideal, string FILENAME){
65      ofstream outputfile(FILENAME);
66      for (int k=0;k<11;k++){
67          double sum = 0;
68          double n = pow(2,k);
69          double h = (b-a)/(double)n;
70          for (int i=0;i<n;i++){
71              double xi = a + h*i;
72              double xinext = a + h*(i+1);
73              sum += h*f((xi+xinext)/2.0);
74          }
75          sum += 2;
76          outputfile << n << "␣" << fabs(sum-ideal) <<endl;
77      }
```

```
}

void daikei2(double a, double b, double(*f)(double), double ideal, string FILENAME){
    ofstream outputfile(FILENAME);
    for (int k=0;k<11;k++){
        double sum = 0;
        double n = pow(2,k);
        double h = (b-a)/(double)n;
        for (int i=1;i<n;i++){
            double xi = a + h*i;
            sum += h*f(xi);
        }
        sum += h*(f(a)+f(b))/2.0;
        sum += 2;
        outputfile << n << "␣" << fabs(sum-ideal) <<endl;
    }
}

void simp2(double a, double b, double(*f)(double), double ideal, string FILENAME){
    ofstream outputfile(FILENAME);
    for (int k=0;k<11;k++){
        double sum = 0;
        double n = pow(2,k);
        double h = (b-a)/(double)n;
        for (int i=1;i<n;i++){
            double xi = a + h*i;
            double xinext = a + h*(i+1);
            sum += h*f(xi)/3.0 + h*2*f((xi+xinext)/2.0)/3.0;
        }
        sum += h*2*f((a+a+h)/2.0)/3.0;
        sum += h*(f(a)+f(b))/6.0;
        sum += 2;
        outputfile << n << "␣" << fabs(sum-ideal) <<endl;
    }
}

int main(void){
    double ideal = 1.49364826562;

    chuten(0,1,m1,ideal,"chuten_m1.txt");
    chuten(0,1,m2,ideal,"chuten_m2.txt");
    chuten2(0,1,m3,ideal,"chuten_m3.txt");

    daikei(0,1,m1,ideal,"daikei_m1.txt");
    daikei(0,1,m2,ideal,"daikei_m2.txt");
    daikei2(0,1,m3,ideal,"daikei_m3.txt");

    simp(0,1,m1,ideal,"simp_m1.txt");
    simp(0,1,m2,ideal,"simp_m2.txt");
    simp2(0,1,m3,ideal,"simp_m3.txt");

    return 0;
}
```

**コード 7:** 問題 4.4.5 の u

```
#include<iostream>
```

```
2  #include<cmath>
3  #include<vector>
4  #include<fstream>
5
6  using namespace std;
7
8  #define N 128
9
10 double t(double x, vector<double> xarr, int i){ //の計算 t_i
11     if (xarr[i-1] <= x && x < xarr[i]){
12         return (x - xarr[i-1])/(xarr[i] - xarr[i-1]);
13     }
14     else if (xarr[i] <= x && x < xarr[i+1]){
15         return (xarr[i+1] - x)/(xarr[i] - xarr[i-1]);
16     }
17     else {
18         return 0;
19     }
20 }
21
22 double dt(double x, vector<double> xarr, int i){ //dt_i/の計算 dx
23     if (xarr[i-1] <= x && x < xarr[i]){
24         return 1/(xarr[i] - xarr[i-1]);
25     }
26     else if (xarr[i] <= x && x < xarr[i+1]){
27         return -1/(xarr[i] - xarr[i-1]);
28     }
29     else {
30         return 0;
31     }
32 }
33
34 double f(double x, vector<double> xarr, int i, int j){ //の定義 f
35     return exp(-x*x)*dt(x,xarr,i)*dt(x,xarr,j) -6*exp(-x*x)*t(x,xarr,i)*t(x,xarr,j);
36
37 }
38
39 double a_int(int i, int j){ //の計算 a_ij
40     double sum = 0;
41     vector<double> yarr = {-sqrt(0.6),0,sqrt(0.6)};
42     vector<double> warr = {5/9.0, 8/9.0, 5/9.0};
43     vector<double> xarr(N+1);
44     for (int k=0;k<N+1;k++){
45         xarr[k] = k/(double) N;
46     }
47     for (int k=0; k<N; k++){
48         for (int m=0; m<3; m++){
49             double arg = (yarr[m]+1)*(xarr[k+1]-xarr[k])/2.0 + xarr[k];
50             sum += warr[m]*f(arg,xarr,i,j)*(xarr[k+1]-xarr[k])/2.0;
51         }
52     }
53     return sum;
54 }
55
56 vector<vector<double> > A_calc(void){ //行列への格納
57     vector<vector<double> > A(N-1, vector<double>(N-1));
58     for (int i=0;i<N-1;i++){
```

```
59          for (int j=0;j<N-1;j++){
60              A.at(i).at(j) = a_int(i+1,j+1);
61          }
62      }
63      return A;
64  }
65
66  double b_int(int i){ //の計算 b_i
67      double sum = 0;
68      vector<double> yarr = {-sqrt(0.6),0,sqrt(0.6)};
69      vector<double> warr = {5/9.0, 8/9.0, 5/9.0};
70      vector<double> xarr(N+1);
71      for (int k=0;k<N+1;k++){
72          xarr[k] = k/(double) N;
73      }
74      for (int k=0; k<N; k++){
75          for (int m=0; m<3; m++){
76              double arg = (yarr[m]+1)*(xarr[k+1]-xarr[k])/2.0 + xarr[k];
77              sum += warr[m]*(-16*arg*exp(-arg*arg))*t(arg,xarr,i)*(xarr[k+1]-xarr[k])/2.0;
78          }
79      }
80      return sum;
81  }
82
83  vector<double> b_calc(void){ //行列への格納
84      vector<double> b(N-1);
85      for (int i=0;i<N-1;i++){
86          b[i] = b_int(i+1);
87      }
88      return b;
89  }
90
91  vector<double> gauss_jordan_elim(vector<vector<double> > A, vector<double> d, string
        FILENAME){ //行列の方程式を
        解く
92      ofstream outputfile(FILENAME);
93      vector<double> a(N-1),b(N-2),b_dash(N-2),d_dash(N-1),x(N-1);
94      for (int i=0;i<N-1;i++){
95          a[i] = A.at(i).at(i);
96      }
97      for (int i=0;i<N-2;i++){
98          b[i] = A.at(i).at(i+1);
99      }
100
101     b_dash[0] = b[0]/a[0];
102     for (int i=1;i<N-2;i++){
103         b_dash[i] = b[i]/(a[i]-b[i-1]*b_dash[i-1]);
104     }
105
106     d_dash[0] = d[0]/a[0];
107     for (int i=1;i<N-1;i++){
108         d_dash[i] = (d[i]-b[i-1]*d_dash[i-1])/(a[i]-b[i-1]*b_dash[i-1]);
109     }
110
111     x[N-2] = d_dash[N-2];
112     for (int i=N-1-2;i>=0;i--){
113         x[i] = d_dash[i] - b_dash[i]*x[i+1];
```

```
114         outputfile << x[i] << "␣";
115     }
116     return x;
117 }
118
119 double u(double x, vector<double> c){ //の計算 u
120     double sum = 0;
121     vector<double> xarr(N+1);
122     for (int k=0;k<N+1;k++){
123         xarr[k] = k/(double) N;
124     }
125     for (int k=0;k<N-1;k++){
126         sum += c[k]*t(x,xarr,k+1);
127     }
128     return sum;
129 }
130
131 void calc(vector<double> c, string FILENAME){
132     ofstream outputfile(FILENAME);
133     for (int i=0; i<=N; i++){
134         double s = i/(double)N;
135         outputfile << s << "␣" << u(s,c) << endl;
136     }
137 }
138
139 int main(void){
140     vector<vector<double> > A = A_calc();
141     vector<double> b = b_calc();
142     vector<double> c = gauss_jordan_elim(A,b,"c.txt");
143     calc(c,"SL128.txt");
144     return 0;
145 }
```

**コード 8:** 問題 4.4.5 の誤差

```
1  #include<iostream>
2  #include<cmath>
3  #include<vector>
4  #include<fstream>
5
6  using namespace std;
7
8  #define N 4
9
10 double t(double x, vector<double> xarr, int i){ //大枠は前問と同じ
11     if (xarr[i-1] <= x && x < xarr[i]){
12         return (x - xarr[i-1])/(xarr[i] - xarr[i-1]);
13     }
14     else if (xarr[i] <= x && x < xarr[i+1]){
15         return (xarr[i+1] - x)/(xarr[i] - xarr[i-1]);
16     }
17     else {
18         return 0;
19     }
20 }
21
22 double dt(double x, vector<double> xarr, int i){
```

```
23      if (xarr[i-1] <= x && x < xarr[i]){
24          return 1/(xarr[i] - xarr[i-1]);
25      }
26      else if (xarr[i] <= x && x < xarr[i+1]){
27          return -1/(xarr[i] - xarr[i-1]);
28      }
29      else {
30          return 0;
31      }
32  }
33
34  double f(double x, vector<double> xarr, int i, int j){
35      return exp(-x*x)*dt(x,xarr,i)*dt(x,xarr,j) -6*exp(-x*x)*t(x,xarr,i)*t(x,xarr,j);
36  }
37
38  double a_int(int i, int j){
39      double sum = 0;
40      vector<double> yarr = {-sqrt(0.6),0,sqrt(0.6)};
41      vector<double> warr = {5/9.0, 8/9.0, 5/9.0};
42      vector<double> xarr(N+1);
43      for (int k=0;k<N+1;k++){
44          xarr[k] = k/(double) N;
45      }
46      for (int k=0; k<N; k++){
47          for (int m=0; m<3; m++){
48              double arg = (yarr[m]+1)*(xarr[k+1]-xarr[k])/2.0 + xarr[k];
49              sum += warr[m]*f(arg,xarr,i,j)*(xarr[k+1]-xarr[k])/2.0;
50          }
51      }
52      return sum;
53  }
54
55  vector<vector<double> > A_calc(void){
56      vector<vector<double> > A(N-1, vector<double>(N-1));
57      for (int i=0;i<N-1;i++){
58          for (int j=0;j<N-1;j++){
59              A.at(i).at(j) = a_int(i+1,j+1);
60          }
61      }
62      return A;
63  }
64
65  double b_int(int i){
66      double sum = 0;
67      vector<double> yarr = {-sqrt(0.6),0,sqrt(0.6)};
68      vector<double> warr = {5/9.0, 8/9.0, 5/9.0};
69      vector<double> xarr(N+1);
70      for (int k=0;k<N+1;k++){
71          xarr[k] = k/(double) N;
72      }
73      for (int k=0; k<N; k++){
74          for (int m=0; m<3; m++){
75              double arg = (yarr[m]+1)*(xarr[k+1]-xarr[k])/2.0 + xarr[k];
76              sum += warr[m]*(-16*arg*exp(-arg*arg))*t(arg,xarr,i)*(xarr[k+1]-xarr[k])/2.0;
77          }
78      }
79      return sum;
```

```
80 }
81
82 vector<double> b_calc(void){
83     vector<double> b(N-1);
84     for (int i=0;i<N-1;i++){
85         b[i] = b_int(i+1);
86     }
87     return b;
88 }
89
90 vector<double> gauss_jordan_elim(vector<vector<double> > A, vector<double> d){
91     vector<double> a(N-1),b(N-2),b_dash(N-2),d_dash(N-1),x(N-1);
92     for (int i=0;i<N-1;i++){
93         a[i] = A.at(i).at(i);
94     }
95     for (int i=0;i<N-2;i++){
96         b[i] = A.at(i).at(i+1);
97     }
98
99     b_dash[0] = b[0]/a[0];
100    for (int i=1;i<N-2;i++){
101        b_dash[i] = b[i]/(a[i]-b[i-1]*b_dash[i-1]);
102    }
103
104    d_dash[0] = d[0]/a[0];
105    for (int i=1;i<N-1;i++){
106        d_dash[i] = (d[i]-b[i-1]*d_dash[i-1])/(a[i]-b[i-1]*b_dash[i-1]);
107    }
108
109    x[N-2] = d_dash[N-2];
110    for (int i=N-1-2;i>=0;i--){
111        x[i] = d_dash[i] - b_dash[i]*x[i+1];
112    }
113    return x;
114 }
115
116 double u(double x, vector<double> c){
117     double sum = 0;
118     vector<double> xarr(N+1);
119     for (int k=0;k<N+1;k++){
120         xarr[k] = k/(double) N;
121     }
122     for (int k=0;k<N-1;k++){
123         sum += c[k]*t(x,xarr,k+1);
124     }
125     return sum - 8*x*(x*x-1); //理論値との差を取る
126 }
127
128 void calc(vector<double> c, string FILENAME){
129     ofstream outputfile(FILENAME);
130     for (int i=0; i<=N; i++){
131         double s = i/(double)N;
132         outputfile << s << "␣" << u(s,c) << endl;
133     }
134 }
135
136 int main(void){
```

```
137     vector<vector<double> > A = A_calc();
138     vector<double> b = b_calc();
139     vector<double> c = gauss_jordan_elim(A,b);
140     calc(c,"gosa" + to_string(N) + ".txt");
141     return 0;
142 }
```

# 6　参考文献

## 参考文献

[1] 実験演習ワーキンググループ (2022),「数理工学実験テキスト_2022 年版」