令和四年度後期 数理工学実験 課題レポート

関数の補間と数値積分

情報学科 2 回生 平田蓮

学生番号: 1029342830

実験日: 11 月 8, 14, 15, 28 日

実験場所: 京都大学工学部総合校舎数理計算機室

12 月 5 日 提出

目次

1　目的

本実験では数値積分を扱う。数値積分は理学・工学のさまざまな場面で現れるが、これを解析的に解くことは特別な場合を除いて非常に困難であるため、本実験では関数補間を用いて近似的に積分の数値解を得ることを目標とする。

2　原理

式 (2.1) に示す積分を解くことを考える。本節では、のちに行う課題で用いる積分法について記す。

$$\int_a^b f(x)\mathrm{d}x \tag{2.1}$$

2.1　Newton-Cotes 積分公式

Newton-Cotes 積分公式は最も基本的な積分公式である。これは、Lagrange 補間を用いるため、それについても以下に記す。

2.1.1　Lagrange 補間

$f(x)$ を $n-1$ 次多項式 $P(x)$ で表すことを考える。$x \in [a,b]$ から n 点 $x_1, x_2, \cdots, x_n$ を選び、$P(x_i) = f(x_i)$ となるように $P(x)$ を構成する。$P(x)$ の i 次の係数を a_i とすると、各 $i = 0, 1, \cdots, n-1$ について、

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

を満たせば良い。これは次のように書き直せる。

$$\begin{pmatrix} 1 & x_1 & \cdots & x_1^{n-1} \\ 1 & x_2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & \cdots & x_n^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_n) \end{pmatrix}$$

これの係数行列の行列式は、

$$\begin{vmatrix} 1 & x_1 & \cdots & x_1^{n-1} \\ 1 & x_2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & \cdots & x_n^{n-1} \end{vmatrix} = \prod_{i=1}^{n} \prod_{j=i+1}^{n} (x_i - x_j) \neq 0$$

となるため、条件を満たす $P(x)$ は一意に定まることがわかり、それは

$$P(x) = \sum_{i=1}^{n} \frac{\displaystyle\prod_{j=1, j\neq i}^{n} (x - x_j)}{\displaystyle\prod_{j=1, j\neq i}^{n} (x_i - x_j)} f(x_i) \tag{2.2}$$

と書ける。

2.1.2 Newton-Cotes 積分公式

分点 $\{x_i\}$ を用いて Lagrange 補間を行い、式 (2.1) の積分値を計算する。式 (2.1) を書き直すと、

$$\begin{aligned}\int_a^b f(x)\mathrm{d}x &= \int_a^b P(x)\mathrm{d}x \\ &= \int_a^b \sum_{i=1}^{n} \frac{\prod_{j=1,j\neq i}^{n}(x-x_j)}{\prod_{j=1,j\neq i}^{n}(x_i-x_j)} f(x_i)\mathrm{d}x \\ &= \sum_{i=1}^{n} f(x_i) \int_a^b \frac{\prod_{j=1,j\neq i}^{n}(x-x_j)}{\prod_{j=1,j\neq i}^{n}(x_i-x_j)}\mathrm{d}x\end{aligned}$$

を得る。これを Newton-Cotes 積分公式と呼ぶ。この公式は、分点の取り方によって変化するが、本実験では以下に示す 3 種類を使う。

■中点公式　一つの分点 $x_1 = \dfrac{a+b}{2}$ のみを考える。

$\displaystyle\int_a^b \frac{\prod_{j=1,j\neq i}^{n}(x-x_j)}{\prod_{j=1,j\neq i}^{n}(x_i-x_j)}\mathrm{d}x = \int_a^b \mathrm{d}x = b-a$ となるので、$\displaystyle\int_a^b f(x)\mathrm{d}x \approx (b-a)f(x_1)$ と近似できる。これを中点公式という。

■台形公式　二つの分点 $x_1 = a, x_2 = b$ を考える。

$\displaystyle\int_a^b \frac{\prod_{j=1,j\neq i}^{n}(x-x_j)}{\prod_{j=1,j\neq i}^{n}(x_i-x_j)}\mathrm{d}x = \int_a^b \frac{1}{2}\mathrm{d}x = \frac{b-a}{2}$ となるので、$\displaystyle\int_a^b f(x)\mathrm{d}x \approx \frac{b-a}{2}\{f(x_1)+f(x_2)\}$ と近似できる。これを台形公式という。

■Simpson 公式　三つの分点 $x_1 = a, x_2 = \dfrac{b-a}{2}, x_3 = b$ を考える。

$$\int_a^b \frac{\prod_{j=1,j\neq i}^{n}(x-x_j)}{\prod_{j=1,j\neq i}^{n}(x_i-x_j)}\mathrm{d}x = \begin{cases}\dfrac{b-a}{6} & i=1,3 \\ \dfrac{2}{3}(b-a) & i=2\end{cases}$$

となるので、$\displaystyle\int_a^b f(x)\mathrm{d}x \approx \frac{b-a}{6}\{f(x_1)+4f(x_2)+f(x_3)\}$ と近似できる。これを台形公式という。

2.1.3 複合公式

前節で述べた公式は、$x \in [a,b]$ において $f(x)$ を定数、一次関数、二次関数で近似して得ているため、その実際の値との誤差は $|b-a|$ に比例して大きくなってしまう。そこで、実際の積分範囲を十分に細かく分割し、それぞれに積分公式を用いて計算を行うのが一般的である。

$x_i = a + ih \left(i = 0, 1, \cdots, n,\ h = \dfrac{b-a}{n} \right)$ として、$[a_i, a_{i+1}]$ にそれぞれの積分公式を適用すると、次のように書ける。

■中点複合公式

$$\int_a^b f(x)\mathrm{d}x \approx h \sum_{i=0}^{n-1} f\left(\frac{x_i + x_{i+1}}{2} \right) \tag{2.3}$$

■台形複合公式

$$\int_a^b f(x)\mathrm{d}x \approx \frac{h}{2} \left\{ f(x_0) + \sum_{i=1}^{n-1} f(x_i) + f(x_n) \right\} \tag{2.4}$$

■Simpson 複合公式

$$\int_a^b f(x)\mathrm{d}x \approx \frac{h}{6} \left\{ f(x_0) + \sum_{i=1}^{n-1} f(x_i) + f(x_n) + 4 \sum_{i=0}^{n-1} f\left(\frac{x_i + x_{i+1}}{2} \right) \right\} \tag{2.5}$$

2.2 Gauss 型積分公式

二つの関数 $f(x), g(x)$ の内積 (f,g) を以下で定義する。

$$(f,g) = \int_{-1}^{1} f(x)g(x)\mathrm{d}x$$

これが 0 となるとき、二つの関数は直交するという。任意の $n-1$ 次多項式と直交する n 次式多項式 $P_n(x)$ が存在することが知られ、これを Legendre 多項式と呼ぶ。

$f(x)$ を $P_n(x)$ で割った商を $Q(x)$, 余りを $R(x)$ とすると、

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

と書ける。ここで、$Q(x), R(x)$ は $n-1$ 次多項式であることに注意する。$a=-1, b=1$ として式 (2.1) を解くことを考えると、

$$\begin{aligned} \int_{-1}^{1} f(x)\mathrm{d}x &= \int_{-1}^{1} \{P_n(x)Q(x) + R(x)\}\mathrm{d}x \\ &= \int_{-1}^{1} R(x)\mathrm{d}x \ (\because P_n(x) \text{ と } Q(x) \text{ は直交}) \end{aligned}$$

と書ける。ここで、$R(x)$ の Lagrange 補間は $R(x)$ に等しいため、

$$\int_{-1}^{1} R(x)\mathrm{d}x = \sum_{i=1}^{n} R(x_i) \int_{-1}^{1} \frac{\displaystyle\prod_{j=1, j\neq i}^{n} (x - x_j)}{\displaystyle\prod_{j=1, j\neq i}^{n} (x_i - x_j)} \mathrm{d}x$$

となり、$\{x_i\}$ に $P_n(x)$ の零点を用いると、上の結果と合わせて、

$$\int_{-1}^{1} f(x)\mathrm{d}x = \sum_{i=1}^{n} f(x_i) \int_{-1}^{1} \frac{\prod_{j=1,j\neq i}^{n} (x - x_j)}{\prod_{j=1,j\neq i}^{n} (x_i - x_j)} \mathrm{d}x$$

を得る。これを Gauss 型積分公式と呼ぶ。これは積分範囲が $[-1, 1]$ であるが、適切に変数変換を行うことで任意の範囲で積分を行うことが可能である。

2.2.1 複合公式

Gauss 型積分公式も、一般的に複合公式を作成して計算を行う。$x_i = a + ih \left(i = 0, 1, \cdots, n,\ h = \dfrac{b-a}{n} \right)$ とすると、

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

である。$y = 2\dfrac{x - x_i}{x_{i+1} - x_i} - 1$ と変数変換を行うと、$x \in [x_i, x_{i+1}]$ は $y \in [-1, 1]$ となり、Gauss 型積分公式を適用できる。M 次 Legendre 多項式を用いた Gauss 型積分公式を適用すると、

$$\begin{aligned}
\int_a^b f(x)\mathrm{d}x &= \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} f(x)\mathrm{d}x \\
&= \sum_{i=0}^{n-1} \int_{-1}^{1} f\left(\frac{(y+1)(x_{i+1} - x_i)}{2} + x_i \right) \frac{x_{i+1} - x_i}{2} \mathrm{d}y \\
&\approx \sum_{i=0}^{n-1} \sum_{m=1}^{M} w_m f\left(\frac{(y_m+1)(x_{i+1} - x_i)}{2} + x_i \right) \frac{x_{i+1} - x_i}{2}
\end{aligned} \tag{2.6}$$

と書ける。ここで、y_m は Legendre 多項式の m 番目の零点、w_m は M 次 Lagrange 補間の $m-1$ 次の係数である。

3 課題

3.1 課題 4.2

区間 $[a, b]$ における $n+1$ 個の分点を以下で定める:

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

3.1.1

以下に示す三つの関数について、x_i における値を求め、$n = 2, 4, 8, 16$ として Lagrange 補間で n 次多項式 $P(x)$ を求める。

- $f_1(x) = \ln x \ (x \in [1, 2])$
- $f_2(x) = \dfrac{1}{x^3} \ (x \in [0.1, 2])$

- $f_3(x) = \dfrac{1}{1+25x^2}\ (x \in [-1, 1])$

作成したコードをリスト 1, 2, 3, 4 に示す。

リスト 1　式 (2.2) の実装

```
#include <iostream>
#include <vector>

class Lagrange {
private:
    const double (*f)(const double);
    const double start, end;
    const int n;

    std::vector<double> xs;

    void calculate_points() {
        for (int i = 0; i <= this->n; i++) {
            this->xs[i] = this->start + (this->end - this->start) / (double)n * (double)i;
        }
    }

    const double l(
        const int i,
        const double x
    ) {
        double return_value = 1;
        for (int j = 0; j <= this->n; j++) {
            if (j == i) {
                continue;
            }

            return_value *= (x - this->xs[j]) / (this->xs[i] - this->xs[j]);
        }

        return return_value;
    }

    const double P(const double x) {
        double return_value = 0;
        for (int i = 0; i <= this->n; i++) {
            return_value += f(this->xs[i]) * this->l(i, x);
        }

        return return_value;
    }

public:
    Lagrange(
        const double (*f)(const double),
        const double start,
        const double end,
        const int n
    ) : f(f), start(start), end(end), n(n), xs(n + 1, 0) {
        this->calculate_points();
    }

```

```
53      void print_interpolated(
54          const double dx
55      ) {
56          for (double x = this->start; x <= this->end; x += dx) {
57              std::cout << x << "␣" << this->P(x) << std::endl;
58          }
59      }
60  };
```

リスト 2　$f_1(x)$ の実装

```
1  #include <cmath>
2
3  inline constexpr double f1(const double x) {
4      return log(x);
5  }
```

リスト 3　$f_2(x)$ の実装

```
1  inline constexpr double f2(const double x) {
2      return 1.0 / x / x / x;
3  }
```

リスト 4　$f_3(x)$ の実装

```
1  inline constexpr double f3(const double x) {
2      return 1.0 / (1.0 + 25 * x * x);
3  }
```

■結果・考察　作成したグラフを図 1, 2, 3 に示す。

$f_1(x)$ の近似は、$n = 2$ の時点で人の目では誤差がわからない程度の非常に高い精度で行われていることがわかり、$f_1(x)$ は $x \in [1, 2]$ において二次関数に近い形をとっていることがわかる。一方、$f_{2,3}(x)$ は n が増加するにつれ $x = 0$ 付近から徐々に近似の精度が上がっていることがわかる。これは、Lagrange 補間は $n - 1$ 次までのマクローリン展開を行っているためであると考えられる。

3.1.2

以下に示す四つの関数について、それぞれの範囲における定積分を考える。

- $f_4(x) = \dfrac{1}{x}$ $(x \in [1, 2])$
- $f_5(x) = e^{5x}$ $(x \in [-1, 1])$
- $f_6(x) = 1 + \sin x$ $(x \in [0, \pi])$
- $f_7(x) = 1 + \sin x$ $(x \in [0, 2\pi])$

これらの積分を中点複合公式、台形複合公式、Simpson 複合公式で計算する。分割数 n における計算結果を I_n とし、$n \to \infty$ においてこれが理論値 $\displaystyle\int_a^b f_i(x)\mathrm{d}x$ に収束するか否かを調べる。

なお、それぞれの理論値は以下に示す通りである。

- $\displaystyle\int_1^2 f_4(x)\mathrm{d}x = \ln 2 \approx 0.693147$

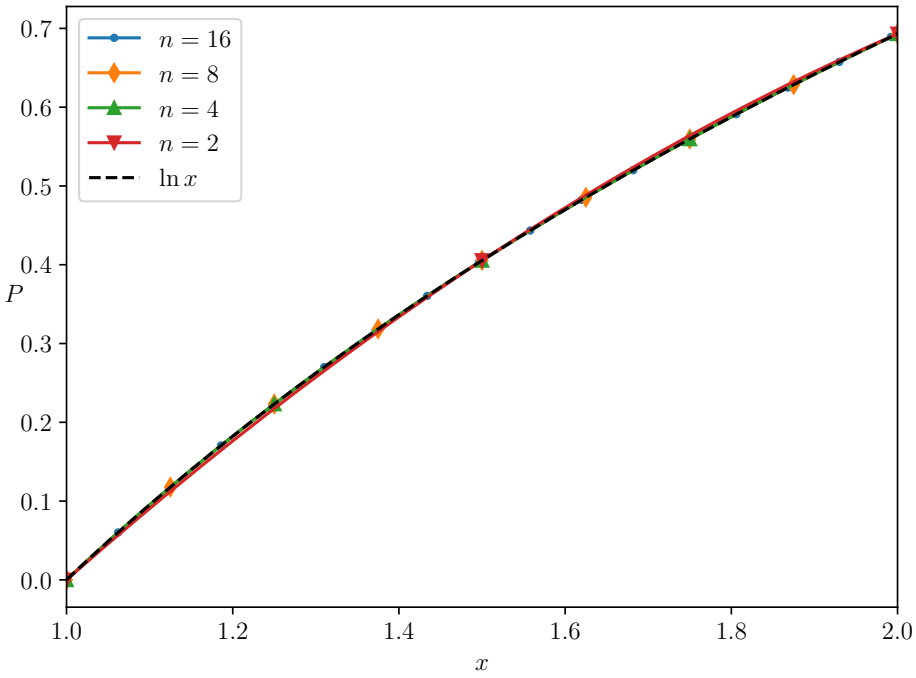

図 1　$f_1(x)$ の Lagrange 補間

- $\displaystyle\int_{-1}^{1} f_5(x)\mathrm{d}x = \frac{e^5 - e^{-5}}{5} \approx 29.6813$
- $\displaystyle\int_{0}^{\pi} f_6(x)\mathrm{d}x = 2 + \pi \approx 5.14159$
- $\displaystyle\int_{0}^{2\pi} f_7(x)\mathrm{d}x = 2\pi \approx 6.28319$

作成したコードをリスト 5, 6, 7, 8, 9, 10 に示す。

リスト 5　式 (2.3) の実装

```
const double midpoint(
    const double (*f)(const double),
    const double start,
    const double end,
    const int n
) {
    const double h = (end - start) / (double)n;
    double lastx = start;
    double return_value = 0;
    for (double x = lastx + h; x <= end; lastx = x, x += h) {
        return_value += f((lastx + x) / 2.0);
    }

    return return_value * h;
}
```

リスト 6　式 (2.4) の実装

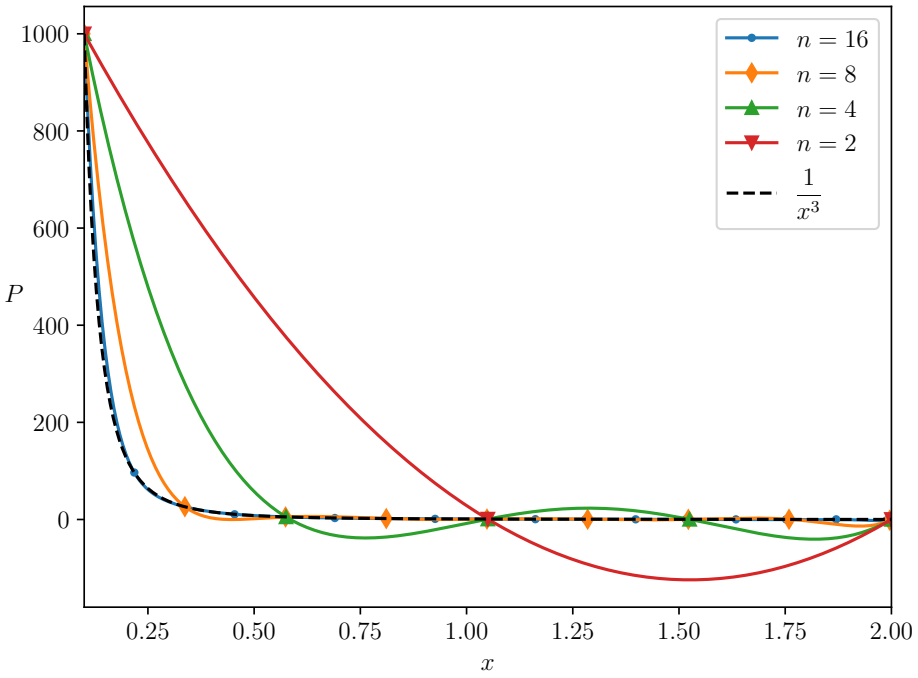

図 2　$f_2(x)$ の Lagrange 補間

```
const double trapezoid(
    const double (*f)(const double),
    const double start,
    const double end,
    const int n
) {
    const double h = (end - start) / (double)n;
    double return_value = 0;
    for (double x = start + h; x < end; x += h) {
        return_value += 2.0 * f(x);
    }

    return (f(start) + return_value + f(end)) * h / 2.0;
}
```

リスト 7　式 (2.5) の実装

```
const double simpson(
    const double (*f)(const double),
    const double start,
    const double end,
    const int n
) {
    const double h = (end - start) / (double)n;
    double return_value = f(start) + f(end);
    double lastx = start;
    for (double x = start + h; x <= end; lastx = x, x += h) {
        return_value += 2.0 * f(lastx) + 4 * f((lastx + x) / 2.0);
    }
```

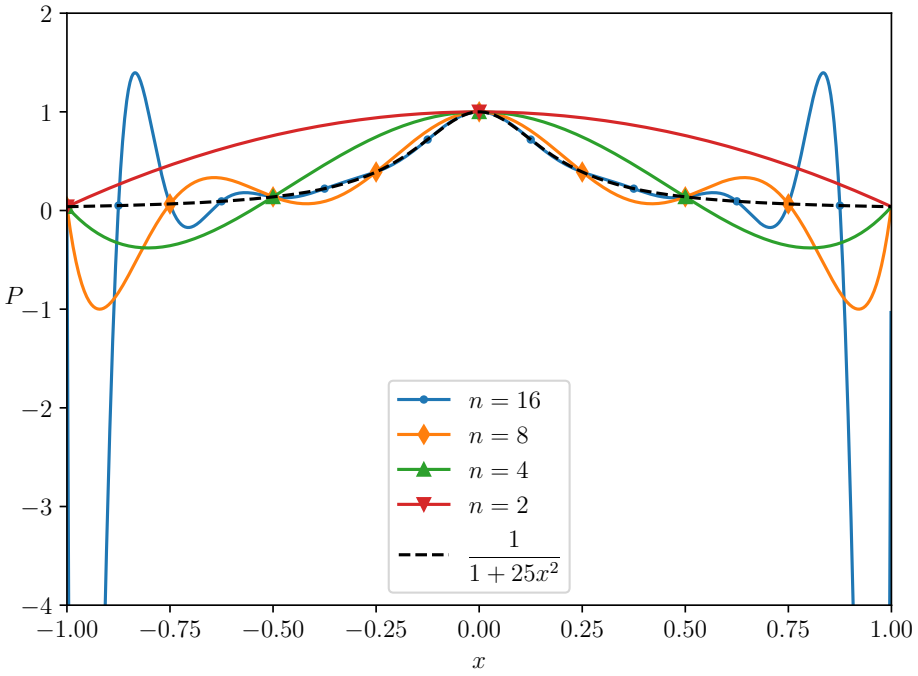

図 3　$f_3(x)$ の Lagrange 補間

```
13
14      return return_value * h / 6.0;
15  }
```

リスト 8　$f_4(x)$ の実装

```
1  inline constexpr double f4(const double x) {
2      return 1.0 / x;
3  }
```

リスト 9　$f_5(x)$ の実装

```
1  #include <cmath>
2
3  inline constexpr double f5(const double x) {
4      return exp(5.0 * x);
5  }
```

リスト 10　$f_{6,7}(x)$ の実装

```
1  #include <cmath>
2
3  inline constexpr double f6(const double x) {
4      return 1.0 + sin(x);
5  }
```

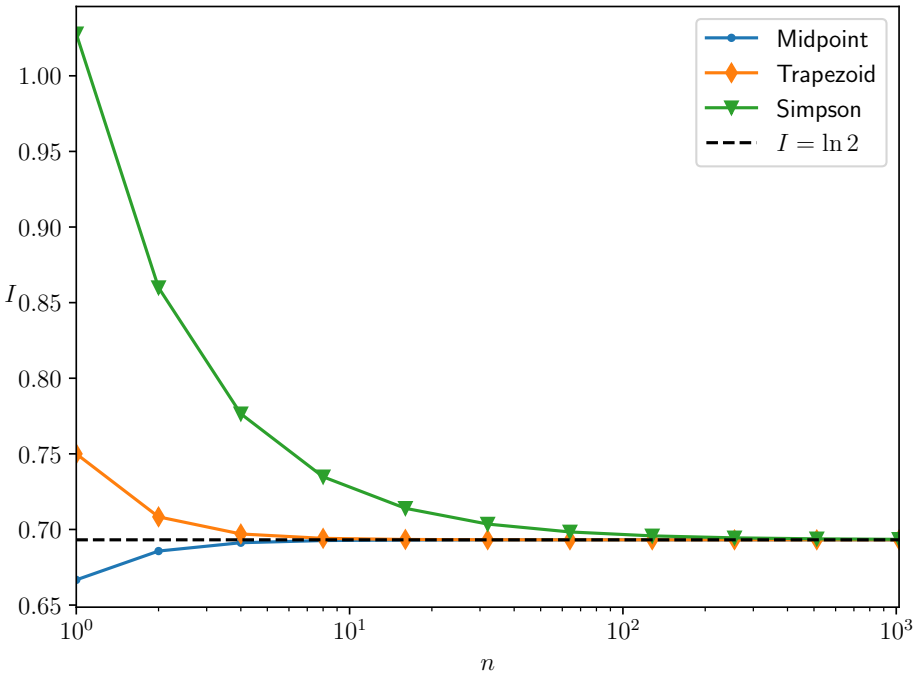

図 4　$f_4(x)$ の積分値とその理論値 (片対数グラフ)。Midpoint, Trapezoid, Simpson はそれぞれ複合中点公式、複合台形公式、複合 Simpson 公式を用いて計算した値を示す。

■結果・考察　作成したグラフを図 4, 5, 6, 7 に示す。

$f_4(x), f_7(x)$ は、複合中点公式と複合台形公式に比べて複合 Simoson 公式の精度が振るっていないが、逆に $f_5(x), f_6(x)$ では複合台形公式が n が小さいときに大きい誤差を出していることがわかる。これらは、定数、一次関数、二次関数それぞれによる近似がどれだけ関数の形に当てはまるかによる。

また、$f_7(x)$ は $n = 1, 2, 4, 16$ の際に誤差が 0 になっているが、これは関数と積分範囲の関係上、偶然一次関数による近似と理論値が一致したためであると考えられる。

3.1.3

先の問題において、計算値 $I_n(f_i)$ と理論値 $I(f_i)$ の誤差 $E_n(f_i) = |I_n(f_i) - I(f_i)|$ を定義する。これの n への依存性を調べる。

■結果・考察　それぞれの関数における誤差を図 8, 9, 10, 11 に示す。

図 8 と 9 から、複合中点公式と複合台形公式の精度は n^{-2} に比例することが見てとれる。しかし、$f_{6,7}(x)$ では一部 n^{-1} に比例している。これは、$1 + \sin x$ の関数形が近似に用いられる定数及び一次関数とは離れているためであると考えられる。

一方、複合 Simpson 公式は $f_{4,6,7}(x)$ では n^{-1} に比例しているが、$f_5(x)$ では $n^{-3.5}$ に比例していることが図よりわかる。これは、$f_5(x)$ の関数形が Simpson 公式の用いる二次関数と非常に似ているためであると考えられる。($f_1(x)$ の Lagrange 補間を参照)

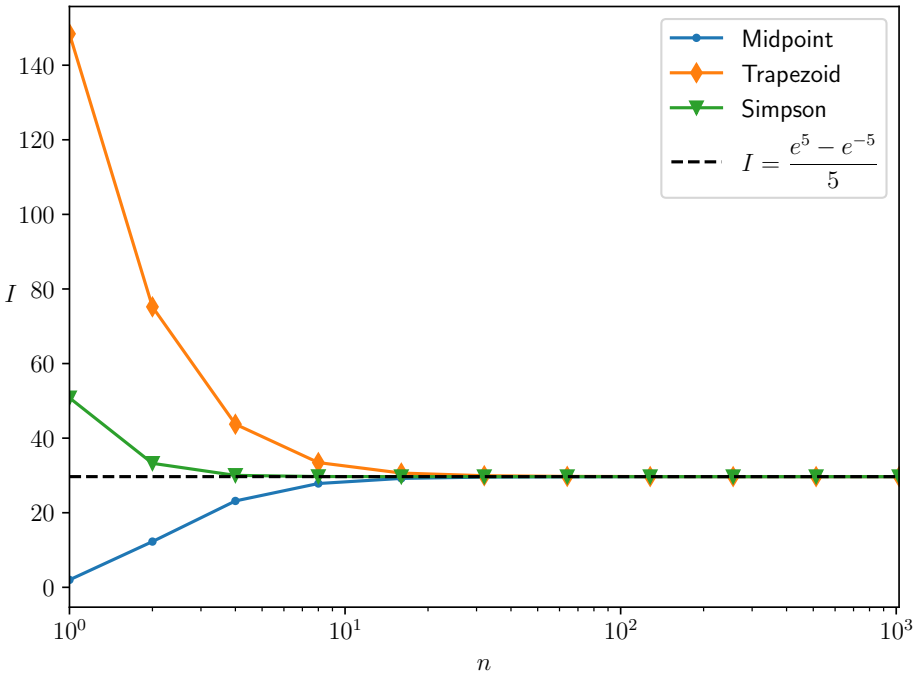

図 5　$f_5(x)$ の積分値とその理論値 (片対数グラフ)。Midpoint, Trapezoid, Simpson はそれぞれ複合中点公式、複合台形公式、複合 Simpson 公式を用いて計算した値を示す。

表 1　$M = 2, 3$ の Gauss 型積分公式で用いる定数

M	2		3		
m	1	2	1	2	3
y_m	$-\frac{1}{\sqrt{3}}$	$\frac{1}{\sqrt{3}}$	$-\sqrt{\frac{3}{5}}$	0	$\sqrt{\frac{3}{5}}$
w_m	1	1	$\frac{5}{9}$	$\frac{8}{9}$	$\frac{5}{9}$

3.2　課題 4.3

3.2.1

3.1.3 節と同様の誤差を $M = 2, 3$ の Gauss 型積分公式について調べる。なお、それぞれの M における Legendre 多項式の零点及び、Lagrange 補間の係数は表 1 に示す。

作成したコードをリスト 11 に示す。

リスト 11　式 (2.6) の実装

```
1 #include <vector>
2
3 const double gauss(
4     const double (*f)(const double),
5     const double start,
6     const double end,
```

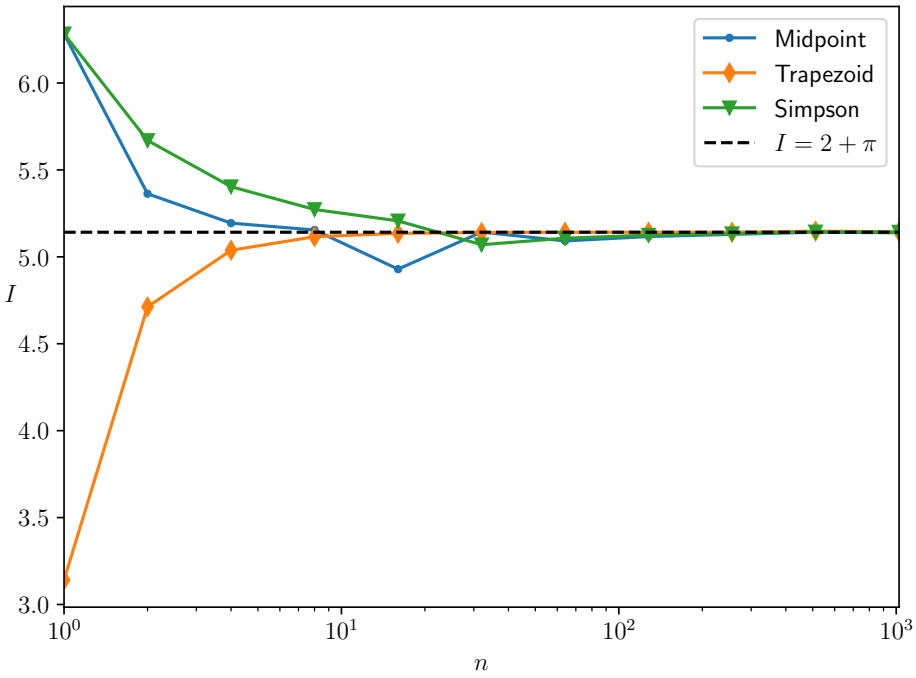

図 6　$f_6(x)$ の積分値とその理論値 (片対数グラフ)。Midpoint, Trapezoid, Simpson はそれぞれ複合中点公式、複合台形公式、複合 Simpson 公式を用いて計算した値を示す。

```
    const int n,
    const int M,
    const std::vector<double> y,
    const std::vector<double> w
) {
    const double h = (end - start) / (double)n;
    double return_value = 0;
    for (double x = start + h; x <= end; x += h) {
        for (int m = 0; m < M; m++) {
            return_value += w[m] * f((y[m] + 1.0) * h / 2.0 + x) * h / 2.0;
        }
    }

    return return_value;
}
```

■結果・考察　作成したグラフを図 12, 13 に示す。

図より、Gauss 型積分公式の誤差は、$n^{-0.5m}$ に比例することがわかる。

3.2.2

次の定積分を考える。

$$\int_0^1 \frac{e^{-x}}{\sqrt{x}} \mathrm{d}x \approx 1.49364826562$$

これを $M = 2, 3$ の Gauss 型積分公式を用いて、以下の三つの方法で計算する。

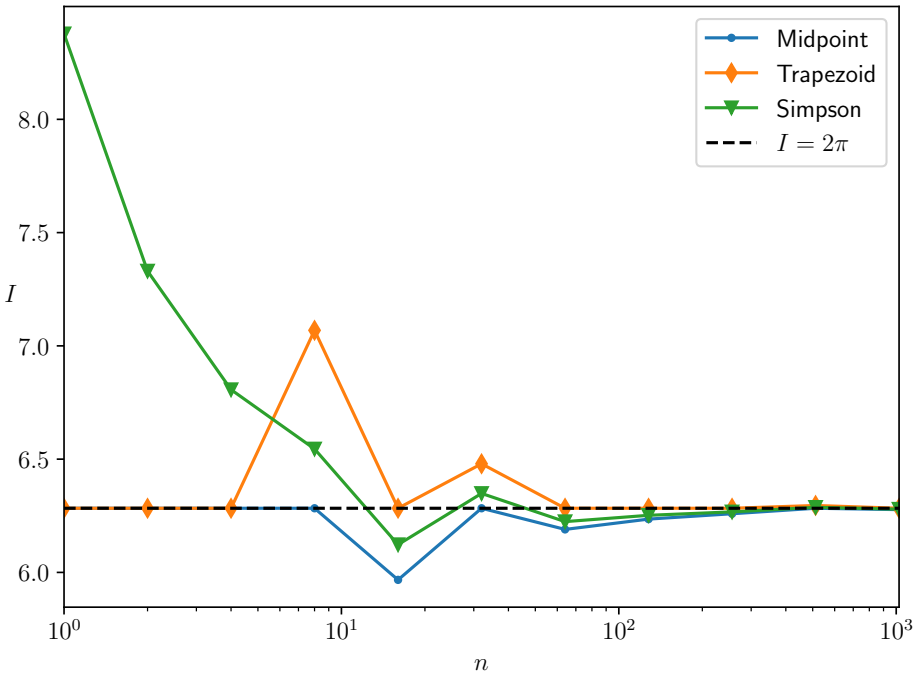

図 7　$f_7(x)$ の積分値とその理論値 (片対数グラフ)。Midpoint, Trapezoid, Simpson はそれぞれ複合中点公式、複合台形公式、複合 Simpson 公式を用いて計算した値を示す。

1. $\displaystyle\int_0^1 \frac{e^{-x}}{\sqrt{x}}\mathrm{d}x$ を計算する。
2. $t = \sqrt{x}$ と変数変換を行い、$\displaystyle\int_0^1 2e^{-t^2}\mathrm{d}t$ 計算する。
3. $\displaystyle\int_0^1 \frac{e^{-x}}{\sqrt{x}}\mathrm{d}x = \int_0^1 \frac{1}{\sqrt{x}}\mathrm{d}x + \int_0^1 \frac{e^{-x}-1}{\sqrt{x}}\mathrm{d}x = 2 + \int_0^1 \frac{e^{-x}-1}{\sqrt{x}}\mathrm{d}x$ を計算する。

作成したコードをリスト 12, 13, 14 に示す。

リスト 12　方法 1 の被積分関数の実装

```cpp
#include <cmath>

inline constexpr double f(const double x) {
    return exp(-x) / sqrt(x);
}
```

リスト 13　方法 2 の被積分関数の実装

```cpp
#include <cmath>

inline constexpr double f(const double t) {
    return 2 * exp(- t * t);
}
```

リスト 14　方法 3 の被積分関数の実装

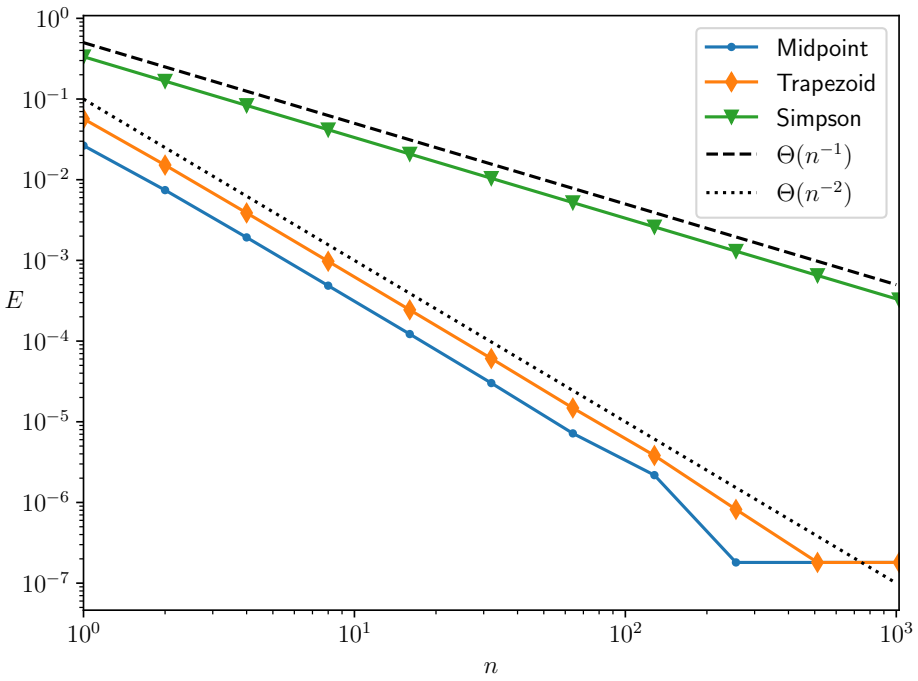

図 8　$f_4(x)$ の積分値とその理論値との誤差 (両対数グラフ)。

```cpp
#include <cmath>

inline constexpr double f(const double x) {
    return (exp(-x) - 1.0) / sqrt(x);
}
```

■結果・考察　作成したグラフを図 14, 15 に示す。

図を見ると、M に関わらず、方法 1 を用いると精度が低いことがわかる。これは、それぞれの被積分関数に原因があると考えられる。図 16 にそれぞれの被積分関数を示す。

それぞれの被積分関数を比較すると、方法 1 の場合のみ、$x = 0$ で発散していることがわかる。これが精度の低下につながっていると考えられる。また、同様の理由 ($f(0)$ が計算不能) で方法 1 に台形公式や Simpson を用いることはできないが、方法 2,3 には適用可能である。

3.3　課題 4.4

Strum-Liouville 型境界値問題

$$-\frac{\mathrm{d}}{\mathrm{d}x}\left(e^{-x^2}\frac{\mathrm{d}\tilde{u}}{\mathrm{d}x}\right) - 6e^{-x^2}\tilde{u} = 0 \ (0 < x < 1), \tilde{u}(0) = 0, \tilde{u}(1) = -4 \tag{3.1}$$

を、以下の手順で有限要素法を用いて解く。

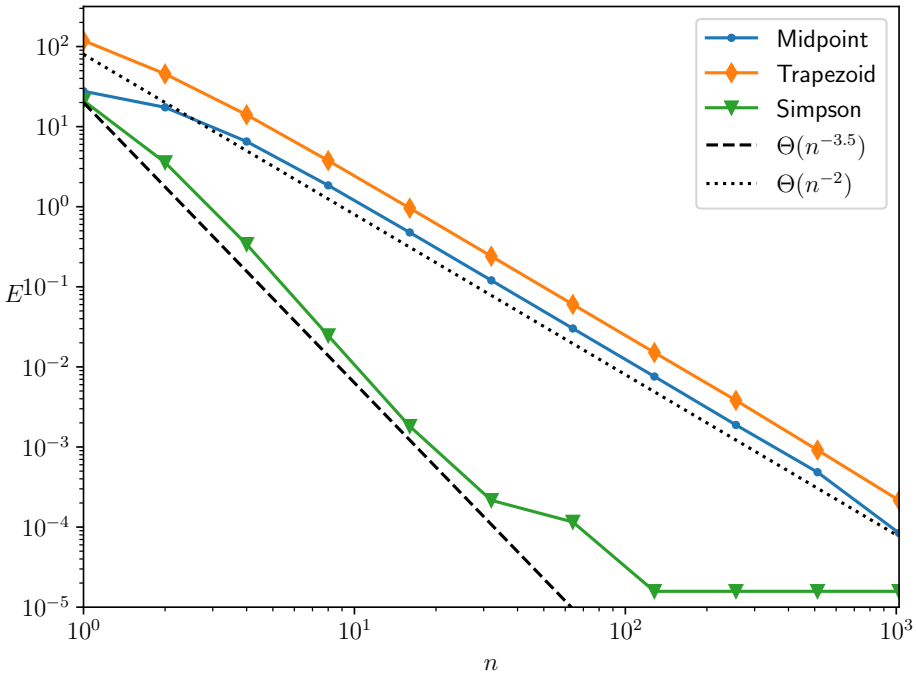

図 9　$f_5(x)$ の積分値とその理論値との誤差 (両対数グラフ)。

3.3.1

まず、$\tilde{u}(x) = 8x^3 - 12x$ が解であることを確認する。式 (3.1) にこれを代入して、

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

となり、$\tilde{u}(x) = 8x^3 - 12x$ が確かに解であることがわかる。

3.3.2

次に、$u = \tilde{u} + 4x$ を変数変換を行う。

$$\begin{aligned}
u = \tilde{u} + 4x \to \tilde{u} &= u - 4x \\
\therefore \frac{\mathrm{d}\tilde{u}}{\mathrm{d}x} &= \frac{\mathrm{d}u}{\mathrm{d}x} - 4
\end{aligned}$$

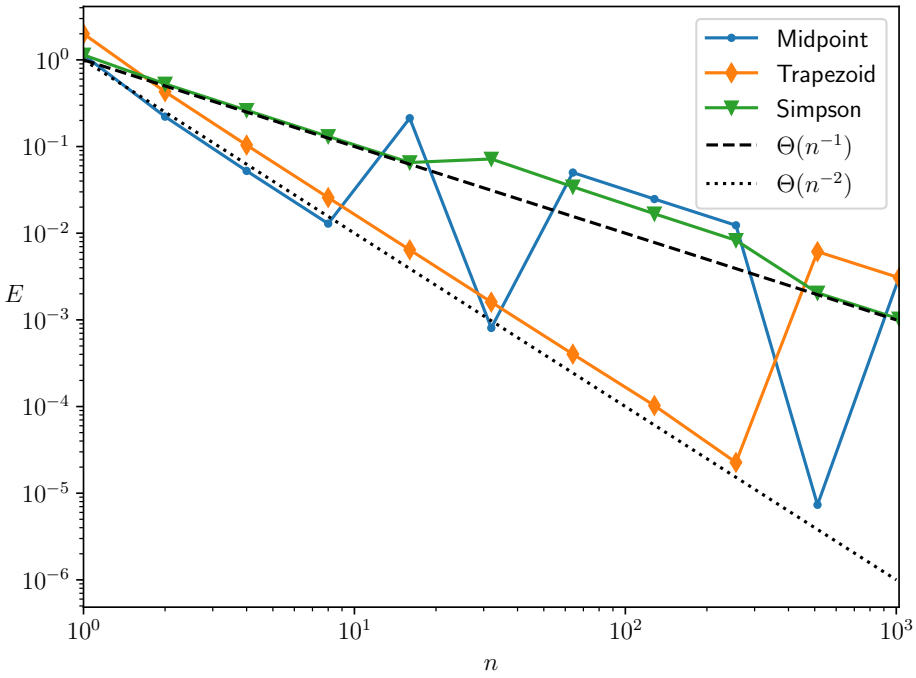

図 10　$f_6(x)$ の積分値とその理論値との誤差 (両対数グラフ)。

であるため、これらを式 (3.1) に代入して、

$$
\begin{aligned}
-\frac{\mathrm{d}}{\mathrm{d}x}\left(e^{-x^2}\frac{\mathrm{d}\tilde{u}}{\mathrm{d}x}\right) - 6e^{-x^2}\tilde{u} = 0 \\
\to -\frac{\mathrm{d}}{\mathrm{d}x}\left\{e^{-x^2}\left(\frac{\mathrm{d}u}{\mathrm{d}x} - 4\right)\right\} - 6e^{-x^2}(u - 4x) = 0 \\
\to -\frac{\mathrm{d}}{\mathrm{d}x}\left(e^{-x^2}\frac{\mathrm{d}u}{\mathrm{d}x}\right) - 6e^{-x^2}u = -16xe^{-x^2}
\end{aligned}
$$

を得る。これは $p(x) = e^{-x^2}, q(x) = -6e^{-x^2}, f(x) = -16xe^{-x^2}$ として、

$$
-\frac{\mathrm{d}}{\mathrm{d}x}\left(p(x)\frac{\mathrm{d}u}{\mathrm{d}x}\right) + q(x)u = f(x)\ (0 < x < 1), u(0) = u(1) = 1
$$

と書ける。

3.4

関数 $t_i(x)$ $(i = 1, 2, \cdots, n-1)$ を次のように定義する。

$$
t_i(x) = \begin{cases} \dfrac{x - x_{i-1}}{h} & (x_{i-1} \le x < x_i) \\ \dfrac{x_{i+1} - x}{h} & (x_i \le x < x_{i+1}) \\ 0 & (\text{その他}) \end{cases} \quad (h = x_i - x_{i-1})
$$

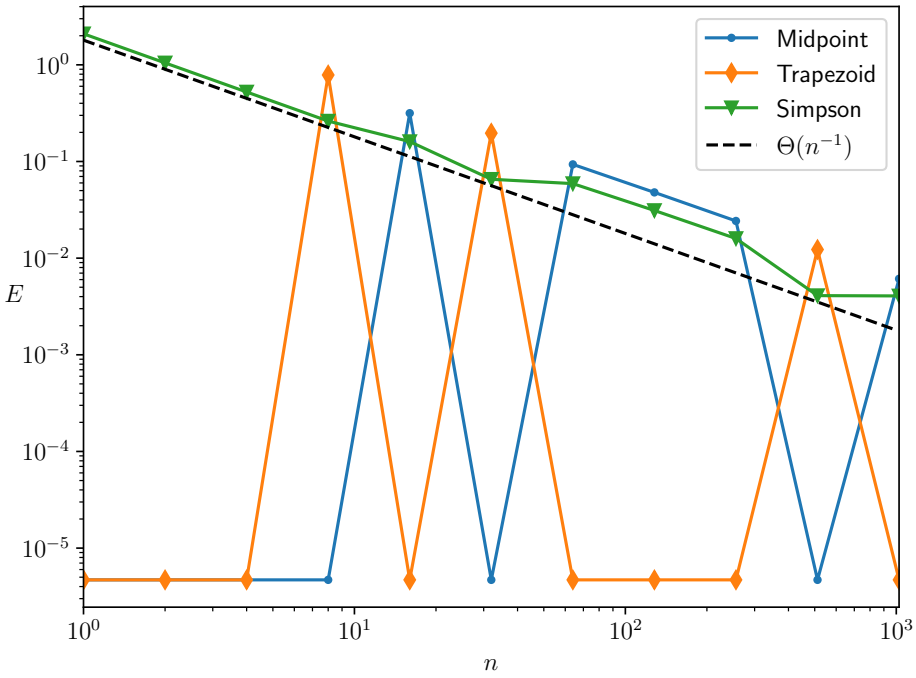

図 11　$f_7(x)$ の積分値とその理論値との誤差 (両対数グラフ)。

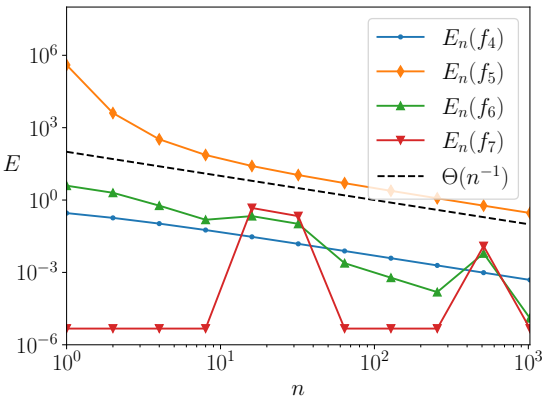

図 12　$M = 2$ の Gauss 型積分公式の誤差 (両対数グラフ)

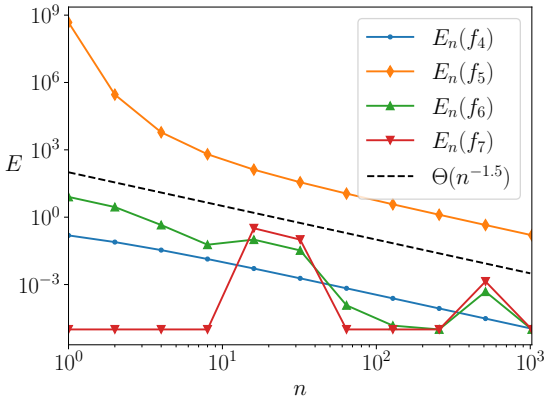

図 13　$M = 3$ の Gauss 型積分公式の誤差 (両対数グラフ)

これの傾き $\dfrac{\mathrm{d}t_i}{\mathrm{d}x}(x)$ は次のように書ける。

$$\frac{\mathrm{d}t_i}{\mathrm{d}x}(x) = \begin{cases} \dfrac{1}{h} & (x_{i-1} \le x < x_i) \\ -\dfrac{1}{h} & (x_i \le x < x_{i+1}) \\ 0 & (\text{その他}) \end{cases} \quad (h = x_i - x_{i-1})$$

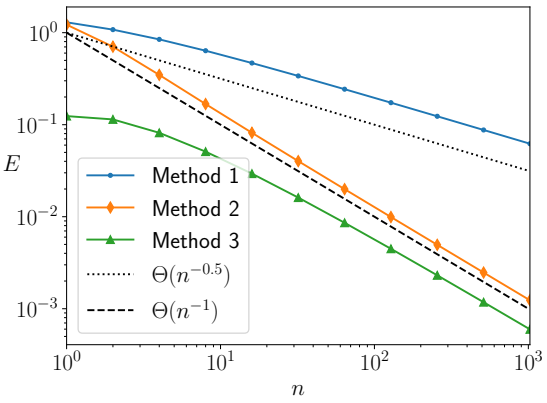

図 14　$M = 2$ の Gauss 型積分公式の誤差 (両対数グラフ)。Method1, Method2, Method3 はそれぞれの方法を用いた値を示す。

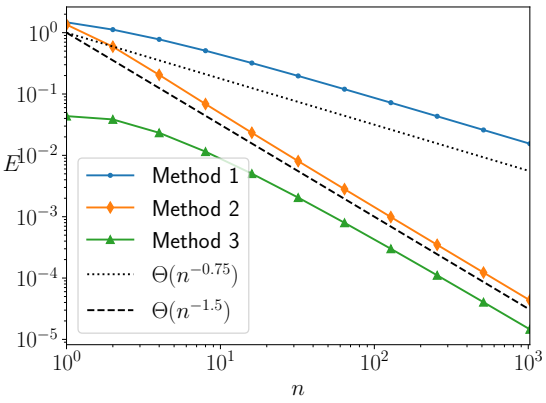

図 15　$M = 3$ の Gauss 型積分公式の誤差 (両対数グラフ)。Method1, Method2, Method3 はそれぞれの方法を用いた値を示す。

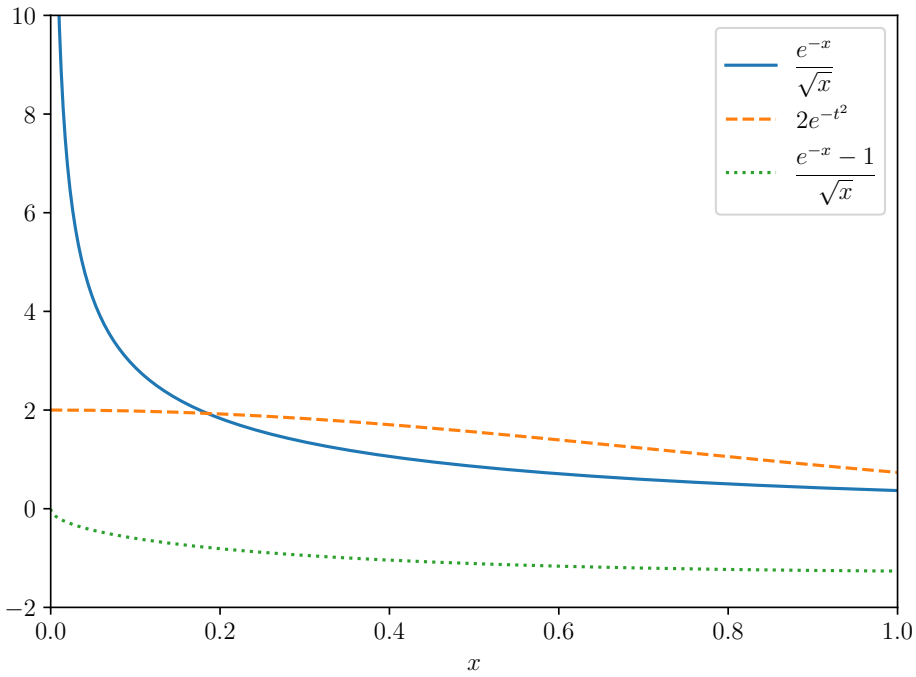

図 16　それぞれの方法の被積分関数

3.5

u を t_i の線型結合で $u(x) = \sum_{i=1}^{n-1} c_i t_i(x)$ と表すことを考える。ここで、$\boldsymbol{c} = \{c_i\}$ は以下に示される $A = \{A_{ij}\}, \boldsymbol{b} = \{b_i\}$ を用いて $A\boldsymbol{c} = \boldsymbol{b}$ と書ける。

$$A_{ij} = \int_0^1 \left\{ p(x) \frac{\mathrm{d}t_i}{\mathrm{d}x} \frac{\mathrm{d}t_j}{\mathrm{d}x} + q(x) t_i t_j \right\} \mathrm{d}x$$
$$b_i = \int_0^1 f(x) t_i \mathrm{d}x$$

ここで、i と j の差が 1 を超えるとき、上記の $t_i, \dfrac{\mathrm{d}t_i}{\mathrm{d}x}$ をみると、$\dfrac{\mathrm{d}t_i}{\mathrm{d}x} \dfrac{\mathrm{d}t_j}{\mathrm{d}x}$ 及び $t_i t_j$ はどちらも 0 となり、$A_{ij} = 0$ であることがわかる。よって、A は対称三重対角行列である。

また、一定の範囲を除いて $t_i, \dfrac{\mathrm{d}t_i}{\mathrm{d}x}$ が 0 であることから、A_{ij}と b_i を求める際、実際の積分範囲は $x \in [x_{i-1}, x_{i+1}]$ で良いことがわかる。

3.6

$M = 3$ の Gauss 型積分公式を用い、有限要素法のコードを作り u を数値計算で求める。なお、理論値は $u(x) = 8x(x^2 - 1)$ である。作成したコードをリスト 15, 16 に示す。

リスト 15　$p(x), q(x), f(x)$ の実装

```
#include <cmath>

const double p(const double x) {
    return exp(-x * x);
}

const double q(const double x) {
    return -6 * exp(-x * x);
}

const double f(const double x) {
    return -16 * x * exp(-x * x);
}
```

リスト 16　有限要素法の実装

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

#include "gauss.hpp"

class GaussJordan {
private:
    std::vector<std::vector<double>> A;
    std::vector<double> b;
    std::vector<double> x;

    const int get_pivot(const int i) {
        int pivot = i;
        for (int j = i + 1; j < A.size(); j++) {
            if (abs(A[pivot][i]) < abs(A[j][i])) {
                pivot = j;
            }
        }
```

```
20
21          return pivot;
22      }
23
24      const void up2down() {
25          for (int i = 0; i < this->A[0].size() - 1; i++) {
26              const int pivot = this->get_pivot(i);
27              std::iter_swap(this->A.begin() + i, this->A.begin() + pivot);
28              std::iter_swap(this->b.begin() + i, this->b.begin() + pivot);
29
30              for (int j = i + 1; j < this->A.size(); j++) {
31                  this->b[j] -= this->b[i] * this->A[j][i] / this->A[i][i];
32                  for (int k = this->A[0].size() - 1; k >= i; k--) {
33                      this->A[j][k] -= this->A[i][k] * this->A[j][i] / this->A[i][i];
34                  }
35              }
36          }
37      }
38
39      const void down2up() {
40          for (int i = this->A.size() - 1; i >= 0; i--) {
41              this->x[i] = this->b[i];
42              for (int j = i + 1; j < this->A[0].size(); j++) {
43                  this->x[i] -= this->A[i][j] * this->x[j];
44              }
45              this->x[i] /= this->A[i][i];
46          }
47      }
48
49  public:
50      GaussJordan(
51          const std::vector<std::vector<double>> A,
52          const std::vector<double> b
53      ) : A(A), b(b), x(b.size()) {
54          this->up2down();
55          this->down2up();
56      }
57
58      const std::vector<double> get_x() {
59          return this->x;
60      }
61  };
62
63  class TGenerator {
64  public:
65  const double x_previous, x_now, x_next, h;
66      TGenerator(
67          const double x_previous,
68          const double x_now,
69          const double x_next,
70          const double h
71      ) : x_previous(x_previous), x_now(x_now), x_next(x_next), h(h) {
72      }
73
74      const double t(const double x) const {
75          if (this->x_previous <= x && x < this->x_now) {
76              return (x - this->x_previous) / this->h;
```

```
77          } else if (this->x_now <= x && x < x_next) {
78              return (this->x_next - x) / this->h;
79          } else {
80              return 0;
81          }
82      }
83
84      const double dt(const double x) const {
85          if (this->x_previous <= x && x < this->x_now) {
86              return 1.0 / this->h;
87          } else if (this->x_now <= x && x < x_next) {
88              return -1.0 / this->h;
89          } else {
90              return 0;
91          }
92      }
93  };
94
95  class FEM {
96  private:
97      const double start, end;
98      const int n;
99      const int M = 3;
100     const std::vector<double> y{-sqrt(3.0 / 5.0), 0, sqrt(3.0 / 5.0)};
101     const std::vector<double> w{5.0 / 9.0, 8.0 / 9.0, 5.0 / 9.0};
102
103     std::vector<TGenerator> t;
104     std::vector<double> c;
105
106     void calculate_t(
107         const double start,
108         const double end,
109         const int n
110     ) {
111         const double h = (end - start) / (double)n;
112         double x = start + h;
113         for (int i = 0; i < n - 1; i++, x += h) {
114             this->t.push_back(TGenerator(x - h, x, x + h, h));
115         }
116     }
117
118     const std::vector<std::vector<double>> calculate_A(
119         const double start,
120         const double end,
121         const int n,
122         const double (*p)(const double),
123         const double (*q)(const double)
124     ) {
125         std::vector<std::vector<double>> A(n - 1, std::vector<double>(n - 1));
126         for (int i = 0; i < n - 1; i++) {
127             for (int j = 0; j < n - 1; j++) {
128                 std::function<double(double)> f = [&](const double x) {
129                     return p(x) * t[i].dt(x) * t[j].dt(x) + q(x) * t[i].t(x) * t[j].t(x);
130                 };
131                 A[i][j] = gauss(f, start, end, 1000, M, y, w);
132             }
133         }
```

```
134
135         return A;
136     }
137
138     const std::vector<double> calculate_b(
139         const double start,
140         const double end,
141         const int n,
142         const double (*f)(const double)
143     ) {
144         std::vector<double> b(n - 1);
145         for (int i = 0; i < n - 1; i++) {
146             const std::function<double(double)> ff = [&](const double x) {
147                 return f(x) * t[i].t(x);
148             };
149
150             b[i] = gauss(ff, start, end, 1000, M, y, w);
151         }
152
153         return b;
154     }
155
156 public:
157     FEM(
158         const double start,
159         const double end,
160         const int n,
161         const double (*p)(const double),
162         const double (*q)(const double),
163         const double (*f)(const double)
164     ) : start(start), end(end), n(n) {
165         this->calculate_t(start, end, n);
166
167         const std::vector<std::vector<double>> A = this->calculate_A(start, end, n, p, q);
168
169         const std::vector<double> b = this->calculate_b(start, end, n, f);
170         this->c = GaussJordan(A, b).get_x();
171     }
172
173     const std::vector<double> get_u(const std::vector<double> x) const {
174         std::vector<double> u;
175         const double h = (this->end - this->start) / this->n;
176         for (const double xx: x) {
177             double uu = 0;
178             for (int j = 0; j < this->n - 1; j++) {
179                 uu += this->c[j] * this->t[j].t(xx);
180             }
181             u.push_back(uu);
182         }
183
184         return u;
185     }
186 };
```

■結果・考察　計算結果を図 17 に示す。

ある程度大きい分割数 n では高い精度で計算ができていることがわかる。分割数 n における計算値 u_n と理

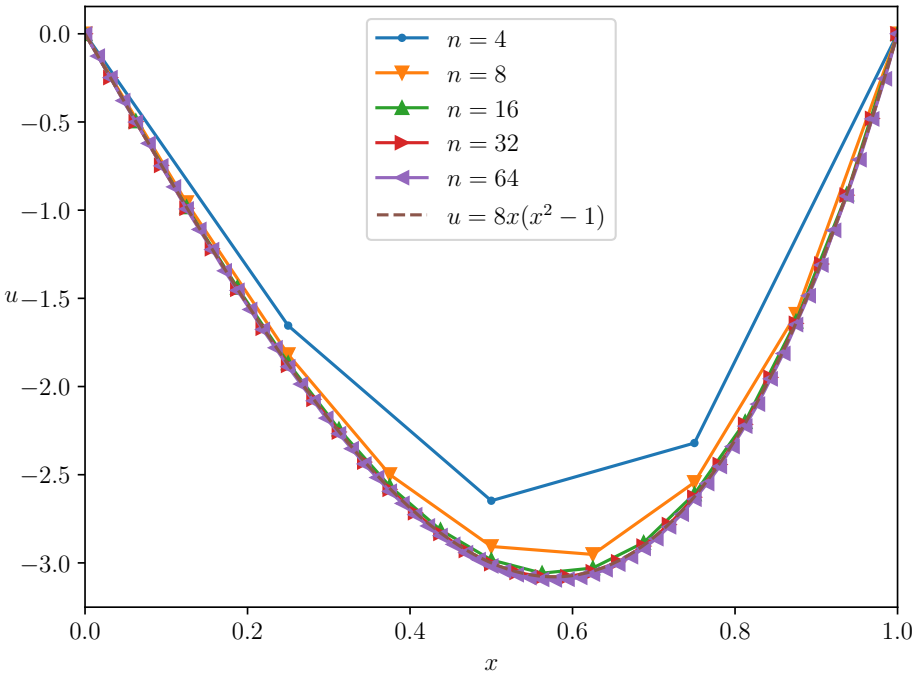

図 17　u の計算結果。n は分割数。

論値 u の平均誤差を次のように定義する。

$$E(n) = \frac{1}{n+1} \sum_{i=0}^{n} |u_n(x_i) - u(x_i)|$$

これをプロットしたものを図 18 に示す。図を見ると、$n = 32$ のときに誤差が最小となり、それ以前は誤差は n^{-2} に比例して減少し、それ以後は $n^{1.2}$ に比例して増加することがわかった。

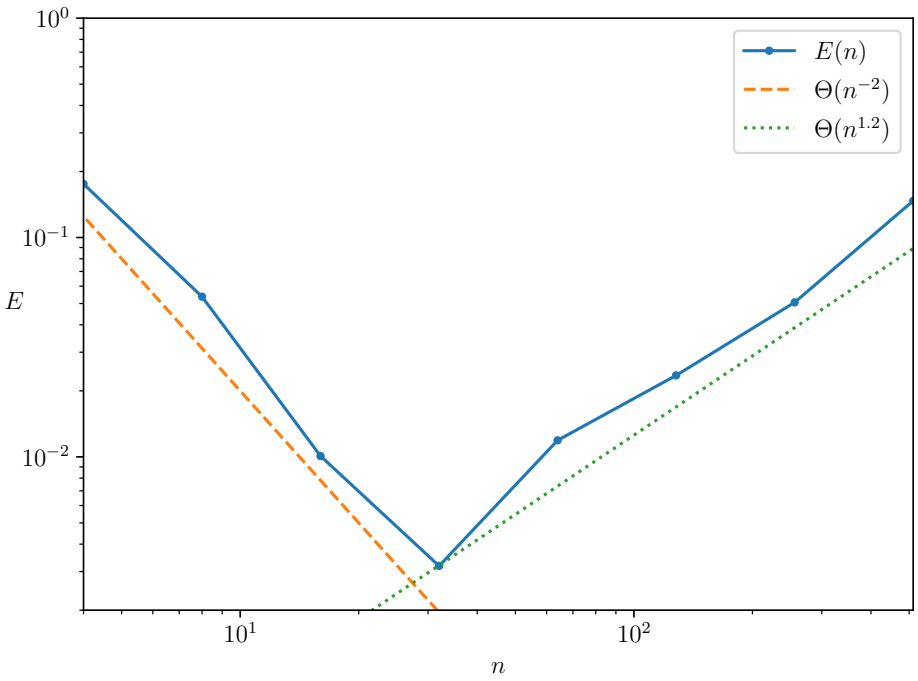

図 18　計算値と理論値の誤差

参考文献

[1] 実験演習ワーキンググループ、“数理工学実験 2022 年度版”、京都大学工学部情報学科数理工学コース (2022)