数理工学実験
テーマ４　数値積分

2021年11月29日 提出

工学部情報学科数理工学コース２年
1029-32-7314　岡本淳志

1　はじめに

被積分関数を積分計算が容易な既知関数で補間することで積分値を近似的に求める、関数補間に基づく数値積分についての実験を行う。

今回の実験では、

- Newton-Cotes 公式：被積分関数を Lagrange 補間（多項式補間）することで積分値を計算する方法
- Gauss 型積分公式：被積分関数を直交多項式で補間することで積分値を計算する方法

の 2 つの公式を用いる。

Gauss 型積分公式は積分区間の分割を工夫している点で、Newton-Cotes 積分公式よりも精度が高い数値積分法である。

1.1　Lagrange 補間

相異なる n 個の点 $x_1 < ... < x_n$ において関数 f の値 $f(x_i),\ (i = 1, ..., n)$ が与えられているとする。ここから

$$P(x_i) = f(x_i) \quad (i = 1, ..., n) \tag{1}$$

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 $P(x)$ と推定すること考える。（このことを多項式補間または Lagrange 補間と呼ぶ。）

式（1）を満たす $n-1$ 次の補間多項式 $P(x)$ は

$$P(x) = \sum_{i=1}^{n} f(x_i) l_i(x) \tag{2}$$

$$l_i(x) = \frac{\prod_{j \neq i}(x - x_j)}{\prod_{j \neq i}(x_i - x_j)} \tag{3}$$

で与えられる。

1.2　Newton-Cotes 公式

定積分

$$I(f) = \int_a^b f(x) dx \tag{4}$$

を近似的に計算するために、Lagrange 補間で関数 f を近似する方法を考える。x_i を n 個の等間隔分点

$$x_i = x_1 + (i-1)h \quad (1 \leq i \leq n, h > 0) \tag{5}$$

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

$$\begin{aligned} I(f) &= \int_a^b f(x)dx \\ &\approx \int_a^b P(x)dx \\ &= \int_a^b \sum_{i=1}^{n} f(x_i)l_i(x) \end{aligned} \tag{6}$$

となる。（この数値積分公式を n 点 Newton-Cotes 公式という）

主な等間隔分点 x_i の個数およびそのとり方は、以下の 3 種類の公式である。

- 中点公式：
  区間 (a, b) において、$x_1 = \frac{a+b}{2}$ の 1 点 Newton-Cotes 公式
- 台形公式：
  区間 (a, b) において、$x_1 = a$, $x_2 = b$ の 2 点 Newton-Cotes 公式
- Simpson の公式：
  区間 (a, b) において、$x_1 = a$, $x_2 = \frac{a+b}{2}$, $x_3 = b$ の 3 点 Newton-Cotes 公式

実際の計算では、積分区間 $[a, b]$ を等分して小区間を作り、各区間に対してこの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、この 3 つの積分公式の複合公式をそれぞれ、複合中点公式、複合台形公式、複合 Simpson 公式と呼ぶ。以下にこれらの公式の形を示す。

$h = \frac{b-a}{n}$, $x_i = a + ih$, $(0 \leq i \leq n)$ とおき、積分区間 (a, b) を n 個の小区間 (x_i, x_{i+1}) $(0 \leq i \leq n-1)$ に分割し、各々の小区間に対してこれらの公式を適用すると、

- 複合中点公式：

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

- 複合台形公式：

$$\int_a^b f(x)dx \approx \frac{h}{2}\left(f(x_0) + 2\sum_{i=1}^{n-1} f(x_i) + f(x_n)\right) \tag{8}$$

- 複合 Simpson 公式：

$$\int_a^b f(x)dx \approx \frac{h}{6}\left(f(x_0) + 2\sum_{i=1}^{n-1} f(x_i) + f(x_n) + 4\sum_{i=0}^{n-1} f\left(\frac{x_i + x_{i+1}}{2}\right)\right) \tag{9}$$

となる。

1.3 直行多項式

$w(x) > 0$ を区間 (a, b) で定義された連続な関数とし、実係数多項式 $p(x)$,$q(x)$ に対して

$$(p, q)_w = \int_a^b w(x)p(x)q(x)dx \tag{10}$$

とおく。$(\ ,\ )_w$ は内積を与えるので、以下の性質を満たす。

- 正値性　$(p, p)_w \geq 0$ で、$(p, p)_w = 0 \Rightarrow p \equiv 0$
- 対称性　$(p, q)_w = (q, p)_w$
- 線形性　多項式 $p(x)$,$q(x)$,$r(x)$ と実数 α, β に対して $(\alpha p + \beta q, r)_w = \alpha(p, r)_w + \beta(q, r)_w$

とくに $(p, q)_w = 0$ のとき、多項式 p, q は重み関数 w に関して直交するという。さらに $\phi_n(x),\ (n = 0, 1, 2, ...)$ が n 次多項式 $(\phi_0(x) \neq 0)$ で $(\phi_m, \phi_n)_w = 0,\ (m \neq n)$ を満たすとき、ϕ_n を直交多項式系といい、以下の性質を満たす。

- 直交多項式系の存在　任意の内積に対して直交多項式系は存在する。
- 直交性　$\phi_n(x)$ は $n - 1$ 次以下の任意の多項式と直交する。
- 一意性　二つの直交多項式系 $\{\phi_n(x)\}$,$\{\widetilde{\phi}_n(x)\}$ に対して、ある定数 $c_n \neq 0$ が存在して、$\phi_n(x) = c_n\widetilde{\phi}_n(x)$。すなわち直交多項式系は定数倍を除いて一意に定まる。

これより、内積（すなわち重み関数 w と区間 (a, b)）を一つ定めると直交多項式系が一つ定まる。

今回は、

$$(p, q) = \int_{-1}^1 p(x)q(x)dx \tag{11}$$

のように、内積（10）の重み関数 $w \equiv 1$、区間 $(a, b) = (-1, 1)$ としたものを考える。（このような直交多項式系 $P_n(x)$ を Legendre 多項式と呼ぶ。）

1.4 Gauss 型積分公式

n 個の重み w_i と分点 $x_i,\ (i = 1, ..., n)$ に対して積分公式

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

を考える。これが $2n - 1$ 次（最大）以下の多項式に対して正確にするには、分点は直交多項式の零点、重みは

$$w_i = \int_{-1}^1 l_i(x)dx \tag{13}$$

で与えればよい。n 次の Legendre 多項式の零点 x_i と重み（13）による積分公式（12）を n 次の Gauss 型積分公式または Gauss-Legendre 積分公式と呼ぶ。

Gauss 型積分公式も Newton-Cotes 公式と同様に複合公式を使うのが一般的である。区間 (a,b) における積分を分点 $x_i = a + \frac{i(b-a)}{N}$ $(i = 0, ..., N)$ で N 等分すると

$$\int_a^b f(x)dx = \sum_{i=0}^{N-1} \int_{x_i}^{x_{i+1}} f(x)dx \tag{14}$$

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 型積分公式の分点と重みをそれぞれ y_m, w_m $(m = 1, ..., n)$ とすると、

$$\begin{aligned}
\int_a^b f(x)dx &= \sum_{i=0}^{N-1} \int_{x_i}^{x_{i+1}} f(x)dx \\
&= \sum_{i=0}^{N-1} \int_{-1}^{1} f\left\{ \frac{(y+1)(x_{i+1}-x_i)}{2} + x_i \right\} \frac{x_{i+1}-x_i}{2} dy \qquad (15) \\
&\approx \sum_{i=0}^{N-1} \sum_{m=1}^{n} w_m f\left\{ \frac{(y_m+1)(x_{i+1}-x_i)}{2} + x_i \right\} \frac{x_{i+1}-x_i}{2}
\end{aligned}$$

となる。これが、Gauss 型積分公式に対する複合公式である。

2　4.2 節の課題

2.1　課題 1

$n = 2, 4, , 8, 16$ に対して、

$$\begin{aligned} &(a) \quad f(x) = \log x \qquad x \in (1, 2) \\ &(b) \quad f(x) = \frac{1}{x^3} \qquad x \in (0.1, 2) \\ &(c) \quad f(x) = \frac{1}{1 + 25x^2} \qquad x \in (-1, 1) \end{aligned} \tag{16}$$

の $n-1$ 次多項式による Lagrange 補間のグラフを図示する。ただし、関数が定義される区間を (a, b) としたとき、分点 x_i, $(i = 1, ..., n)$ を

$$x_i = a + \frac{b-a}{n-1}(i-1) \tag{17}$$

とする。

2.1.1　実験方法

x_i よりもさらに細かい分点 y_i を取り、$P(y_i)$ を計算するコードを書いた。(今回は x_i より 10 倍細かく y_i を取った。)

(a), (b), (c) の Lagrange 補間のグラフを図示するためのコードを、それぞれ付録のソースコード 1, 2, 3 に示す。

プログラムの出力をもとに、gnuplot を用いて図の作成を行う。

2.1.2　実験結果

出力に基づき、横軸を x、縦軸を $P(x)$ とした図を作成した。(a), (b), (c) の Lagrange 補間のグラフを、それぞれ以下の図 1, 2, 3 に示す。

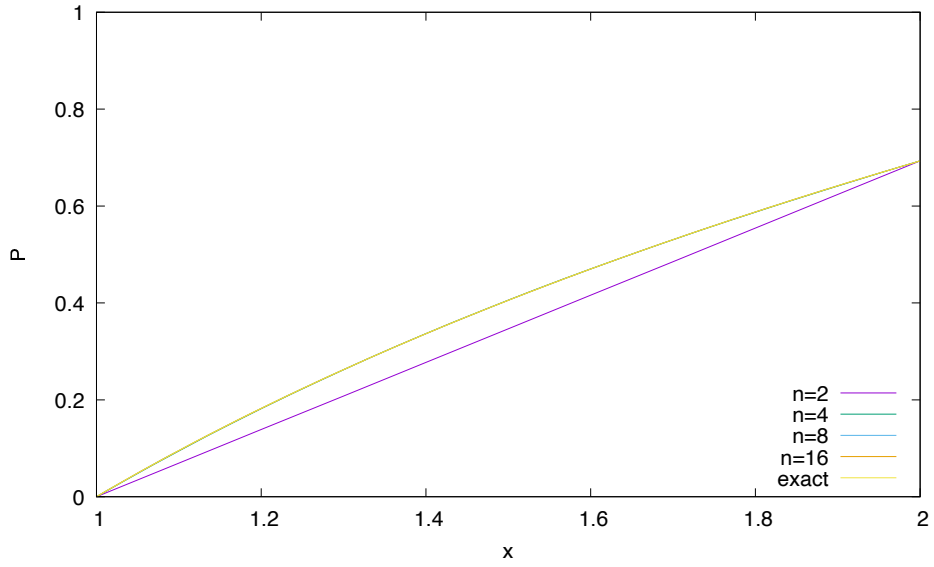

図 1: (a) の Lagrange 補間のグラフ, 横軸の範囲は x の定義域, 縦軸の範囲は $0 \leq P(x) \leq 1$

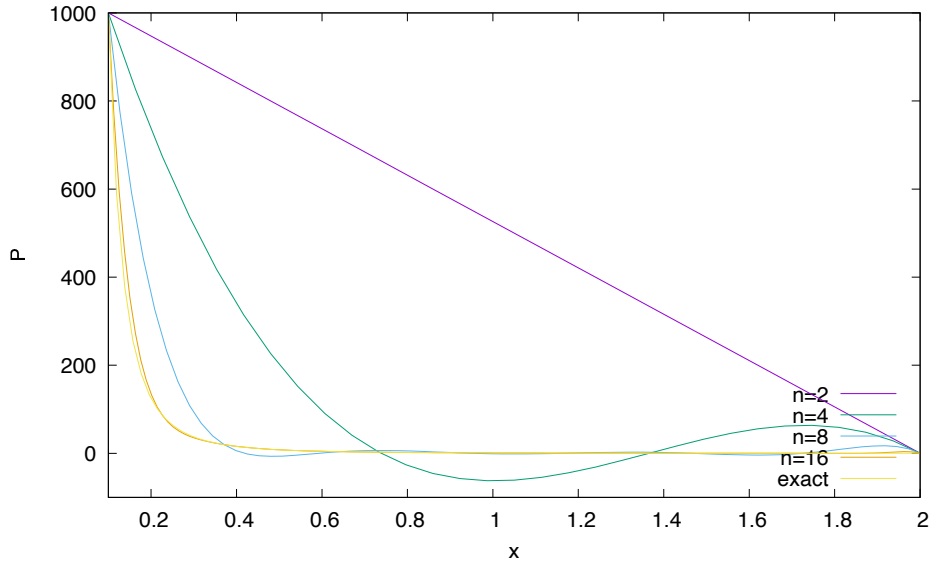

図 2: (b) の Lagrange 補間のグラフ, 横軸の範囲は x の定義域, 縦軸の範囲は $-100 \leq P(x) \leq 1000$

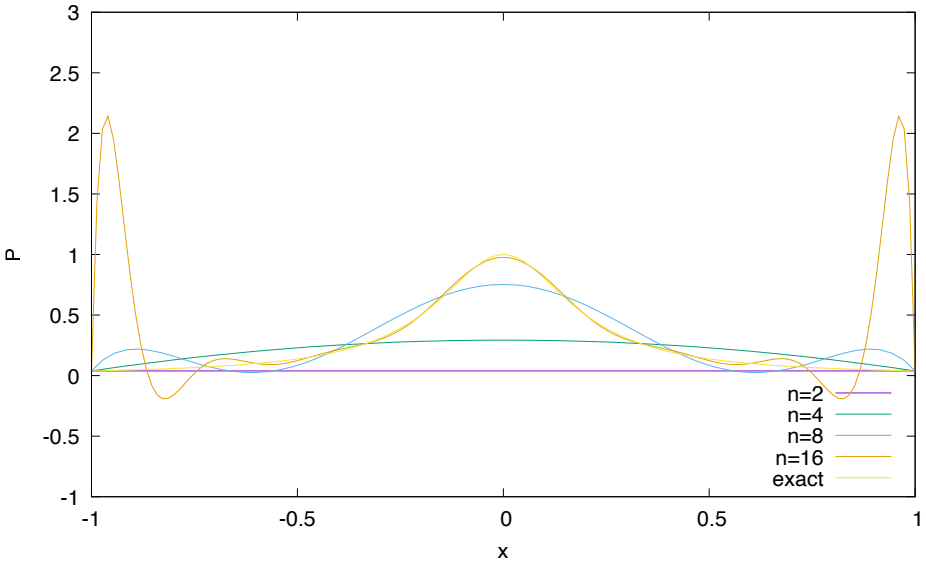

図 3: (c) の Lagrange 補間のグラフ, 横軸の範囲は x の定義域, 縦軸の範囲は $-1 \leq P(x) \leq 3$

2.1.3 考察

図 2, 3 では、n の値を大きくしていけば、近似の精度が上がっていく様子が顕著に見られた。また、x 軸方向に等間隔で、$f(x)$ と $P(x)$ が n 回交わっていることから、理論通り $P(x_i) = f(x_i)$ となっていると考えられる。(実際に、出力からもこのことが分かった。)

対して図 1 では、$n = 4, 8, 16$ のとき、多項式で近似した関数がほとんど正確な軌道を描いていることが分かる。このことは、$f(x) = \log x$ が単調（増加）な関数であることや、$f'(x) = \frac{1}{x}$, $f''(x) = -\frac{1}{x^2}$,... から、$\log x$ をテイラー展開した形が $f(x) = a_0 + \frac{a_1}{x} + \frac{a_2}{x^2}$... と書けることが理由ではないかと考えた。また、この図ではわかりにくいが、プログラムの出力で $P(x_i) = f(x_i)$ となっているので、「x 軸方向に等間隔で、$f(x)$ と $P(x)$ が n 回交わっている」ことが分かる。

2.2 課題 2, 3, 4

区間 (a, b) における定積分

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

$$\begin{aligned}
&(a) \quad f(x) = \frac{1}{x} \qquad x \in (1, 2) \\
&(b) \quad f(x) = e^{5x} \qquad x \in (-1, 1) \\
&(c) \quad f(x) = 1 + \sin x \qquad x \in (0, \pi) \\
&(d) \quad f(x) = 1 + \sin x \qquad x \in (0, 2\pi)
\end{aligned} \tag{19}$$

を分割数 $n = 2^i (i = 0, 1, ..., 10)$ の複合中点公式、複合台形公式、複合 Simpson 公式により求める。

2.2.1 実験方法

(a), (b), (c), (d) を複合中点公式（7）、複合台形公式（8）、複合 Simpson 公式（9）を用いて数値積分するためのコードを、それぞれ付録のソースコード 4, 5, 6, 7 に示す。

プログラムの出力をもとに、gnuplot を用いて図の作成を行う。

2.2.2 実験結果（課題 2）

出力に基づき、i を横軸に、$I(f)$ を数値積分した値を縦軸にとり、図を作成した。(a), (b), (c), (d) の数値積分をもとに作成した図を、それぞれ以下の図 4, 5, 6, 7 に示す。

ただし、図中の Midpoint, Trapezoid, Simpson は、それぞれ複合中点公式、複合台形公式、複合 Simpson 公式により得たグラフを指す。

また、exact が示す値は解析的に求めたものを用いた。(a) の場合は $I(f) = \log 2$、(b) の場合は $I(f) = \frac{1}{5}(\mathrm{e}^5 - \mathrm{e}^{-5})$、$(c)$ の場合は $I(f) = 2 + \pi$、(d) の場合は $I(f) = 2\pi$ である。

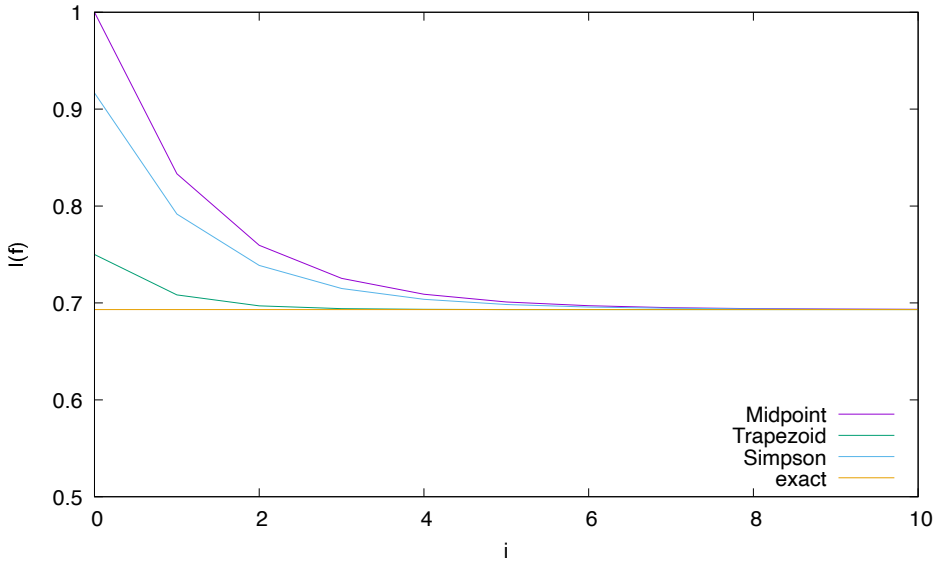

図 4: i を横軸に、(a) の $I(f)$ を数値積分した値を縦軸にとり、作成したグラフ, 横軸の範囲は $0 \leq i \leq 10$, 縦軸の範囲は $0.5 \leq I(f) \leq 1$

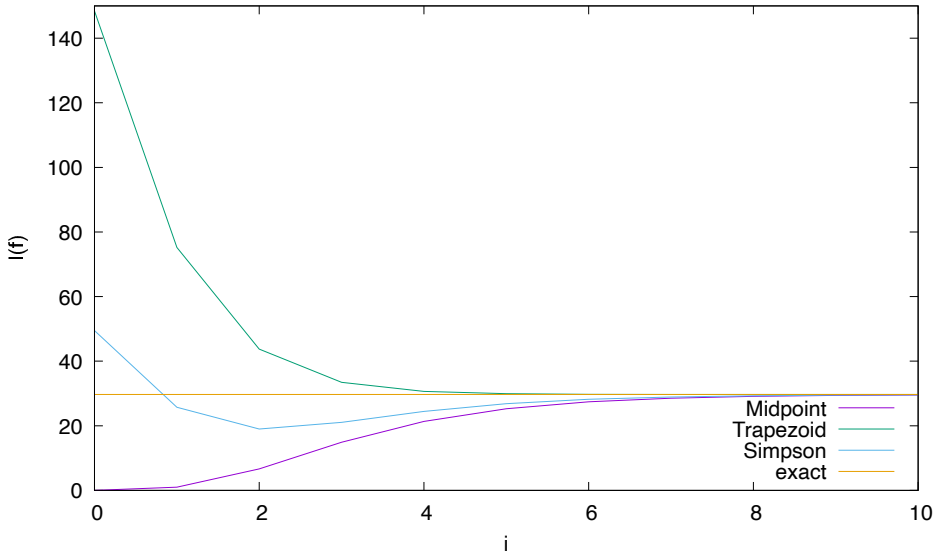

図 5: i を横軸に、(b) の $I(f)$ を数値積分した値を縦軸にとり、作成したグラフ, 横軸の範囲は $0 \leq i \leq 10$, 縦軸の範囲は $0 \leq I(f) \leq 150$

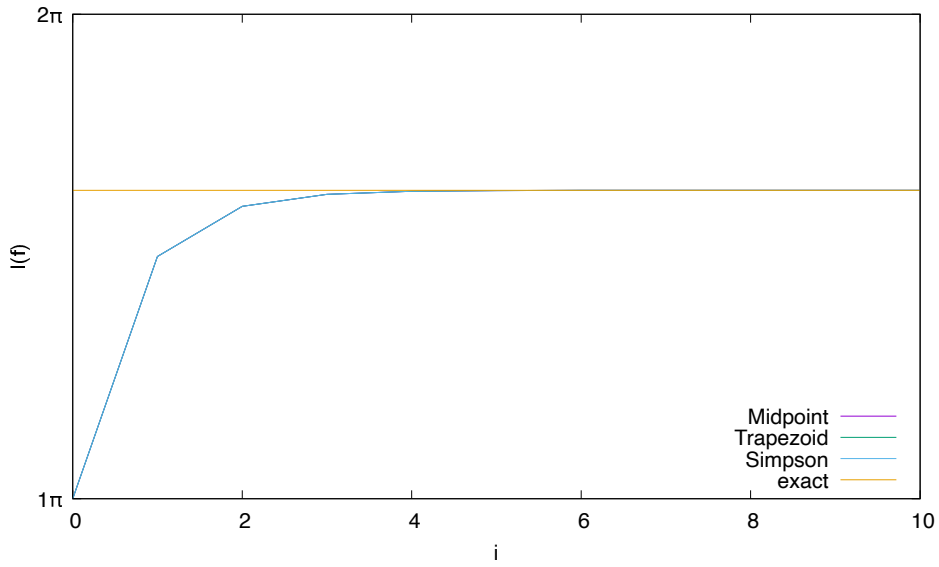

図 6: i を横軸に、(c) の $I(f)$ を数値積分した値を縦軸にとり、作成したグラフ, 横軸の範囲は $0 \leq i \leq 10$, 縦軸の範囲は $\pi \leq I(f) \leq 2\pi$, 図からは不明瞭だが、このグラフ作成のもとにした出力より、Midpoint, Trapezoid, Simpson が描く軌道は完全に一致している

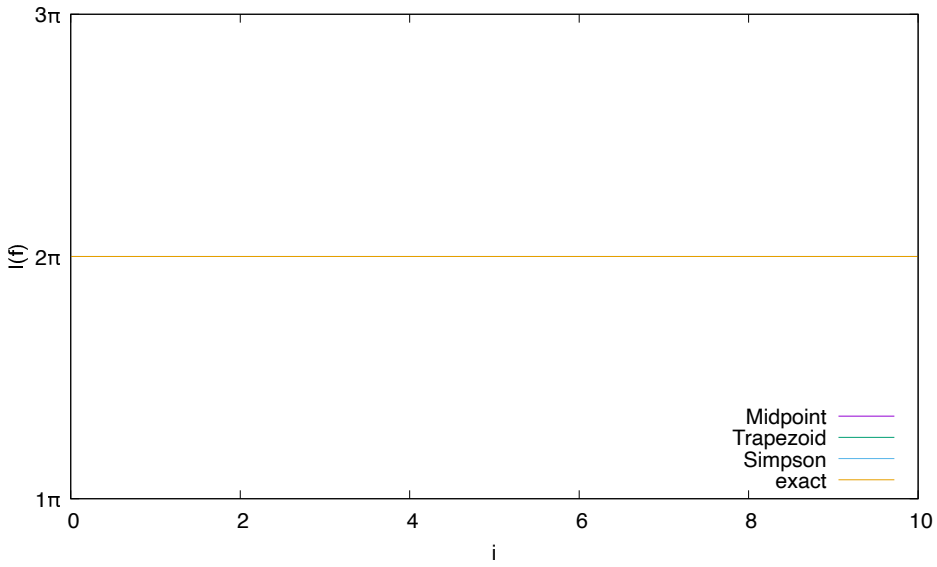

図 7: i を横軸に、(d) の $I(f)$ を数値積分した値を縦軸にとり、作成したグラフ, 横軸の範囲は $0 \leq i \leq 10$, 縦軸の範囲は $\pi \leq I(f) \leq 3\pi$, 図からは不明瞭だが、このグラフ作成のもとにした出力より、Midpoint, Trapezoid, Simpson は全て 2π の値を取っている

2.2.3　実験結果（課題 3）

出力に基づき、i を横軸に、$E_n(f)$ を縦軸にとり、図を作成した。(a), (b), (c), (d) の場合の図を、それぞれ以下の図 8, 9, 10, 11 に示す。

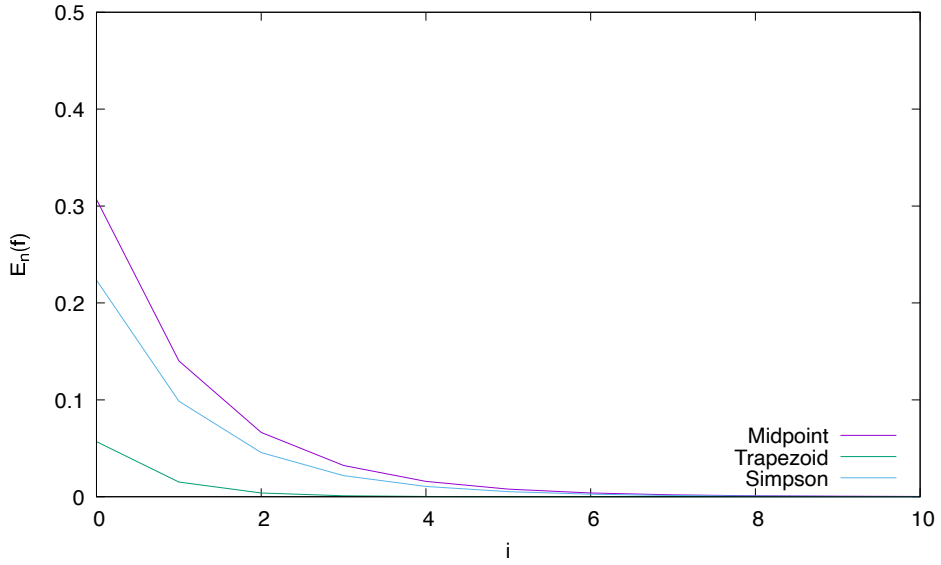

図 8: i を横軸に、(a) の $E_n(f)$ を縦軸にとり、作成したグラフ, 横軸の範囲は $0 \leq i \leq 10$, 縦軸の範囲は $0 \leq E_n(f) \leq 0.5$

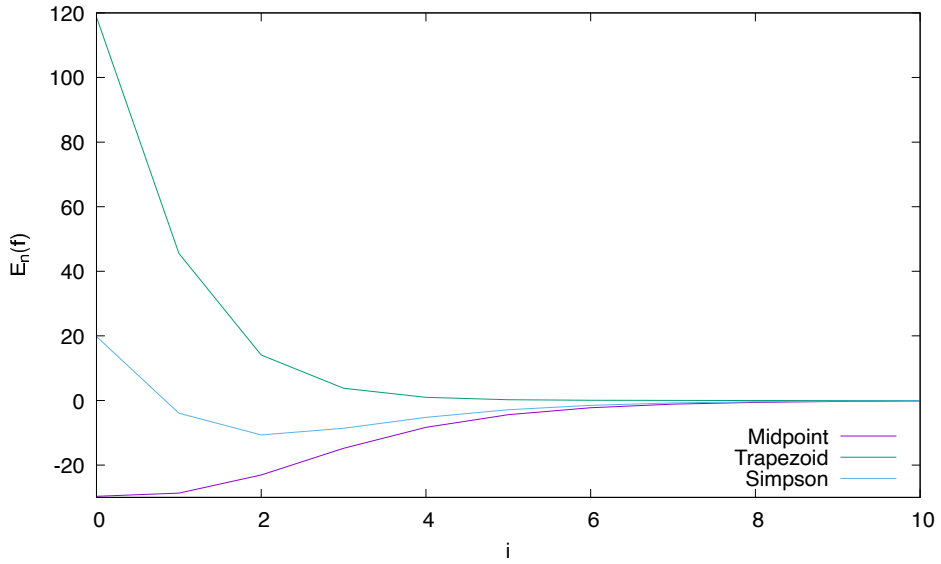

図 9: i を横軸に、(b) の $E_n(f)$ を縦軸にとり、作成したグラフ, 横軸の範囲は $0 \leq i \leq 10$, 縦軸の範囲は $-30 \leq E_n(f) \leq 120$

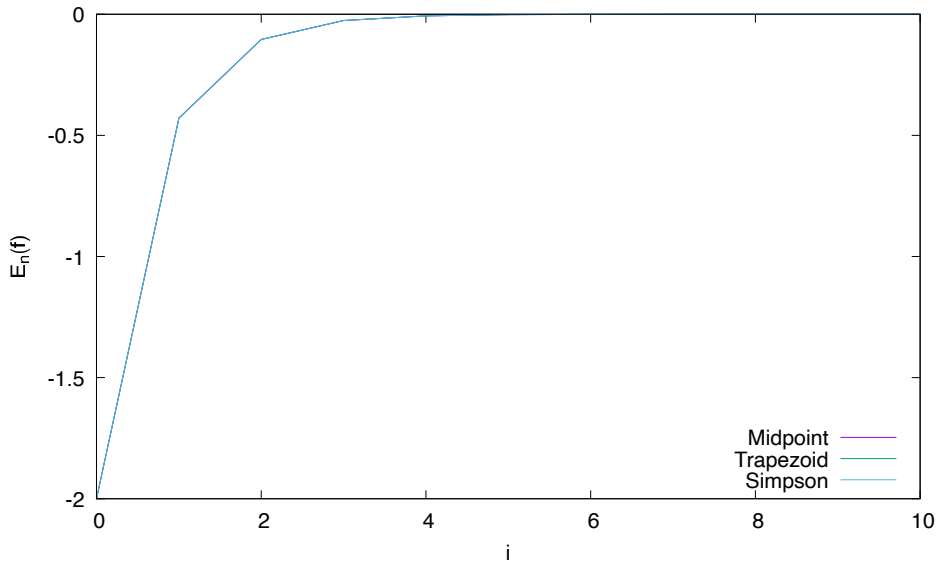

図 10: i を横軸に、(c) の $E_n(f)$ を縦軸にとり、作成したグラフ, 横軸の範囲は $0 \leq i \leq 10$, 縦軸の範囲は $-2 \leq E_n(f) \leq 0$, 図からは不明瞭だが、このグラフ作成のもとにした出力より、Midpoint, Trapezoid, Simpson が描く軌道は完全に一致している

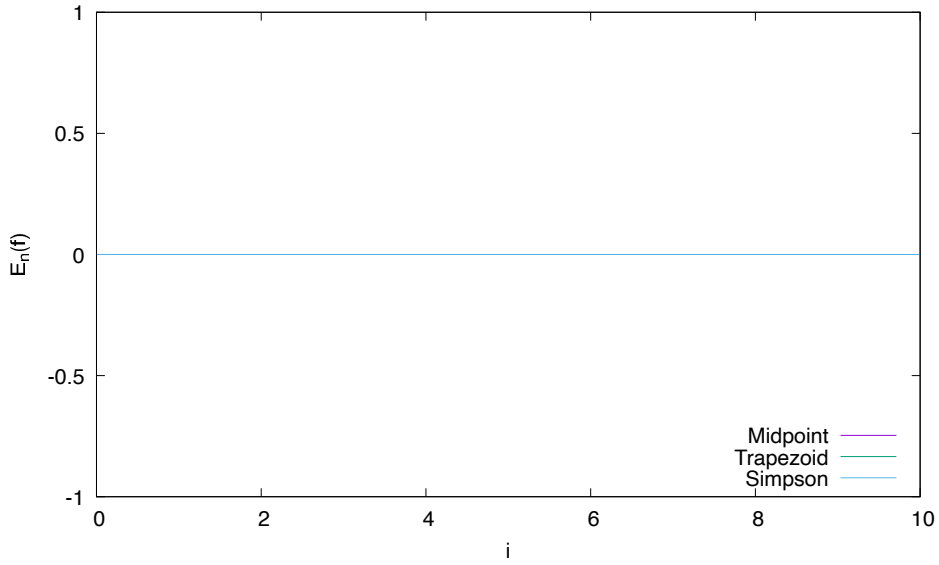

図 11: i を横軸に、(d) の $E_n(f)$ を縦軸にとり、作成したグラフ, 横軸の範囲は $0 \leq i \leq 10$, 縦軸の範囲は $-1 \leq E_n(f) \leq 1$, 図からは不明瞭だが、このグラフ作成のもとにした出力より、Midpoint, Trapezoid, Simpson は全て 0 の値を取っている

2.2.4 考察（課題 4）

図 4, 5, 6(, 7) と図 8, 9, 10(, 11) より、i の値を大きくする、すなわち分割数 $n(=2^i)$ の値を大きくすれば、いずれの複合公式を用いた場合でも数値積分の精度が高まり、誤差 $E_n(f)$ が 0 に近づいていくことが分かる。

図 8, 9 を見比べると分かるように、少なくとも分割数 $n(=2^i)$ の値が小さい場合では、「より多くの分点を用いて数値積分を行う Simpson 公式の方が、精度が高く誤差 $E_n(f)$ の値が小さくなる」という訳ではないことが分かった。

図 8, 9, 10, 11 を見て分かるように、被積分関数によって、より精度が高くなるような複合公式や誤差に大きな違いがあることが分かる。また、図 10, 11 を見て分かるように、被積分関数によって、いずれの複合公式を用いても数値積分の値が同じになるものや、誤差が（ほぼ）ないものがあることが分かった。(c)　$f(x) = 1 + \sin x$ $x \in (0, \pi)$ の関数を数値積分した際に、いずれの複合公式を用いても数値積分の値が同じになるのは、Newton-Cotes 公式が分点を等間隔にとることと、この関数を $x-y$ 平面に描いた時、$x = \frac{\pi}{2}$ で線対称な軌道を描くことが理由ではないかと考えた。(d)　$f(x) = 1 + \sin x$ $x \in (0, 2\pi)$ の関数を数値積分した際に、分割数 $n(=2^i)$ の値が小さい場合でも誤差がないのは、この関数を $x-y$ 平面に描いた時、$(\pi, 1)$ で点対称な軌道を描くことが理由ではないかと考えた。これらのことは、$x \in (0, 2\pi)$ の単一区間で中点

公式、台形公式、Simpson の式を用いて数値積分することを考えると理解しやすい。

3　4.3 節の課題

3.1　課題 1

4.2 節の課題 2 と同じ積分

区間 (a, b) における定積分

$$I(f) = \int_a^b f(x)dx \tag{20}$$

$$\begin{aligned}
&(a) \quad f(x) = \frac{1}{x} \qquad x \in (1, 2) \\
&(b) \quad f(x) = e^{5x} \qquad x \in (-1, 1) \\
&(c) \quad f(x) = 1 + \sin x \qquad x \in (0, \pi) \\
&(d) \quad f(x) = 1 + \sin x \qquad x \in (0, 2\pi)
\end{aligned} \tag{21}$$

を分割数 $N = 2^i$ $(i = 0, 1, ..., 10)$ の複合 $M(= 2, 3)$ 次 Gauss 型積分公式（15）で計算し、結果を図示する。ただし、積分区間を N 分割した確証区間では表 1,2 に従う。

表 1: $M = 2$ の Gauss 型積分公式

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

表 2: $M = 3$ の Gauss 型積分公式

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

3.1.1　実験方法

(a), (b), (c), (d) を複合 Gauss 型積分公式（15）を用いて数値積分するためのコードを、それぞれ付録のソースコード 8, 9, 10, 11 に示す。

プログラムの出力をもとに、gnuplot を用いて図の作成を行う。

3.1.2 実験結果

出力に基づき、i を横軸に、$I(f)$ を数値積分した値を縦軸にとり、図を作成した。(a), (b), (c), (d) の数値積分をもとに作成した図を、それぞれ以下の図 12, 13, 14, 15 に示す。

また、exact が示す値は解析的に求めたものを用いた。（値は 4.2 節の課題 2 の実験結果を参照）

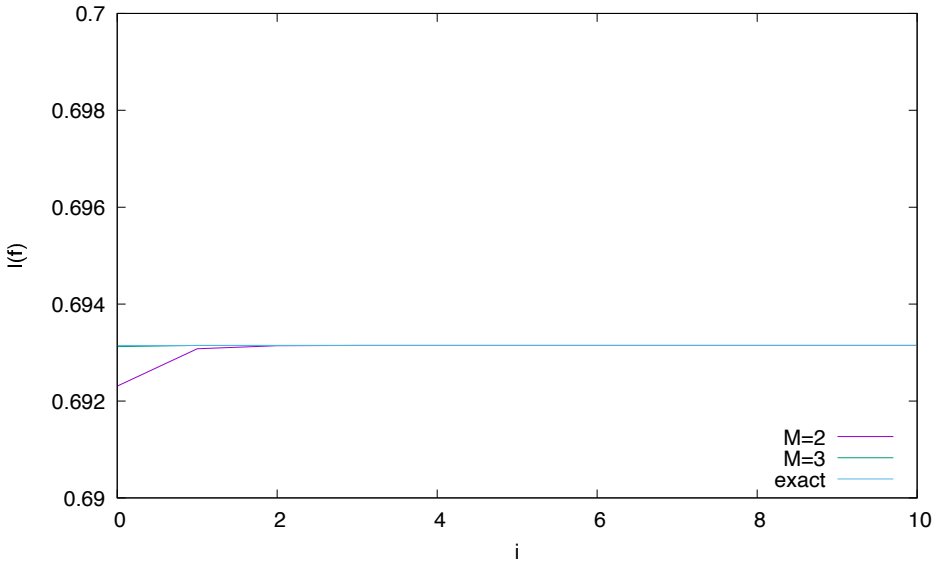

図 12: i を横軸に、(a) の $I(f)$ を数値積分した値を縦軸にとり、作成したグラフ, 横軸の範囲は $0 \leq i \leq 10$, 縦軸の範囲は $0.69 \leq I(f) \leq 0.70$

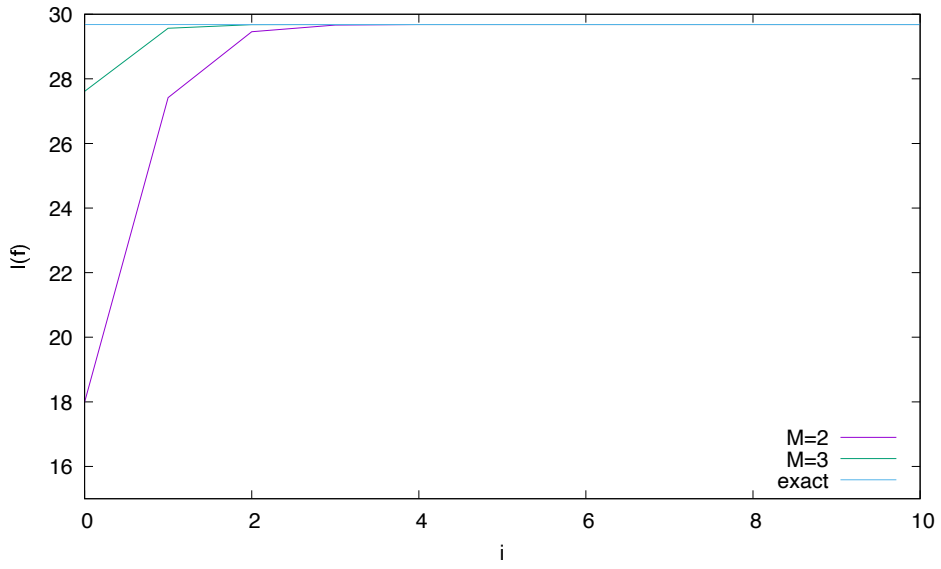

図 13: i を横軸に、(b) の $I(f)$ を数値積分した値を縦軸にとり、作成したグラフ, 横軸の範囲は $0 \leq i \leq 10$, 縦軸の範囲は $15 \leq I(f) \leq 30$

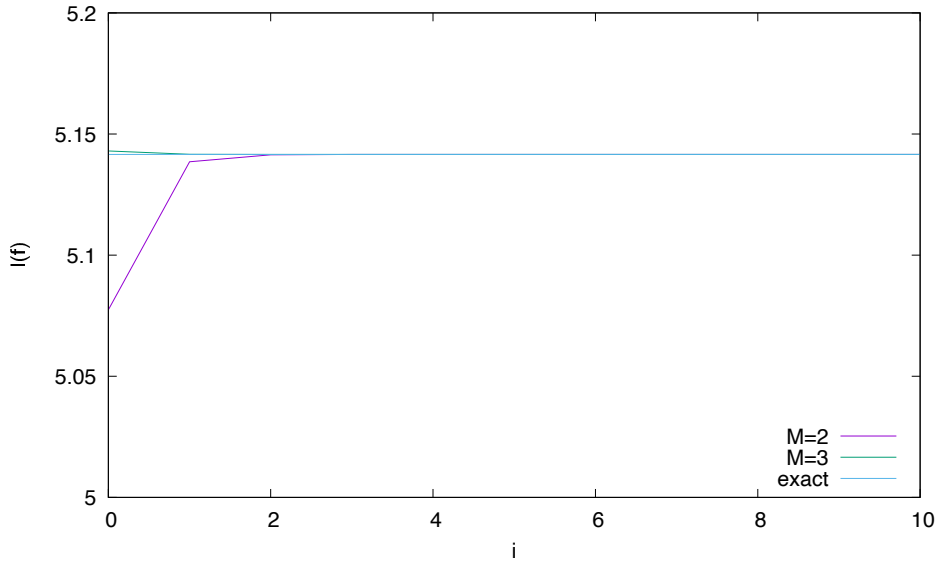

図 14: i を横軸に、(c) の $I(f)$ を数値積分した値を縦軸にとり、作成したグラフ, 横軸の範囲は $0 \leq i \leq 10$, 縦軸の範囲は $5.0 \leq I(f) \leq 5.2$

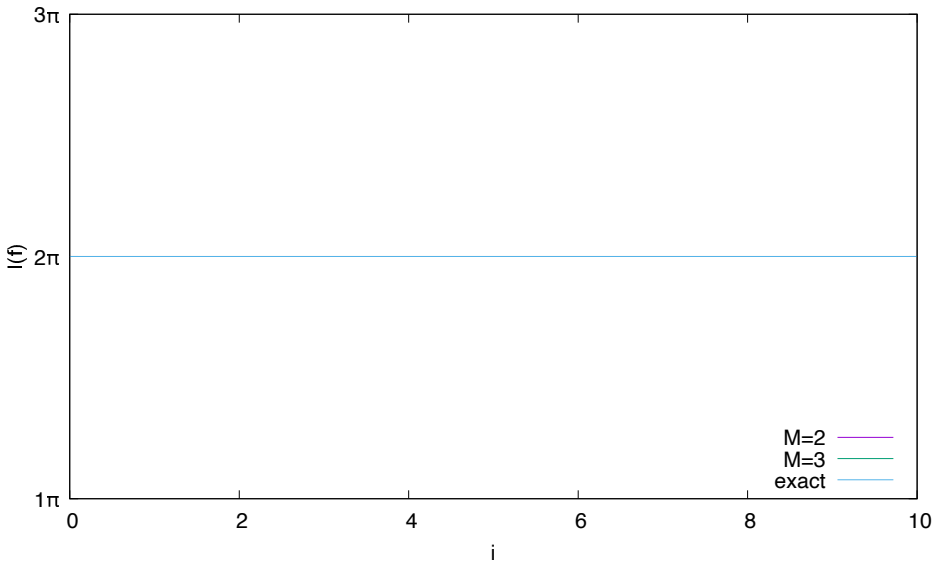

図 15: i を横軸に、(d) の $I(f)$ を数値積分した値を縦軸にとり、作成したグラフ, 横軸の範囲は $0 \leq i \leq 10$, 縦軸の範囲は $\pi \leq I(f) \leq 3\pi$

3.1.3 考察

図 12, 13, 14, 15 より、i の値を大きくする、すなわち分割数 $N(= 2^i)$ の値を大きくすれば数値積分の精度が高まることが分かる。また、n 次 Gauss 型積分公式の次数が上がれば精度が高まることも分かる。

図 4, 5, 6, 7 と比べると分かるように、Gauss 型積分公式は Newton-Cotes 公式と違い、分割数 $N(= 2^i)$ の値が小さくてもより高い精度で数値積分を行えることが分かる。

(d) $f(x) = 1 + \sin x$ $x \in (0, 2\pi)$ の関数を数値積分した際に、分割数 $n(= 2^i)$ の値が小さい場合でも誤差がないのは、4.2 節の課題 2 の場合と同様に、この関数を $x - y$ 平面に描いた時、$(\pi, 1)$ で点対称な軌道を描くことが理由ではないかと考えた。

3.2 課題 2

以下の積分を考える。

$$\int_0^1 \frac{\mathrm{e}^{-x}}{\sqrt{x}} dx \tag{22}$$

3.2.1 2(a)

定積分（22）に対して、積分区間 $(0,1)$ を $N=2^i$ $(i=0,1,...,10)$ 等分した小区間に $m=2,3$ の Gauss 型積分公式（15）を適用し、積分値 I_N とその誤差 $En(=|I_N-I|)$ を求める。ただし、正しい積分値を

$$\int_0^1 \frac{\mathrm{e}^{-x}}{\sqrt{x}}dx \approx 1.49364826562(=I) \tag{23}$$

と定める。

用いたコードを付録のソースコード 12 に示す。また、出力をもとに gnuplot を用いて以下の図 16, 17 を作成した。

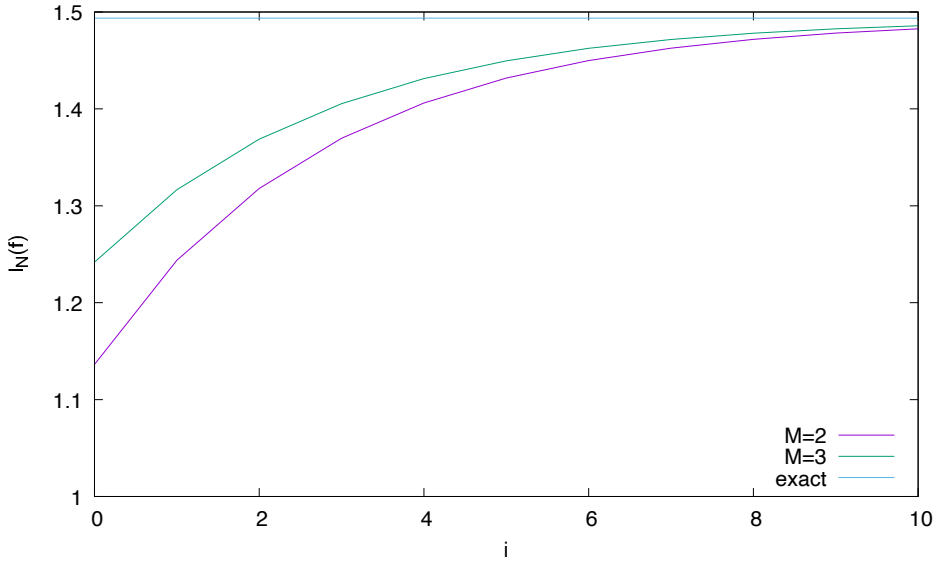

図 16: i を横軸に、数値積分の値 I_N を縦軸にとり、作成したグラフ, 横軸の範囲は $0 \leq i \leq 10$, 縦軸の範囲は $0 \leq I_N \leq 1.5$

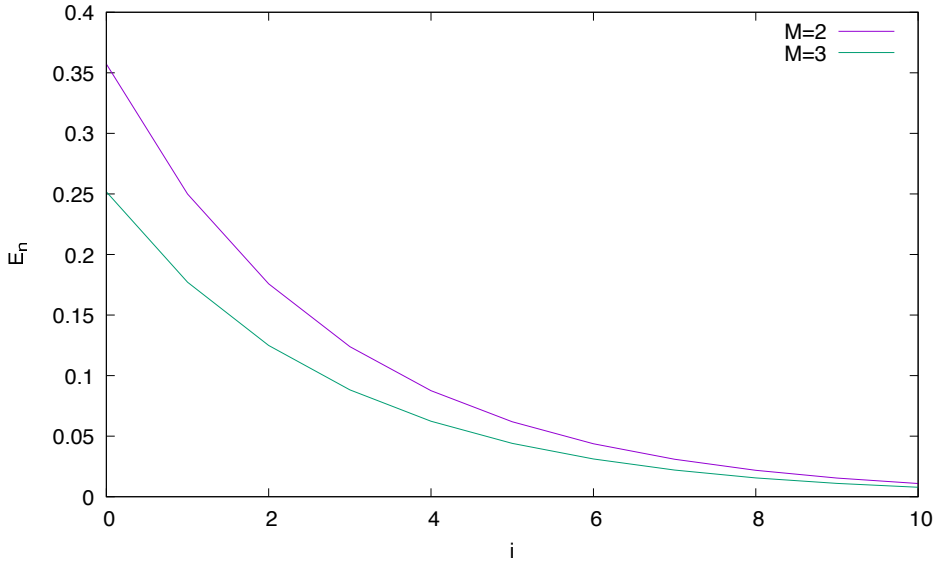

図 17: i を横軸に、誤差 E_n を縦軸にとり、作成したグラフ, 横軸の範囲は $0 \leq i \leq 10$, 縦軸の範囲は $0 \leq I_N \leq 0.4$

また、この定積分を複合中点公式、複合台形公式、複合 Simpson 公式による Newton-Cotes 公式を用いて計算した。用いたコードを付録のソースコード 13 に示す。出力は全て inf となった。これは、被積分関数が $x \to +0$ で発散することが原因だと考えられる。

3.2.2 2(b)

定積分（22）を $\sqrt{x} = t$ で変数変換する。

$$\begin{aligned} \sqrt{x} &= t \\ x &= t^2 \end{aligned}$$

$$dx = 2tdt$$

$$\begin{aligned} x &: 0 \to 1 \\ t &: 0 \to 1 \end{aligned} \tag{24}$$

$$\begin{aligned} \int_0^1 \frac{\mathrm{e}^{-x}}{\sqrt{x}} dx &= \int_0^1 \frac{\mathrm{e}^{-t^2}}{t} 2tdt \\ &= \int_0^1 2\mathrm{e}^{-t^2} dt \end{aligned}$$

この積分を 2(a) と同様の方法で数値積分し、積分値 I_N とその誤差 $En(=|I_N - I|)$ を求める。

用いたコードを付録のソースコード 14 に示す。また、出力をもとに gnuplot を用いて以下の図 18, 19 を作成した。（比較しやすいように 2(a) で作成した図と同じスケールにした。）

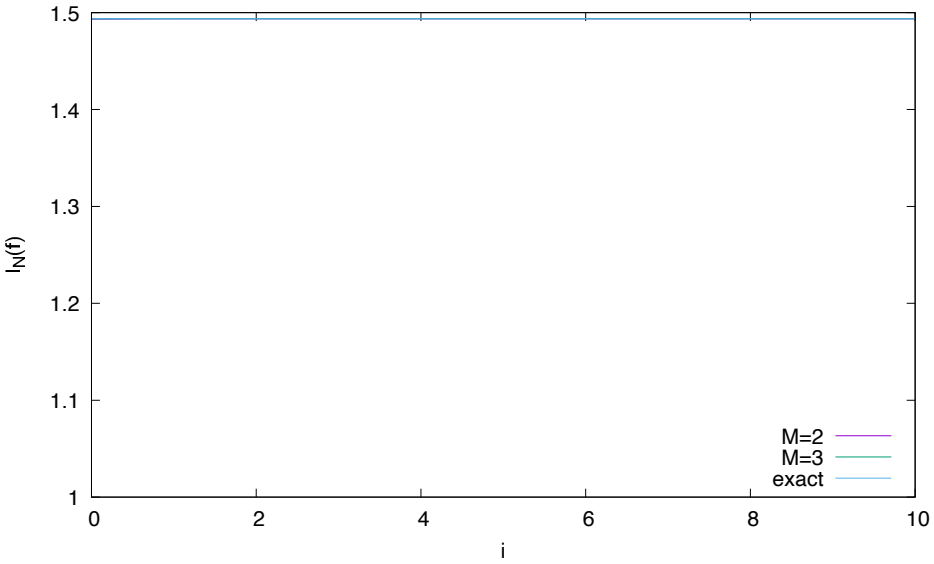

図 18: i を横軸に、数値積分の値 I_N を縦軸にとり、作成したグラフ, 横軸の範囲は $0 \leq i \leq 10$, 縦軸の範囲は $0 \leq I_N \leq 1.5$

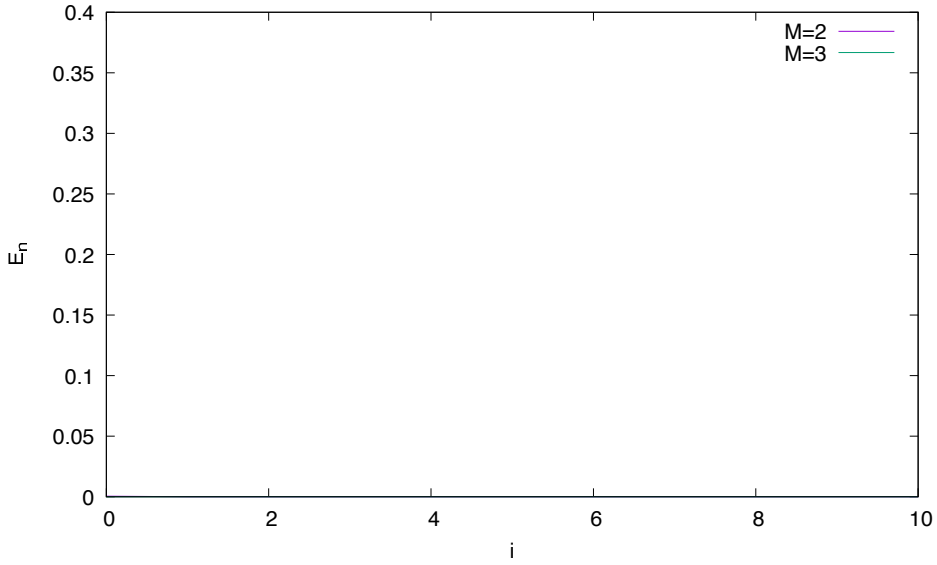

図 19: i を横軸に、誤差 E_n を縦軸にとり、作成したグラフ, 横軸の範囲は $0 \leq i \leq 10$, 縦軸の範囲は $0 \leq I_N \leq 0.4$, 図からはわかりにくいが i の値が小さい時から 0 に近い値をとっている

3.2.3 2(c)

定積分（22）を

$$\int_0^1 \frac{\mathrm{e}^{-x}}{\sqrt{x}}dx = \int_0^1 \frac{1}{\sqrt{x}}dx + \int_0^1 \frac{\mathrm{e}^{-x}-1}{\sqrt{x}}dx \tag{25}$$

と変形する。第一項は、

$$\int_0^1 \frac{1}{\sqrt{x}}dx = [2\sqrt{x}]_0^1 = 2 \tag{26}$$

となる。第二項は 2(a),(b) と同じ数値積分をそれぞれ行う。そして積分値 I_N とその誤差 $En(= |I_N - I|)$ を求める。

第二項の変数変換

$$\int_0^1 \frac{\mathrm{e}^{-x}-1}{\sqrt{x}}dx = \int_0^1 2(\mathrm{e}^{-t^2}-1)dt \tag{27}$$

用いたコードを付録のソースコード 15, 16 に示す。また、出力をもとに gnuplot を用いて以下の図 20, 21 を作成した。（比較しやすいように 2(a),(b) で作成した図と同じスケールにした。）ソースコード 16 の出力は 2(b) のものと全く同じになったので、図の作成を省く。

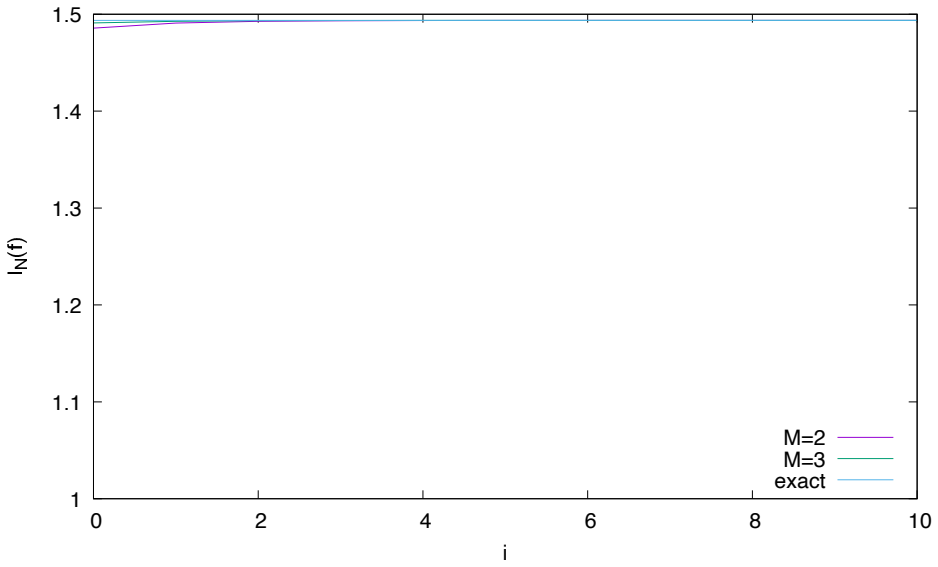

図 20: i を横軸に、数値積分の値 I_N を縦軸にとり、作成したグラフ, 横軸の範囲は $0 \leq i \leq 10$, 縦軸の範囲は $0 \leq I_N \leq 1.5$

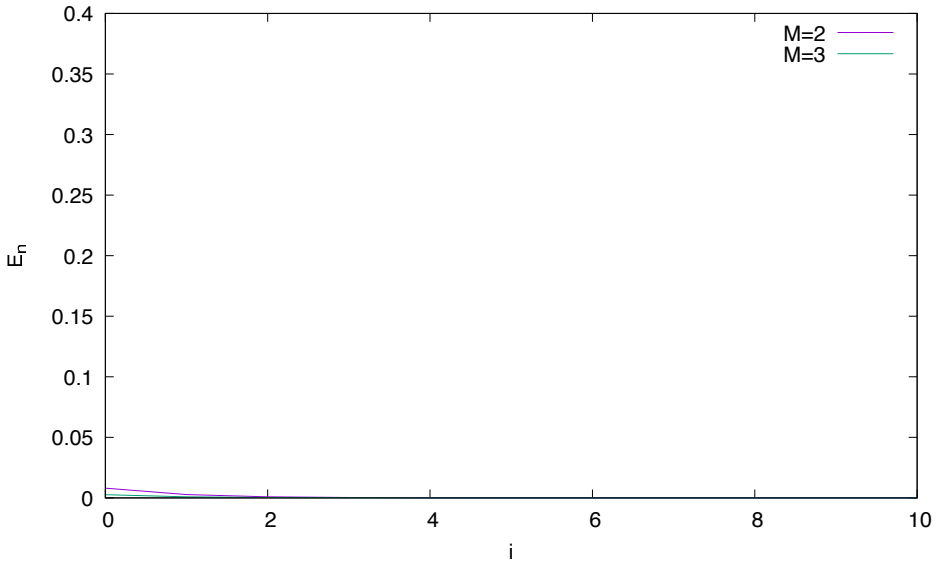

図 21: i を横軸に、誤差 E_n を縦軸にとり、作成したグラフ, 横軸の範囲は $0 \leq i \leq 10$, 縦軸の範囲は $0 \leq I_N \leq 0.4$

3.2.4 2(d)

（以下 (c) とは、式（25）の第二項を (a) と同じ数値積分を用いて計算したものとする。）

まず、精度としては (b)>(c)>>(a) の順に高いことが分かった。

(b) は i の値が小さい、すなわち分割数 $N(=2^i)$ の値が小さくてもほとんど正確な値をとる。$M=2$ の場合は $i=3$ で $E_n<10^{-6}$ に、$M=3$ の場合は $i=1$ で $E_n<10^{-6}$ になった。(c) は、$M=2$ の場合は $i=10$ で $E_n<10^{-6}$ に、$M=3$ の場合は $i=9$ で $E_n<10^{-6}$ になった。対して (a) は、$M=2$ の場合は $i=10$ で $E_n<10^{-1}$ に、$M=3$ の場合は $i=10$ で $E_n<10^{-2}$ になった。

(a) と (b) の精度の違いがこのようになったのは、式（22）の被積分関数は複雑な形をしているが、$\sqrt{x}=t$ で変数変換すると被積分関数が $2\mathrm{e}^{-t^2}$ とより単純な式になることが大きな要因だと考えられる。このことから、数値積分を行う被積分関数が単純であるほど精度が高くなることが分かった。

(a) と (c) の精度の違いがこのようになったのは、(c) では一部を解析的に計算したからである。残す関数がより複雑にならない場合は、一部を解析的に計算すれば精度が高くなることが分かった。

(b) と (c) の精度の違いがこのようになったのは、確かに式（25）の第一項を解析的に求め、(a) に比べれば精度は上がっているが、式（25）の第二項が複雑な形のままであるからだと考えられる。またこのことから、（少なくとも今回の場合においては、）一部を解析的に計算しても残された数値積分される関数が複雑な形をしているなら、全体として変数変換を行い、より単純な形に変えた方が精度が高くなることが分かった。

これらのことから、複雑な形をした被積分関数は、まず変数変換によって関数全体をより単純な関数に変えられないかを試み、それが無理な場合は残す関数がより複雑にならない範囲で一部を解析的に計算することによって、数値積分の精度を高めることができると言える。

4 付録

ソースコード 1: C 言語を用いて、Lagrange 補間を行うプログラム

```
1  //数理工学実験テーマ 4 //4.2節の課題 //1(a)
2  #include <stdio.h>
3  #include <math.h>
4
5  int main(int argc, char **argv)
6  {
7      int n;
8
9      int i; //x の添字
10     double x[17]; //x[0]は空､他は添字に対応
11     double fx[17];
12
13     int k; //y の添字
14     double y[152]; //x_i よりもさらに細かい分点 y_k
15     double l[17]; //l_i(x)
16     double P[152]; //P(x)
17
18     double a = 1.0; //左端
19     double b = 2.0; //右端
20
21     for (n = 2; n <= 16; n = n * 2)
22     {
23         printf("n=%d の時\n", n);
24         for (i = 1; i <= n; i++) //分点x_i の計算
25         {
26             x[i] = a + (b - a) / (n - 1) * (i - 1);
27         }
28         for (i = 1; i <= n; i++) //f(x)の計算
29         {
30             fx[i] = log(x[i]);
31         }
32
33         //ここからP(x)を数値的に計算
34         for (k = 1; k <= (n - 1) * 10 + 1; k++) //P の初期化
35         {
36             P[k] = 0;
37         }
38
39         for (k = 1; k <= (n - 1) * 10 + 1; k++) //
               x_i より 10 倍細かい分点 y_k
40         {
41             y[k] = a + (b - a) / (10 * (n - 1)) * (k - 1);
```

```
42          }
43
44          for (k = 1; k <= (n - 1) * 10 + 1; k++)
45          {
46              for (i = 1; i <= n; i++) //l の初期化
47              {
48                  l[i] = 1.0;
49              }
50
51              for (i = 1; i <= n; i++) //l_i(y_k)の計算
52              {
53                  for (int j = 1; j <= n; j++)
54                  {
55                      if (j != i)
56                      {
57                          l[i] = l[i] * (y[k] - x[j]) / (x[i]
                                - x[j]);
58                      }
59                  }
60              }
61              for (i = 1; i <= n; i++) //P(y_k)の計算
62              {
63                  P[k] = P[k] + fx[i] * l[i];
64              }
65
66              printf("%lf %lf %lf\n", y[k], P[k], log(y[k]));
67          }
68
69          printf("\n");
70      }
71
72      return 0;
73  }
```

ソースコード 2: C 言語を用いて、Lagrange 補間を行うプログラム

```
1  //数理工学実験テーマ 4 //4.2節の課題 //1(b)
2  #include <stdio.h>
3  #include <math.h>
4
5  double fb(double x)
6  {
7      return 1.0 / pow(x, 3.0);
8  }
9
10 int main(int argc, char **argv)
```

```
{
    int n;

    int i; //x の添字
    double x[17]; //x[0]は空、他は添字に対応
    double fx[17];

    int k; //y の添字
    double y[152]; //x_i よりもさらに細かい分点 y_k
    double l[17]; //l_i(x)
    double P[152]; //P(x)

    double a = 0.1; //左端
    double b = 2.0; //右端

    for (n = 2; n <= 16; n = n * 2)
    {
        printf("n=%d の時\n", n);
        for (i = 1; i <= n; i++) //分点x_i の計算
        {
            x[i] = a + (b - a) / (n - 1) * (i - 1);
        }
        for (i = 1; i <= n; i++) //f(x)の計算
        {
            fx[i] = fb(x[i]);
        }

        //ここからP(x)を数値的に計算
        for (k = 1; k <= (n - 1) * 10 + 1; k++) //P の初期化
        {
            P[k] = 0;
        }

        for (k = 1; k <= (n - 1) * 10 + 1; k++) //
            x_i より 10 倍細かい分点 y_k
        {
            y[k] = a + (b - a) / (10 * (n - 1)) * (k - 1);
        }

        for (k = 1; k <= (n - 1) * 10 + 1; k++)
        {
            for (i = 1; i <= n; i++) //l の初期化
            {
                l[i] = 1.0;
            }

```

```
56              for (i = 1; i <= n; i++) //l_i(y_k)の計算
57              {
58                  for (int j = 1; j <= n; j++)
59                  {
60                      if (j != i)
61                      {
62                          l[i] = l[i] * (y[k] - x[j]) / (x[i]
                                - x[j]);
63                      }
64                  }
65              }
66              for (i = 1; i <= n; i++) //P(y_k)の計算
67              {
68                  P[k] = P[k] + fx[i] * l[i];
69              }
70
71              printf("%lf %lf\n", y[k], P[k]);
72          }
73
74          printf("\n");
75      }
76
77      return 0;
78  }
```

ソースコード 3: C 言語を用いて、Lagrange 補間を行うプログラム

```
1  //数理工学実験テーマ 4 //4.2節の課題 //1(c)
2  #include <stdio.h>
3  #include <math.h>
4
5  double fc(double x)
6  {
7      return 1.0 / (1 + 25 * pow(x, 2.0));
8  }
9
10 int main(int argc, char **argv)
11 {
12     int n;
13
14     int i; //x の添字
15     double x[17]; //x[0]は空､他は添字に対応
16     double fx[17];
17
18     int k; //y の添字
19     double y[152]; //x_i よりもさらに細かい分点 y_k
```

```
20      double l[17]; //l_i(x)
21      double P[152]; //P(x)
22
23      double a = -1.0; //左端
24      double b = 1.0; //右端
25
26      for (n = 2; n <= 16; n = n * 2)
27      {
28          printf("n=%d の時\n", n);
29          for (i = 1; i <= n; i++) //分点x_i の計算
30          {
31              x[i] = a + (b - a) / (n - 1) * (i - 1);
32          }
33          for (i = 1; i <= n; i++) //f(x)の計算
34          {
35              fx[i] = fc(x[i]);
36          }
37
38          //ここからP(x)を数値的に計算
39          for (k = 1; k <= (n - 1) * 10 + 1; k++) //P の初期化
40          {
41              P[k] = 0;
42          }
43
44          for (k = 1; k <= (n - 1) * 10 + 1; k++) //
                x_i より 10 倍細かい分点 y_k
45          {
46              y[k] = a + (b - a) / (10 * (n - 1)) * (k - 1);
47          }
48
49          for (k = 1; k <= (n - 1) * 10 + 1; k++)
50          {
51              for (i = 1; i <= n; i++) //l の初期化
52              {
53                  l[i] = 1.0;
54              }
55
56              for (i = 1; i <= n; i++) //l_i(y_k)の計算
57              {
58                  for (int j = 1; j <= n; j++)
59                  {
60                      if (j != i)
61                      {
62                          l[i] = l[i] * (y[k] - x[j]) / (x[i]
                                - x[j]);
63                      }
```

```
64                  }
65              }
66              for (i = 1; i <= n; i++) //P(y_k)の計算
67              {
68                  P[k] = P[k] + fx[i] * l[i];
69              }
70
71              printf("%lf %lf\n", y[k], P[k]);
72          }
73
74          printf("\n");
75      }
76
77      return 0;
78  }
```

ソースコード 4: C 言語を用いて、複合中点公式、複合台形公式、複合 Simpson 公式により数値積分するプログラム

```
1  //数理工学実験テーマ 4 //4.2節の課題 //2&3(a)
2  #include <stdio.h>
3  #include <math.h>
4
5  double func(double x)
6  {
7      return 1.0 / x;
8  }
9
10 int main(int argc, char **argv)
11 {
12     double a = 1.0; //左端
13     double b = 2.0; //右端
14
15     int n, i; //分割数
16     double h; //分点の間隔
17
18     int m; //x の添字
19     double x[1025];
20     double fx[1025];
21
22     double integralC[11] = {0}; //複合中点公式による積分の値
           を格納する配列
23     double integralT[11] = {0}; //複合台形公式による...
24     double integralS[11] = {0}; //複合Simpson 公式による...
25
26     double I = log(2.0); //解析的に求めた積分の値
27
28     double EC[11] = {0}; //複合中点公式による積分の値と解析
           解との差を格納する配列
29     double ET[11] = {0}; //複合台形公式による...
30     double ES[11] = {0}; //複合Simpson 公式による...
31
32     for (i = 0; i <= 10; i++)
33     {
34         n = pow(2, i);
35         h = (b - a) / n; //刻み幅
36         for (m = 0; m <= n; m++) //分点x_i の計算
37         {
38             x[m] = a + m * h;
39             //printf("%d %lf\n", i, x[i]);
40         }
41         for (m = 0; m <= n; m++) //f(x)の計算
42         {
```

```
43              fx[m] = func(x[m]);
44              //printf("%d %lf\n", i, fx[i]);
45          }
46
47          //複合中点公式
48          for (m = 0; m <= n - 1; m++)
49          {
50              integralC[i] = integralC[i] + h * fx[(m + m + 1)
                    / 2];
51          }
52          EC[i] = integralC[i] - I; //誤差の計算
53
54          //複合台形公式
55          integralT[i] = h / 2.0 * (fx[0] + fx[n]);
56          for (m = 1; m <= n - 1; m++)
57          {
58              integralT[i] = integralT[i] + h * fx[m];
59          }
60          ET[i] = integralT[i] - I; //誤差の計算
61
62          //複合Simpson 公式
63          integralS[i] = h / 6.0 * (fx[0] + fx[n]);
64          for (m = 1; m <= n - 1; m++)
65          {
66              integralS[i] = integralS[i] + h / 3.0 * fx[m];
67          }
68          for (m = 0; m <= n - 1; m++)
69          {
70              integralS[i] = integralS[i] + 2 * h / 3.0 * fx[(
                    m + m + 1) / 2];
71          }
72          ES[i] = integralS[i] - I; //誤差の計算
73      }
74
75      printf("複合中点公式の場合\n");
76      printf("i の値　積分値\n");
77      for (i = 0; i <= 10; i++)
78      {
79          printf("%d %lf\n", i, integralC[i]);
80      }
81      printf("i の値　E_n の値\n");
82      for (i = 0; i <= 10; i++)
83      {
84          printf("%d %lf\n", i, EC[i]);
85      }
86      printf("\n");
```

```
87
88      printf("複合台形公式の場合\n");
89      printf("i の値   積分値\n");
90      for (i = 0; i <= 10; i++)
91      {
92          printf("%d %lf\n", i, integralT[i]);
93      }
94      printf("i の値   E_n の値\n");
95      for (i = 0; i <= 10; i++)
96      {
97          printf("%d %lf\n", i, ET[i]);
98      }
99      printf("\n");
100     printf("\n");
101
102     printf("複合Simpson 公式の場合\n");
103     printf("i の値   積分値\n");
104     for (i = 0; i <= 10; i++)
105     {
106         printf("%d %lf\n", i, integralS[i]);
107     }
108     printf("i の値   E_n の値\n");
109     for (i = 0; i <= 10; i++)
110     {
111         printf("%d %lf\n", i, ES[i]);
112     }
113     printf("\n");
114     printf("\n");
115
116     return 0;
117 }
```

ソースコード 5: C 言語を用いて、複合中点公式、複合台形公式、複合 Simpson 公式により数値積分するプログラム

```
1  //数理工学実験テーマ 4 //4.2節の課題 //2&3(b)
2  #include <stdio.h>
3  #include <math.h>
4
5  double func(double x)
6  {
7      return exp(5 * x);
8  }
9
10 int main(int argc, char **argv)
11 {
```

```
12      double a = -1.0; //左端
13      double b = 1.0; //右端
14
15      int n, i; //分割数
16      double h; //分点の間隔
17
18      int m; //x の添字
19      double x[1025];
20      double fx[1025];
21
22      double integralC[11] = {0}; //複合中点公式による積分の値
            を格納する配列
23      double integralT[11] = {0}; //複合台形公式による...
24      double integralS[11] = {0}; //複合Simpson 公式による...
25
26      double I = 1.0 / 5.0 * (exp(5.0) - exp(-5.0)); //解析
            的に求めた積分の値
27
28      double EC[11] = {0}; //複合中点公式による積分の値と解析
            解との差を格納する配列
29      double ET[11] = {0}; //複合台形公式による...
30      double ES[11] = {0}; //複合Simpson 公式による...
31
32      for (i = 0; i <= 10; i++)
33      {
34          n = pow(2, i);
35          h = (b - a) / n; //刻み幅
36          for (m = 0; m <= n; m++) //分点x_i の計算
37          {
38              x[m] = a + m * h;
39          }
40          for (m = 0; m <= n; m++) //f(x)の計算
41          {
42              fx[m] = func(x[m]);
43          }
44
45          //複合中点公式
46          for (m = 0; m <= n - 1; m++)
47          {
48              integralC[i] = integralC[i] + h * fx[(m + m + 1)
                    / 2];
49          }
50          EC[i] = integralC[i] - I; //誤差の計算
51
52          //複合台形公式
53          integralT[i] = h / 2.0 * (fx[0] + fx[n]);
```

```
54          for (m = 1; m <= n - 1; m++)
55          {
56              integralT[i] = integralT[i] + h * fx[m];
57          }
58          ET[i] = integralT[i] - I; //誤差の計算
59
60          //複合Simpson 公式
61          integralS[i] = h / 6.0 * (fx[0] + fx[n]);
62          for (m = 1; m <= n - 1; m++)
63          {
64              integralS[i] = integralS[i] + h / 3.0 * fx[m];
65          }
66          for (m = 0; m <= n - 1; m++)
67          {
68              integralS[i] = integralS[i] + 2 * h / 3.0 * fx[(
                    m + m + 1) / 2];
69          }
70          ES[i] = integralS[i] - I; //誤差の計算
71      }
72
73      printf("複合中点公式の場合\n");
74      printf("i の値  積分値\n");
75      for (i = 0; i <= 10; i++)
76      {
77          printf("%d %lf\n", i, integralC[i]);
78      }
79      printf("i の値  E_n の値\n");
80      for (i = 0; i <= 10; i++)
81      {
82          printf("%d %lf\n", i, EC[i]);
83      }
84      printf("\n");
85
86      printf("複合台形公式の場合\n");
87      printf("i の値  積分値\n");
88      for (i = 0; i <= 10; i++)
89      {
90          printf("%d %lf\n", i, integralT[i]);
91      }
92      printf("i の値  E_n の値\n");
93      for (i = 0; i <= 10; i++)
94      {
95          printf("%d %lf\n", i, ET[i]);
96      }
97      printf("\n");
98      printf("\n");
```

```
99
100     printf("複合Simpson 公式の場合\n");
101     printf("i の値   積分値\n");
102     for (i = 0; i <= 10; i++)
103     {
104         printf("%d %lf\n", i, integralS[i]);
105     }
106     printf("i の値   E_n の値\n");
107     for (i = 0; i <= 10; i++)
108     {
109         printf("%d %lf\n", i, ES[i]);
110     }
111     printf("\n");
112     printf("\n");
113
114     return 0;
115 }
```

ソースコード 6: C 言語を用いて、複合中点公式、複合台形公式、複合 Simpson 公式により数値積分するプログラム

```
1  //数理工学実験テーマ 4 //4.2節の課題 //2&3(c)
2  #include <stdio.h>
3  #include <math.h>
4
5  double func(double x)
6  {
7      return 1 + sin(x);
8  }
9
10 int main(int argc, char **argv)
11 {
12     double a = 0; //左端
13     double b = M_PI; //右端
14
15     int n, i; //分割数
16     double h; //分点の間隔
17
18     int m; //x の添字
19     double x[1025];
20     double fx[1025];
21
22     double integralC[11] = {0}; //複合中点公式による積分の値
           を格納する配列
23     double integralT[11] = {0}; //複合台形公式による...
24     double integralS[11] = {0}; //複合Simpson 公式による...
```

```
25
26      double I = 2 + M_PI; //解析的に求めた積分の値
27
28      double EC[11] = {0}; //複合中点公式による積分の値と解析
            解との差を格納する配列
29      double ET[11] = {0}; //複合台形公式による...
30      double ES[11] = {0}; //複合Simpson 公式による...
31
32      for (i = 0; i <= 10; i++)
33      {
34          n = pow(2, i);
35          h = (b - a) / n; //刻み幅
36          for (m = 0; m <= n; m++) //分点x_i の計算
37          {
38              x[m] = a + m * h;
39          }
40          for (m = 0; m <= n; m++) //f(x)の計算
41          {
42              fx[m] = func(x[m]);
43          }
44
45          //複合中点公式
46          for (m = 0; m <= n - 1; m++)
47          {
48              integralC[i] = integralC[i] + h * fx[(m + m + 1)
                    / 2];
49          }
50          EC[i] = integralC[i] - I; //誤差の計算
51
52          //複合台形公式
53          integralT[i] = h / 2.0 * (fx[0] + fx[n]);
54          for (m = 1; m <= n - 1; m++)
55          {
56              integralT[i] = integralT[i] + h * fx[m];
57          }
58          ET[i] = integralT[i] - I; //誤差の計算
59
60          //複合Simpson 公式
61          integralS[i] = h / 6.0 * (fx[0] + fx[n]);
62          for (m = 1; m <= n - 1; m++)
63          {
64              integralS[i] = integralS[i] + h / 3.0 * fx[m];
65          }
66          for (m = 0; m <= n - 1; m++)
67          {
68              integralS[i] = integralS[i] + 2 * h / 3.0 * fx[(
```

```
                m + m + 1) / 2];
69          }
70          ES[i] = integralS[i] - I; //誤差の計算
71      }
72
73      printf("複合中点公式の場合\n");
74      printf("i の値  積分値\n");
75      for (i = 0; i <= 10; i++)
76      {
77          printf("%d %lf\n", i, integralC[i]);
78      }
79      printf("i の値  E_n の値\n");
80      for (i = 0; i <= 10; i++)
81      {
82          printf("%d %lf\n", i, EC[i]);
83      }
84      printf("\n");
85
86      printf("複合台形公式の場合\n");
87      printf("i の値  積分値\n");
88      for (i = 0; i <= 10; i++)
89      {
90          printf("%d %lf\n", i, integralT[i]);
91      }
92      printf("i の値  E_n の値\n");
93      for (i = 0; i <= 10; i++)
94      {
95          printf("%d %lf\n", i, ET[i]);
96      }
97      printf("\n");
98      printf("\n");
99
100     printf("複合Simpson 公式の場合\n");
101     printf("i の値  積分値\n");
102     for (i = 0; i <= 10; i++)
103     {
104         printf("%d %lf\n", i, integralS[i]);
105     }
106     printf("i の値  E_n の値\n");
107     for (i = 0; i <= 10; i++)
108     {
109         printf("%d %lf\n", i, ES[i]);
110     }
111     printf("\n");
112     printf("\n");
113
```

```
114     return 0;
115 }
```

ソースコード 7: C 言語を用いて、複合中点公式、複合台形公式、複合 Simpson 公式により数値積分するプログラム

```
1  //数理工学実験テーマ 4 //4.2節の課題 //2&3(d)
2  #include <stdio.h>
3  #include <math.h>
4
5  double func(double x)
6  {
7      return 1 + sin(x);
8  }
9
10 int main(int argc, char **argv)
11 {
12     double a = 0; //左端
13     double b = 2 * M_PI; //右端
14
15     int n, i; //分割数
16     double h; //分点の間隔
17
18     int m; //x の添字
19     double x[1025];
20     double fx[1025];
21
22     double integralC[11] = {0}; //複合中点公式による積分の値
           を格納する配列
23     double integralT[11] = {0}; //複合台形公式による...
24     double integralS[11] = {0}; //複合Simpson 公式による...
25
26     double I = 2 * M_PI; //解析的に求めた積分の値
27
28     double EC[11] = {0}; //複合中点公式による積分の値と解析
           解との差を格納する配列
29     double ET[11] = {0}; //複合台形公式による...
30     double ES[11] = {0}; //複合Simpson 公式による...
31
32     for (i = 0; i <= 10; i++)
33     {
34         n = pow(2, i);
35         h = (b - a) / n; //刻み幅
36         for (m = 0; m <= n; m++) //分点x_i の計算
37         {
38             x[m] = a + m * h;
```

```
39          }
40          for (m = 0; m <= n; m++) //f(x)の計算
41          {
42              fx[m] = func(x[m]);
43          }
44
45          //複合中点公式
46          for (m = 0; m <= n - 1; m++)
47          {
48              integralC[i] = integralC[i] + h * fx[(m + m + 1)
                    / 2];
49          }
50          EC[i] = integralC[i] - I; //誤差の計算
51
52          //複合台形公式
53          integralT[i] = h / 2.0 * (fx[0] + fx[n]);
54          for (m = 1; m <= n - 1; m++)
55          {
56              integralT[i] = integralT[i] + h * fx[m];
57          }
58          ET[i] = integralT[i] - I; //誤差の計算
59
60          //複合Simpson 公式
61          integralS[i] = h / 6.0 * (fx[0] + fx[n]);
62          for (m = 1; m <= n - 1; m++)
63          {
64              integralS[i] = integralS[i] + h / 3.0 * fx[m];
65          }
66          for (m = 0; m <= n - 1; m++)
67          {
68              integralS[i] = integralS[i] + 2 * h / 3.0 * fx[(
                    m + m + 1) / 2];
69          }
70          ES[i] = integralS[i] - I; //誤差の計算
71      }
72
73      printf("複合中点公式の場合\n");
74      printf("i の値  積分値\n");
75      for (i = 0; i <= 10; i++)
76      {
77          printf("%d %lf\n", i, integralC[i]);
78      }
79      printf("i の値  E_n の値\n");
80      for (i = 0; i <= 10; i++)
81      {
82          printf("%d %lf\n", i, EC[i]);
```

```
83      }
84      printf("\n");
85
86      printf("複合台形公式の場合\n");
87      printf("i の値  積分値\n");
88      for (i = 0; i <= 10; i++)
89      {
90          printf("%d %lf\n", i, integralT[i]);
91      }
92      printf("i の値  E_n の値\n");
93      for (i = 0; i <= 10; i++)
94      {
95          printf("%d %lf\n", i, ET[i]);
96      }
97      printf("\n");
98      printf("\n");
99
100     printf("複合Simpson 公式の場合\n");
101     printf("i の値  積分値\n");
102     for (i = 0; i <= 10; i++)
103     {
104         printf("%d %lf\n", i, integralS[i]);
105     }
106     printf("i の値  E_n の値\n");
107     for (i = 0; i <= 10; i++)
108     {
109         printf("%d %lf\n", i, ES[i]);
110     }
111     printf("\n");
112     printf("\n");
113
114     return 0;
115 }
```

ソースコード 8: C 言語を用いて、複合 Gauss 型積分公式により数値積分するプログラム

```
1  //数理工学実験テーマ 4 //4.3節の課題 //1(a)
2  #include <stdio.h>
3  #include <math.h>
4
5  double func(double x)
6  {
7      return 1.0 / x;
8  }
9
10 int main(int argc, char **argv)
11 {
12     double a = 1.0; //左端
13     double b = 2.0; //右端
14
15     int N, i; //分割数
16
17     int j; //x の添字
18     double x[1025];
19
20     int m; //y とωの添字
21     double y2[3] = {0, -1.0 / sqrt(3), 1.0 / sqrt(3)}; //M
           =2のときのy、添字 m に対応
22     double omega2[3] = {0, 1, 1}; //M=2のときのω
23     double y3[4] = {0, -sqrt(0.6), 0, sqrt(0.6)}; //M=3のと
           きのy
24     double omega3[4] = {0, 5.0 / 9, 8.0 / 9, 5.0 / 9}; //M
           =3のときのω
25
26     double integral2[11] = {0}; //複合M(=2)次
           Gauss 型積分公式の値を格納する配列
27     double integral3[11] = {0}; //複合M(=3)次
           Gauss 型積分公式の値を格納する配列
28
29     double I = log(2.0); //解析的に求めた積分の値
30
31     for (i = 0; i <= 10; i++)
32     {
33         N = pow(2, i); //分割数
34         for (j = 0; j <= N; j++) //分点x_i の計算
35         {
36             x[j] = a + j * (b - a) / N;
37         }
38
39         //複合M(=2)次Gauss 型積分公式
```

```
40          for (j = 0; j <= N - 1; j++)
41          {
42              for (m = 1; m <= 2; m++)
43              {
44                  integral2[i] = integral2[i] + omega2[m] *
                        func((y2[m] + 1) * (x[j + 1] - x[j]) /
                        2 + x[j]) * (x[j + 1] - x[j]) / 2;
45              }
46          }
47
48          //複合M(=3)次Gauss 型積分公式
49          for (j = 0; j <= N - 1; j++)
50          {
51              for (m = 1; m <= 3; m++)
52              {
53                  integral3[i] = integral3[i] + omega3[m] *
                        func((y3[m] + 1) * (x[j + 1] - x[j]) /
                        2 + x[j]) * (x[j + 1] - x[j]) / 2;
54              }
55          }
56      }
57
58      printf("複合M(=2)次Gauss 型積分公式\n");
59      printf("i の値  積分値\n");
60      for (i = 0; i <= 10; i++) //数値積分の値を出力
61      {
62          printf("%d %lf\n", i, integral2[i]);
63      }
64      printf("\n");
65      printf("i の値  |E_n|の値\n");
66      for (i = 0; i <= 10; i++) //解析値との誤差を出力
67      {
68          printf("%d %lf\n", i, fabs(integral2[i] - I));
69      }
70      printf("\n\n");
71
72      printf("複合M(=3)次Gauss 型積分公式\n");
73      printf("i の値  積分値\n");
74      for (i = 0; i <= 10; i++) //数値積分の値を出力
75      {
76          printf("%d %lf\n", i, integral3[i]);
77      }
78      printf("\n");
79      printf("i の値  |E_n|の値\n");
80      for (i = 0; i <= 10; i++) //解析値との誤差を出力
81      {
```

```
82          printf("%d %lf\n", i, fabs(integral3[i] - I));
83      }
84      printf("\n");
85
86      return 0;
87  }
```

ソースコード 9: C 言語を用いて、複合 Gauss 型積分公式により数値積分するプログラム

```
1   //数理工学実験テーマ 4 //4.3節の課題 //1(b)
2   #include <stdio.h>
3   #include <math.h>
4
5   double func(double x)
6   {
7       return exp(5 * x);
8   }
9
10  int main(int argc, char **argv)
11  {
12      double a = -1.0; //左端
13      double b = 1.0; //右端
14
15      int N, i; //分割数
16
17      int j; //x の添字
18      double x[1025];
19
20      int m; //y とωの添字
21      double y2[3] = {0, -1.0 / sqrt(3), 1.0 / sqrt(3)}; //M
            =2のときのy、添字 m に対応
22      double omega2[3] = {0, 1, 1}; //M=2のときのω
23      double y3[4] = {0, -sqrt(0.6), 0, sqrt(0.6)}; //M=3のと
            きのy
24      double omega3[4] = {0, 5.0 / 9, 8.0 / 9, 5.0 / 9}; //M
            =3のときのω
25
26      double integral2[11] = {0}; //複合M(=2)次
            Gauss 型積分公式の値を格納する配列
27      double integral3[11] = {0}; //複合M(=3)次
            Gauss 型積分公式の値を格納する配列
28
29      double I = 1.0 / 5 * (exp(5.0) - exp(-5.0)); //解析的に
            求めた積分の値
30
```

```
31      for (i = 0; i <= 10; i++)
32      {
33          N = pow(2, i); //分割数
34          for (j = 0; j <= N; j++) //分点x_i の計算
35          {
36              x[j] = a + j * (b - a) / N;
37          }
38
39          //複合M(=2)次Gauss 型積分公式
40          for (j = 0; j <= N - 1; j++)
41          {
42              for (m = 1; m <= 2; m++)
43              {
44                  integral2[i] = integral2[i] + omega2[m] *
                        func((y2[m] + 1) * (x[j + 1] - x[j]) /
                        2 + x[j]) * (x[j + 1] - x[j]) / 2;
45              }
46          }
47
48          //複合M(=3)次Gauss 型積分公式
49          for (j = 0; j <= N - 1; j++)
50          {
51              for (m = 1; m <= 3; m++)
52              {
53                  integral3[i] = integral3[i] + omega3[m] *
                        func((y3[m] + 1) * (x[j + 1] - x[j]) /
                        2 + x[j]) * (x[j + 1] - x[j]) / 2;
54              }
55          }
56      }
57
58      printf("複合M(=2)次Gauss 型積分公式\n");
59      printf("i の値  積分値\n");
60      for (i = 0; i <= 10; i++) //数値積分の値を出力
61      {
62          printf("%d %lf\n", i, integral2[i]);
63      }
64      printf("\n");
65      printf("i の値  |E_n|の値\n");
66      for (i = 0; i <= 10; i++) //解析値との誤差を出力
67      {
68          printf("%d %lf\n", i, fabs(integral2[i] - I));
69      }
70      printf("\n\n");
71
72      printf("複合M(=3)次Gauss 型積分公式\n");
```

```
73      printf("i の値  積分値\n");
74      for (i = 0; i <= 10; i++) //数値積分の値を出力
75      {
76          printf("%d %lf\n", i, integral3[i]);
77      }
78      printf("\n");
79      printf("i の値  |E_n|の値\n");
80      for (i = 0; i <= 10; i++) //解析値との誤差を出力
81      {
82          printf("%d %lf\n", i, fabs(integral3[i] - I));
83      }
84      printf("\n");
85
86      return 0;
87  }
```

ソースコード 10: C 言語を用いて、複合 Gauss 型積分公式により数値積分するプログラム

```
1   //数理工学実験テーマ 4 //4.3節の課題 //1(c)
2   #include <stdio.h>
3   #include <math.h>
4
5   double func(double x)
6   {
7       return 1 + sin(x);
8   }
9
10  int main(int argc, char **argv)
11  {
12      double a = 0; //左端
13      double b = M_PI; //右端
14
15      int N, i; //分割数
16
17      int j; //x の添字
18      double x[1025];
19
20      int m; //y とωの添字
21      double y2[3] = {0, -1.0 / sqrt(3), 1.0 / sqrt(3)}; //M
            =2のときのy、添字 m に対応
22      double omega2[3] = {0, 1, 1}; //M=2のときのω
23      double y3[4] = {0, -sqrt(0.6), 0, sqrt(0.6)}; //M=3のと
            きのy
24      double omega3[4] = {0, 5.0 / 9, 8.0 / 9, 5.0 / 9}; //M
            =3のときのω
```

```
25
26      double integral2[11] = {0}; //複合M(=2)次
            Gauss 型積分公式の値を格納する配列
27      double integral3[11] = {0}; //複合M(=3)次
            Gauss 型積分公式の値を格納する配列
28
29      double I = 2 + M_PI; //解析的に求めた積分の値
30
31      for (i = 0; i <= 10; i++)
32      {
33          N = pow(2, i); //分割数
34          for (j = 0; j <= N; j++) //分点x_i の計算
35          {
36              x[j] = a + j * (b - a) / N;
37          }
38
39          //複合M(=2)次Gauss 型積分公式
40          for (j = 0; j <= N - 1; j++)
41          {
42              for (m = 1; m <= 2; m++)
43              {
44                  integral2[i] = integral2[i] + omega2[m] *
                        func((y2[m] + 1) * (x[j + 1] - x[j]) /
                        2 + x[j]) * (x[j + 1] - x[j]) / 2;
45              }
46          }
47
48          //複合M(=3)次Gauss 型積分公式
49          for (j = 0; j <= N - 1; j++)
50          {
51              for (m = 1; m <= 3; m++)
52              {
53                  integral3[i] = integral3[i] + omega3[m] *
                        func((y3[m] + 1) * (x[j + 1] - x[j]) /
                        2 + x[j]) * (x[j + 1] - x[j]) / 2;
54              }
55          }
56      }
57
58      printf("複合M(=2)次Gauss 型積分公式\n");
59      printf("i の値   積分値\n");
60      for (i = 0; i <= 10; i++) //数値積分の値を出力
61      {
62          printf("%d %lf\n", i, integral2[i]);
63      }
64      printf("\n");
```

```
65      printf("i の値  |E_n|の値\n");
66      for (i = 0; i <= 10; i++) //解析値との誤差を出力
67      {
68          printf("%d %lf\n", i, fabs(integral2[i] - I));
69      }
70      printf("\n\n");
71
72      printf("複合M(=3)次Gauss 型積分公式\n");
73      printf("i の値  積分値\n");
74      for (i = 0; i <= 10; i++) //数値積分の値を出力
75      {
76          printf("%d %lf\n", i, integral3[i]);
77      }
78      printf("\n");
79      printf("i の値  |E_n|の値\n");
80      for (i = 0; i <= 10; i++) //解析値との誤差を出力
81      {
82          printf("%d %lf\n", i, fabs(integral3[i] - I));
83      }
84      printf("\n");
85
86      return 0;
87  }
```

ソースコード 11: C 言語を用いて、複合 Gauss 型積分公式により数値積分するプログラム

```
1   //数理工学実験テーマ 4 //4.3節の課題 //1(d)
2   #include <stdio.h>
3   #include <math.h>
4
5   double func(double x)
6   {
7       return 1 + sin(x);
8   }
9
10  int main(int argc, char **argv)
11  {
12      double a = 0; //左端
13      double b = 2 * M_PI; //右端
14
15      int N, i; //分割数
16
17      int j; //x の添字
18      double x[1025];
19
```

```
20      int m; //yとωの添字
21      double y2[3] = {0, -1.0 / sqrt(3), 1.0 / sqrt(3)}; //M
            =2のときのy、添字mに対応
22      double omega2[3] = {0, 1, 1}; //M=2のときのω
23      double y3[4] = {0, -sqrt(0.6), 0, sqrt(0.6)}; //M=3のと
            きのy
24      double omega3[4] = {0, 5.0 / 9, 8.0 / 9, 5.0 / 9}; //M
            =3のときのω
25
26      double integral2[11] = {0}; //複合M(=2)次
            Gauss型積分公式の値を格納する配列
27      double integral3[11] = {0}; //複合M(=3)次
            Gauss型積分公式の値を格納する配列
28
29      double I = 2 * M_PI; //解析的に求めた積分の値
30
31      for (i = 0; i <= 10; i++)
32      {
33          N = pow(2, i); //分割数
34          for (j = 0; j <= N; j++) //分点x_iの計算
35          {
36              x[j] = a + j * (b - a) / N;
37          }
38
39          //複合M(=2)次Gauss型積分公式
40          for (j = 0; j <= N - 1; j++)
41          {
42              for (m = 1; m <= 2; m++)
43              {
44                  integral2[i] = integral2[i] + omega2[m] *
                        func((y2[m] + 1) * (x[j + 1] - x[j]) /
                        2 + x[j]) * (x[j + 1] - x[j]) / 2;
45              }
46          }
47
48          //複合M(=3)次Gauss型積分公式
49          for (j = 0; j <= N - 1; j++)
50          {
51              for (m = 1; m <= 3; m++)
52              {
53                  integral3[i] = integral3[i] + omega3[m] *
                        func((y3[m] + 1) * (x[j + 1] - x[j]) /
                        2 + x[j]) * (x[j + 1] - x[j]) / 2;
54              }
55          }
56      }
```

```
57
58      printf("複合M(=2)次Gauss 型積分公式\n");
59      printf("i の値  積分値\n");
60      for (i = 0; i <= 10; i++) //数値積分の値を出力
61      {
62          printf("%d %lf\n", i, integral2[i]);
63      }
64      printf("\n");
65      printf("i の値  |E_n|の値\n");
66      for (i = 0; i <= 10; i++) //解析値との誤差を出力
67      {
68          printf("%d %lf\n", i, fabs(integral2[i] - I));
69      }
70      printf("\n\n");
71
72      printf("複合M(=3)次Gauss 型積分公式\n");
73      printf("i の値  積分値\n");
74      for (i = 0; i <= 10; i++) //数値積分の値を出力
75      {
76          printf("%d %lf\n", i, integral3[i]);
77      }
78      printf("\n");
79      printf("i の値  |E_n|の値\n");
80      for (i = 0; i <= 10; i++) //解析値との誤差を出力
81      {
82          printf("%d %lf\n", i, fabs(integral3[i] - I));
83      }
84      printf("\n");
85
86      return 0;
87  }
```

ソースコード 12: C 言語を用いて、複合 Gauss 型積分公式により数値積分するプログラム

```
1  //数理工学実験テーマ 4 //4.3節の課題 //2(a)
2  #include <stdio.h>
3  #include <math.h>
4
5  double func(double x)
6  {
7      return exp(-x) / sqrt(x);
8  }
9
10 int main(int argc, char **argv)
11 {
12     double a = 0; //左端
13     double b = 1.0; //右端
14
15     int N, i; //分割数
16
17     int j; //x の添字
18     double x[1025];
19
20     int m; //y とωの添字
21     double y2[3] = {0, -1.0 / sqrt(3), 1.0 / sqrt(3)}; //M
           =2のときのy、添字 m に対応
22     double omega2[3] = {0, 1, 1}; //M=2のときのω
23     double y3[4] = {0, -sqrt(0.6), 0, sqrt(0.6)}; //M=3のと
           きのy
24     double omega3[4] = {0, 5.0 / 9, 8.0 / 9, 5.0 / 9}; //M
           =3のときのω
25
26     double integral2[11] = {0}; //複合M(=2)次
           Gauss 型積分公式の値を格納する配列
27     double integral3[11] = {0}; //複合M(=3)次
           Gauss 型積分公式の値を格納する配列
28
29     double I = 1.49364826562; //与えられた(解析的に求めた?)
           積分の値
30
31     for (i = 0; i <= 10; i++)
32     {
33         N = pow(2, i); //分割数
34         for (j = 0; j <= N; j++) //分点x_i の計算
35         {
36             x[j] = a + j * (b - a) / N;
37         }
38
```

```
39          //複合M(=2)次Gauss 型積分公式
40          for (j = 0; j <= N - 1; j++)
41          {
42              for (m = 1; m <= 2; m++)
43              {
44                  integral2[i] = integral2[i] + omega2[m] *
                        func((y2[m] + 1) * (x[j + 1] - x[j]) /
                        2 + x[j]) * (x[j + 1] - x[j]) / 2;
45              }
46          }
47
48          //複合M(=3)次Gauss 型積分公式
49          for (j = 0; j <= N - 1; j++)
50          {
51              for (m = 1; m <= 3; m++)
52              {
53                  integral3[i] = integral3[i] + omega3[m] *
                        func((y3[m] + 1) * (x[j + 1] - x[j]) /
                        2 + x[j]) * (x[j + 1] - x[j]) / 2;
54              }
55          }
56      }
57
58      printf("複合M(=2)次Gauss 型積分公式\n");
59      printf("i の値  積分値\n");
60      for (i = 0; i <= 10; i++) //数値積分の値を出力
61      {
62          printf("%d %lf\n", i, integral2[i]);
63      }
64      printf("\n");
65      printf("i の値  |E_n|の値\n");
66      for (i = 0; i <= 10; i++) //解析値との誤差を出力
67      {
68          printf("%d %lf\n", i, fabs(integral2[i] - I));
69      }
70      printf("\n\n");
71
72      printf("複合M(=3)次Gauss 型積分公式\n");
73      printf("i の値  積分値\n");
74      for (i = 0; i <= 10; i++) //数値積分の値を出力
75      {
76          printf("%d %lf\n", i, integral3[i]);
77      }
78      printf("\n");
79      printf("i の値  |E_n|の値\n");
80      for (i = 0; i <= 10; i++) //解析値との誤差を出力
```

```
81      {
82          printf("%d %lf\n", i, fabs(integral3[i] - I));
83      }
84      printf("\n");
85
86      return 0;
87  }
```

ソースコード 13: C 言語を用いて、複合中点公式、複合台形公式、複合 Simpson 公式により数値積分するプログラム

```
1   //数理工学実験テーマ 4 //4.2節の課題 //2(a) //Newton-
        Cotes 公式
2   #include <stdio.h>
3   #include <math.h>
4
5   double func(double x)
6   {
7       return exp(-x) / sqrt(x);
8   }
9
10  int main(int argc, char **argv)
11  {
12      double a = 0; //左端
13      double b = 1.0; //右端
14
15      int n, i; //分割数
16      double h; //分点の間隔
17
18      int m; //x の添字
19      double x[1025];
20      double fx[1025];
21
22      double integralC[11] = {0}; //複合中点公式による積分の値
            を格納する配列
23      double integralT[11] = {0}; //複合台形公式による...
24      double integralS[11] = {0}; //複合Simpson 公式による...
25
26      double I = 1.49364826562; //解析的に求めた積分の値
27
28      double EC[11] = {0}; //複合中点公式による積分の値と解析
            解との差を格納する配列
29      double ET[11] = {0}; //複合台形公式による...
30      double ES[11] = {0}; //複合Simpson 公式による...
31
32      for (i = 0; i <= 10; i++)
```

```
33      {
34          n = pow(2, i);
35          h = (b - a) / n; //刻み幅
36          for (m = 0; m <= n; m++) //分点x_i の計算
37          {
38              x[m] = a + m * h;
39          }
40          for (m = 0; m <= n; m++) //f(x)の計算
41          {
42              fx[m] = func(x[m]);
43          }
44
45          //複合中点公式
46          for (m = 0; m <= n - 1; m++)
47          {
48              integralC[i] = integralC[i] + h * fx[(m + m + 1)
                    / 2];
49          }
50          EC[i] = integralC[i] - I; //誤差の計算
51
52          //複合台形公式
53          integralT[i] = h / 2.0 * (fx[0] + fx[n]);
54          for (m = 1; m <= n - 1; m++)
55          {
56              integralT[i] = integralT[i] + h * fx[m];
57          }
58          ET[i] = integralT[i] - I; //誤差の計算
59
60          //複合Simpson 公式
61          integralS[i] = h / 6.0 * (fx[0] + fx[n]);
62          for (m = 1; m <= n - 1; m++)
63          {
64              integralS[i] = integralS[i] + h / 3.0 * fx[m];
65          }
66          for (m = 0; m <= n - 1; m++)
67          {
68              integralS[i] = integralS[i] + 2 * h / 3.0 * fx[(
                    m + m + 1) / 2];
69          }
70          ES[i] = integralS[i] - I; //誤差の計算
71      }
72
73      printf("複合中点公式の場合\n");
74      printf("i の値　積分値\n");
75      for (i = 0; i <= 10; i++)
76      {
```

```
77          printf("%d %lf\n", i, integralC[i]);
78      }
79      printf("i の値   E_n の値\n");
80      for (i = 0; i <= 10; i++)
81      {
82          printf("%d %lf\n", i, EC[i]);
83      }
84      printf("\n");
85
86      printf("複合台形公式の場合\n");
87      printf("i の値   積分値\n");
88      for (i = 0; i <= 10; i++)
89      {
90          printf("%d %lf\n", i, integralT[i]);
91      }
92      printf("i の値   E_n の値\n");
93      for (i = 0; i <= 10; i++)
94      {
95          printf("%d %lf\n", i, ET[i]);
96      }
97      printf("\n");
98      printf("\n");
99
100     printf("複合Simpson 公式の場合\n");
101     printf("i の値   積分値\n");
102     for (i = 0; i <= 10; i++)
103     {
104         printf("%d %lf\n", i, integralS[i]);
105     }
106     printf("i の値   E_n の値\n");
107     for (i = 0; i <= 10; i++)
108     {
109         printf("%d %lf\n", i, ES[i]);
110     }
111     printf("\n");
112     printf("\n");
113
114     return 0;
115 }
```

ソースコード 14: C 言語を用いて、複合 Gauss 型積分公式により数値積分するプログラム

```
1  //数理工学実験テーマ 4 //4.3節の課題 //2(b)
2  #include <stdio.h>
3  #include <math.h>
```

```
4
5  double func(double x)
6  {
7      return 2 * exp(-pow(x, 2.0));
8  }
9
10 int main(int argc, char **argv)
11 {
12     double a = 0; //左端
13     double b = 1.0; //右端
14
15     int N, i; //分割数
16
17     int j; //x の添字
18     double x[1025];
19
20     int m; //y とωの添字
21     double y2[3] = {0, -1.0 / sqrt(3), 1.0 / sqrt(3)}; //M
           =2のときのy、添字 m に対応
22     double omega2[3] = {0, 1, 1}; //M=2のときのω
23     double y3[4] = {0, -sqrt(0.6), 0, sqrt(0.6)}; //M=3のと
           きのy
24     double omega3[4] = {0, 5.0 / 9, 8.0 / 9, 5.0 / 9}; //M
           =3のときのω
25
26     double integral2[11] = {0}; //複合M(=2)次
           Gauss 型積分公式の値を格納する配列
27     double integral3[11] = {0}; //複合M(=3)次
           Gauss 型積分公式の値を格納する配列
28
29     double I = 1.49364826562; //与えられた(解析的に求めた?)
           積分の値
30
31     for (i = 0; i <= 10; i++)
32     {
33         N = pow(2, i); //分割数
34         for (j = 0; j <= N; j++) //分点x_i の計算
35         {
36             x[j] = a + j * (b - a) / N;
37         }
38
39         //複合M(=2)次Gauss 型積分公式
40         for (j = 0; j <= N - 1; j++)
41         {
42             for (m = 1; m <= 2; m++)
43             {
```

```
44                  integral2[i] = integral2[i] + omega2[m] *
                        func((y2[m] + 1) * (x[j + 1] - x[j]) /
                        2 + x[j]) * (x[j + 1] - x[j]) / 2;
45              }
46          }
47
48          //複合M(=3)次Gauss 型積分公式
49          for (j = 0; j <= N - 1; j++)
50          {
51              for (m = 1; m <= 3; m++)
52              {
53                  integral3[i] = integral3[i] + omega3[m] *
                        func((y3[m] + 1) * (x[j + 1] - x[j]) /
                        2 + x[j]) * (x[j + 1] - x[j]) / 2;
54              }
55          }
56      }
57
58      printf("複合M(=2)次Gauss 型積分公式\n");
59      printf("i の値  積分値\n");
60      for (i = 0; i <= 10; i++) //数値積分の値を出力
61      {
62          printf("%d %lf\n", i, integral2[i]);
63      }
64      printf("\n");
65      printf("i の値  |E_n|の値\n");
66      for (i = 0; i <= 10; i++) //解析値との誤差を出力
67      {
68          printf("%d %lf\n", i, fabs(integral2[i] - I));
69      }
70      printf("\n\n");
71
72      printf("複合M(=3)次Gauss 型積分公式\n");
73      printf("i の値  積分値\n");
74      for (i = 0; i <= 10; i++) //数値積分の値を出力
75      {
76          printf("%d %lf\n", i, integral3[i]);
77      }
78      printf("\n");
79      printf("i の値  |E_n|の値\n");
80      for (i = 0; i <= 10; i++) //解析値との誤差を出力
81      {
82          printf("%d %lf\n", i, fabs(integral3[i] - I));
83      }
84      printf("\n");
85
```

```
86      return 0;
87  }
```

ソースコード 15: C 言語を用いて、複合 Gauss 型積分公式により数値積分するプログラム

```
1   //数理工学実験テーマ 4 //4.3節の課題 //2(c) //(a)のやり方
2   #include <stdio.h>
3   #include <math.h>
4
5   double func(double x)
6   {
7       return (exp(-x) - 1) / sqrt(x);
8   }
9
10  int main(int argc, char **argv)
11  {
12      double a = 0; //左端
13      double b = 1.0; //右端
14
15      int N, i; //分割数
16
17      int j; //x の添字
18      double x[1025];
19
20      int m; //y とωの添字
21      double y2[3] = {0, -1.0 / sqrt(3), 1.0 / sqrt(3)}; //M
            =2のときのy、添字 m に対応
22      double omega2[3] = {0, 1, 1}; //M=2のときのω
23      double y3[4] = {0, -sqrt(0.6), 0, sqrt(0.6)}; //M=3のと
            きのy
24      double omega3[4] = {0, 5.0 / 9, 8.0 / 9, 5.0 / 9}; //M
            =3のときのω
25
26      double integral2[11] = {0}; //複合M(=2)次
            Gauss 型積分公式の値を格納する配列
27      double integral3[11] = {0}; //複合M(=3)次
            Gauss 型積分公式の値を格納する配列
28
29      double I = 1.49364826562; //与えられた(解析的に求めた?)
            積分の値
30
31      for (i = 0; i <= 10; i++)
32      {
33          N = pow(2, i); //分割数
34          for (j = 0; j <= N; j++) //分点x_i の計算
```

```
35          {
36              x[j] = a + j * (b - a) / N;
37          }
38
39          //複合M(=2)次Gauss 型積分公式
40          for (j = 0; j <= N - 1; j++)
41          {
42              for (m = 1; m <= 2; m++)
43              {
44                  integral2[i] = integral2[i] + omega2[m] *
                        func((y2[m] + 1) * (x[j + 1] - x[j]) /
                        2 + x[j]) * (x[j + 1] - x[j]) / 2;
45              }
46          }
47
48          //複合M(=3)次Gauss 型積分公式
49          for (j = 0; j <= N - 1; j++)
50          {
51              for (m = 1; m <= 3; m++)
52              {
53                  integral3[i] = integral3[i] + omega3[m] *
                        func((y3[m] + 1) * (x[j + 1] - x[j]) /
                        2 + x[j]) * (x[j + 1] - x[j]) / 2;
54              }
55          }
56      }
57
58      printf("複合M(=2)次Gauss 型積分公式\n");
59      printf("i の値  積分値\n");
60      for (i = 0; i <= 10; i++) //数値積分の値を出力
61      {
62          printf("%d %lf\n", i, 2 + integral2[i]);
63      }
64      printf("\n");
65      printf("i の値  |E_n|の値\n");
66      for (i = 0; i <= 10; i++) //解析値との誤差を出力
67      {
68          printf("%d %lf\n", i, fabs(2 + integral2[i] - I));
69      }
70      printf("\n\n");
71
72      printf("複合M(=3)次Gauss 型積分公式\n");
73      printf("i の値  積分値\n");
74      for (i = 0; i <= 10; i++) //数値積分の値を出力
75      {
76          printf("%d %lf\n", i, 2 + integral3[i]);
```

```
77      }
78      printf("\n");
79      printf("i の値  |E_n|の値\n");
80      for (i = 0; i <= 10; i++) //解析値との誤差を出力
81      {
82          printf("%d %lf\n", i, fabs(2 + integral3[i] - I));
83      }
84      printf("\n");
85
86      return 0;
87  }
```

ソースコード 16: C 言語を用いて、複合 Gauss 型積分公式により数値積分するプログラム

```
1   //数理工学実験テーマ 4 //4.3節の課題 //2(c) //(b)のやり方
2   #include <stdio.h>
3   #include <math.h>
4
5   double func(double x)
6   {
7       return 2 * exp(-pow(x, 2.0)) - 2;
8   }
9
10  int main(int argc, char **argv)
11  {
12      double a = 0; //左端
13      double b = 1.0; //右端
14
15      int N, i; //分割数
16
17      int j; //x の添字
18      double x[1025];
19
20      int m; //y とωの添字
21      double y2[3] = {0, -1.0 / sqrt(3), 1.0 / sqrt(3)}; //M
            =2のときのy、添字 m に対応
22      double omega2[3] = {0, 1, 1}; //M=2のときのω
23      double y3[4] = {0, -sqrt(0.6), 0, sqrt(0.6)}; //M=3のと
            きのy
24      double omega3[4] = {0, 5.0 / 9, 8.0 / 9, 5.0 / 9}; //M
            =3のときのω
25
26      double integral2[11] = {0}; //複合M(=2)次
            Gauss 型積分公式の値を格納する配列
27      double integral3[11] = {0}; //複合M(=3)次
```

```
        Gauss 型積分公式の値を格納する配列
28
29      double I = 1.49364826562; //与えられた(解析的に求めた?)
            積分の値
30
31      for (i = 0; i <= 10; i++)
32      {
33          N = pow(2, i); //分割数
34          for (j = 0; j <= N; j++) //分点x_i の計算
35          {
36              x[j] = a + j * (b - a) / N;
37          }
38
39          //複合M(=2)次Gauss 型積分公式
40          for (j = 0; j <= N - 1; j++)
41          {
42              for (m = 1; m <= 2; m++)
43              {
44                  integral2[i] = integral2[i] + omega2[m] *
                        func((y2[m] + 1) * (x[j + 1] - x[j]) /
                        2 + x[j]) * (x[j + 1] - x[j]) / 2;
45              }
46          }
47
48          //複合M(=3)次Gauss 型積分公式
49          for (j = 0; j <= N - 1; j++)
50          {
51              for (m = 1; m <= 3; m++)
52              {
53                  integral3[i] = integral3[i] + omega3[m] *
                        func((y3[m] + 1) * (x[j + 1] - x[j]) /
                        2 + x[j]) * (x[j + 1] - x[j]) / 2;
54              }
55          }
56      }
57
58      printf("複合M(=2)次Gauss 型積分公式\n");
59      printf("i の値  積分値\n");
60      for (i = 0; i <= 10; i++) //数値積分の値を出力
61      {
62          printf("%d %lf\n", i, 2 + integral2[i]);
63      }
64      printf("\n");
65      printf("i の値  |E_n|の値\n");
66      for (i = 0; i <= 10; i++) //解析値との誤差を出力
67      {
```

```
68          printf("%d %lf\n", i, fabs(2 + integral2[i] - I));
69      }
70      printf("\n\n");
71
72      printf("複合M(=3)次Gauss 型積分公式\n");
73      printf("i の値  積分値\n");
74      for (i = 0; i <= 10; i++) //数値積分の値を出力
75      {
76          printf("%d %lf\n", i, 2 + integral3[i]);
77      }
78      printf("\n");
79      printf("i の値  |E_n|の値\n");
80      for (i = 0; i <= 10; i++) //解析値との誤差を出力
81      {
82          printf("%d %lf\n", i, fabs(2 + integral3[i] - I));
83      }
84      printf("\n");
85
86      return 0;
87  }
```

5 参考文献

- 数理工学実験テキスト
- Cloud LaTeX
  https://cloudlatex.io
- LaTeX コマンド一覧（リスト）
  https://medemanabu.net/latex/latex-commands-list/
- LaTeX にソースコードを【美しく】貼る方法
  https://ta-b0.hateblo.jp/entry/2020/08/13/001223
- 範囲外から範囲外へ動く場合の線の描画
  https://qiita.com/bunzaemon/items/c51c1ffb01f011252a27
- コメント（注釈） - 複数行のコメントアウト
  https://medemanabu.net/latex/comment/
- 【LaTeX】箇条書きの方法ついて徹底解説
  https://mathlandscape.com/latex-enum/
- 【gnuplot】sin 波を書いたときの横軸を pi にしたい
  http://coffee.guhaw.com/Entry/194/
- \rm と \textrm と \mathrm の使い分け
  https://e-v-e.hatenablog.com/entry/20150108/1420715672