令和四年度後期 数理工学実験 課題レポート

常微分方程式の数値解法

情報学科 2 回生 平田蓮

学生番号: 1029342830

実験日: 10 月 11, 17, 18, 24 日

実験場所: 京都大学工学部総合校舎数理計算機室

10 月 31 日 提出

目次

1　目的

多くの数理モデルには、微分方程式が現れる。これの解を解析的に得ることは一般的には難しいため、数値計算でその近似解求めるためのアルゴリズムを学習する。

2　原理

本章では、続く課題で用いるアルゴリズムについて述べる。

2.1　オイラー法

以下の関数を考える。

$$\frac{\mathrm{d}u}{\mathrm{d}t}(t) = f(t, u(t)),\ u(0) = u_0 \tag{2.1}$$

この式の両辺を t について、$[t_a, t_b]$ の範囲で積分すると、

$$u(t_b) - u(t_a) = \int_{t_a}^{t_b} f(t, u(t))\mathrm{d}t$$

を得る。$\Delta t = t_b - t_a$ とすると、範囲内の積分を幅 Δt、高さ $f(t_a, u(t_a))$ の長方形の面積で近似でき、

$$\begin{aligned} u(t_b) - u(t_a) &\approx f(t_a, u(t_a))\Delta t \\ \rightarrow u(t_b) &\approx u(t_a) + f(t_a, u(t_a))\Delta t \end{aligned}$$

と書ける。ここで (a, b) を $(n, n+1)$ に置き換え、さらに $u(t_n)$ を u_n に書き換えると、

$$u_{n+1} \approx u_n + f(t_n, u_n)\Delta t \tag{2.2}$$

を得る。この式を n に対して繰り返し適用することで、式 (2.1) を満たす関数を数値計算することができる。

式 (2.2) は積分の近似の際に $f(t_a, u(t_a))$ を用いたが、$f(t_b, u(t_b))$ を用いても同様に近似でき、前者を前進オイラー法、後者を後退オイラー法と呼ぶ。

2.2　クランク・ニコルソン法

オイラー法では $[t_n, t_{n+1}]$ の積分を長方形の面積で近似したが、範囲内の関数を 1 次関数で近似することで、台形の面積で近似でき、精度を向上させることができる。台形で近似を行うと、

$$u_{n+1} \approx u_n + \frac{f(t_n, u_n) + f(t_{n+1}, u_{n+1})}{2}\Delta t \tag{2.3}$$

を得る。この式は、u_{n+1} の近似が u_{n+1} を用いて陰的に表されている陰解法であるため、そのまま計算するには工夫が必要である。

2.3　アダムス・バッシュフォース法

オイラー法やクランク・ニコルソン法では、u_{n+1} を求める際に u_n のみを用いるため、一段法と呼ばれる。対して、u_{n-1} や u_{n-2} など、より過去の値も用いて精度を高めるアルゴリズムを多段法と呼ぶ。

過去に計算した $N+1$ 個の点を用いてラグランジュ補間により式 (2.1) を満たす $f(t,u(t))$ を N 次多項式で構成することを考える。

2.3.1 $N=1$ の場合

まず、1 次式で近似する場合を考える。すでに計算した 2 点 $(t_n, f(t_n,u_n))$, $(t_{n-1}, f(t_{n-1},u_{n-1}))$ を用いてラグランジュ補完を行うと、

$$f(t,u(t)) \approx \frac{t-t_{n-1}}{t_n-t_{n-1}} f(t_n,u_n) + \frac{t-t_n}{t_{n-1}-t_n} f(t_{n-1},u_{n-1})$$

を得る。$t_n - t_{n-1} = \Delta t$ を踏まえると、

$$\begin{aligned} f(t,u(t)) &\approx \frac{t-t_{n-1}}{\Delta t} f(t_n,u_n) - \frac{t-t_n}{\Delta t} f(t_{n-1},u_{n-1}) \\ &= \frac{f(t_n,u_n)-f(t_{n-1},u_{n-1})}{\Delta t} t - \frac{t_{n-1}f(t_n,u_n) - t_n f(t_{n-1},u_{n-1})}{\Delta t} \end{aligned}$$

と書ける。これを $t_{n+1} - t_n = \Delta t$ を踏まえて $[t_n, t_{n+1}]$ の範囲で積分すると、

$$\begin{aligned} \int_{t_n}^{t_{n+1}} f(t,u(t))dt &\approx \int_{t_n}^{t_{n+1}} \left\{ \frac{f(t_n,u_n)-f(t_{n-1},u_{n-1})}{\Delta t} t - \frac{t_{n-1}f(t_n,u_n) - t_n f(t_{n-1},u_{n-1})}{\Delta t} \right\} dt \\ &= 2\Delta t f(t_n,u_n) - \frac{\Delta t}{2} f(t_n,u_n) - \frac{\Delta t}{2} f(t_{n-1},u_{n-1}) \\ &= \frac{\Delta t}{2} \{3f(t_n,u_n) - f(t_{n-1},u_{n-1})\} \end{aligned}$$

よって、

$$u_{n+1} \approx u_n + \frac{\Delta t}{2} \{3f(t_n,u_n) - f(t_{n-1},u_{n-1})\} \tag{2.4}$$

となる。これを 2 次アダムス・バッシュフォース法と呼ぶ。

2.3.2 $N=2$ の場合

2 次式で近似を行う場合、3 点が必要である。$(t_n, f(t_n,u_n))$, $(t_{n-1}, f(t_{n-1},u_{n-1}))$, $(t_{n-2}, f(t_{n-2},u_{n-2}))$ を用いて同様に補間を行うと、

$$\begin{aligned} f(t,u(t)) \approx &\frac{(t-t_{n-1})(t-t_{n-2})}{(t_n-t_{n-1})(t_n-t_{n-2})} f(t_n,u_n) + \\ &\frac{(t-t_{n-2})(t-t_n)}{(t_{n-1}-t_{n-2})(t_{n-1}-t_n)} f(t_{n-1},u_{n-1}) + \\ &\frac{(t-t_n)(t-t_{n-1})}{(t_{n-2}-t_n)(t_{n-2}-t_{n-1})} f(t_{n-2},u_{n-2}) \end{aligned}$$

を得る。$N=1$ の場合と同様に積分を行うと、

$$\begin{aligned} \int_{t_n}^{t_{n+1}} f(t,u(t))dt &\approx \frac{\Delta t}{12} \{23f(t_n,u_n) - 16f(t_{n-1},u_{n-1}) + 5f(t_{n-2},u_{n-2})\} \\ \therefore u_{n+1} &\approx u_n + \frac{\Delta t}{12} \{23f(t_n,u_n) - 16f(t_{n-1},u_{n-1}) + 5f(t_{n-2},u_{n-2})\} \end{aligned} \tag{2.5}$$

となる [1]。これを 3 次アダムス・バッシュフォース法と呼ぶ。

2.4 ホイン法

2.2 節で、クランク・ニコルソン法を計算するには工夫が必要であると述べた。工夫の一つとして、$f(t_{n+1}, u_{n+1})$ を u_n と $f(t_n, u_n)$ を用いて表すことを考える。オイラー法を用いると、

$$u_{n+1} = u_n + f(t_n, u_n)\Delta t$$

と書ける。これを式 (2.3) に代入すると、

$$u_{n+1} \approx u_n + \frac{f(t_n, u_n) + f(t_{n+1}, u_n + f(t_n, u_n)\Delta t)}{2}\Delta t \tag{2.6}$$

を得る。これをホイン法と呼び、u_{n+1} の近似が u_n のみによって陽的に表されている陽解法である。

2.5 ルンゲ・クッタ法

ホイン法は 2 次の一段法であるが、これを 4 次に拡張したものを 4 次ルンゲ・クッタ法と呼び、これは以下の式で表される。

$$u_{n+1} = u_n + \frac{F_1 + 2F_2 + 2F_3 + F_4}{6} \tag{2.7}$$

$$\begin{cases} F_1 = f(t_n, u_n) \\ F_2 = f(t_n + \dfrac{\Delta t}{2}, u_n + F_1\dfrac{\Delta t}{2}) \\ F_3 = f(t_n + \dfrac{\Delta t}{2}, u_n + F_2\dfrac{\Delta t}{2}) \\ F_4 = f(t_{n+1}, u_n + F_3\Delta t) \end{cases} \tag{2.8}$$

これの証明は膨大な記述量であるため、ここでは省略する。詳細は [2] を参照されたい。

3 課題

3.1 課題 3 常微分方程式の数値解

式 (3.1) で表される常微分方程式の初期値問題を考える。

$$\frac{\mathrm{d}u}{\mathrm{d}t} = u,\ u(0) = 1 \tag{3.1}$$

ステップ幅を Δt として、t での数値解を $u_{\mathrm{calc}}(t, \Delta t)$ とする。$t = 1$ において、数値解 $u_{\mathrm{calc}}(1, \Delta t)$ と厳密解 $u(1)$ の差を

$$E(\Delta t) = |u_{\mathrm{calc}}(1, \Delta t) - u(1)| \tag{3.2}$$

とする。

以下に示すアルゴリズムについて、式 (3.2) の関数形を調べる。

- 前進オイラー法 (FE)
- 2 次アダムス・バッシュフォース法 (AB2)
- 3 次アダムス・バッシュフォース法 (AB3)
- ホイン法 (H)
- 4 次ルンゲ・クッタ法 (RK)

3.1.1　結果

描画したグラフを以下に示す。

■前進オイラー法

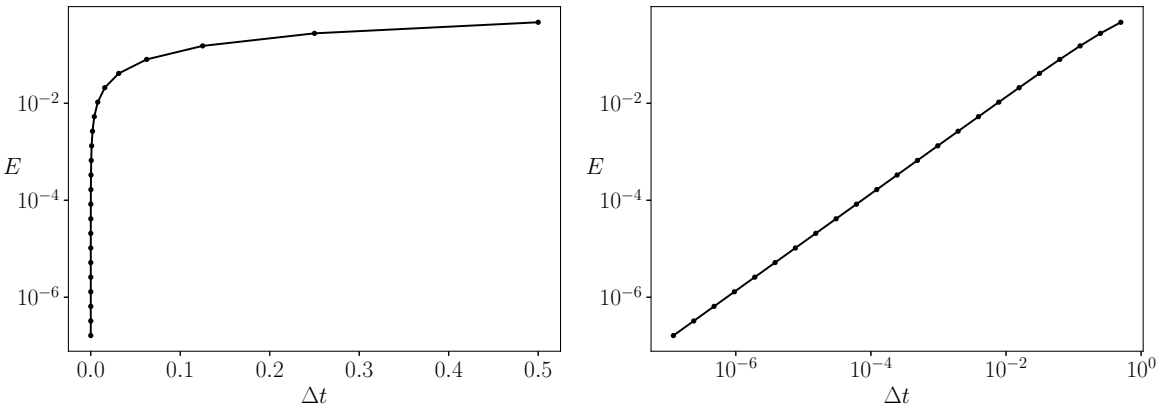

図 1　前進オイラー法を用いて計算した $E(\Delta t)$。左は片対数、右は両対数グラフ

■2 次アダムス・バッシュフォース法

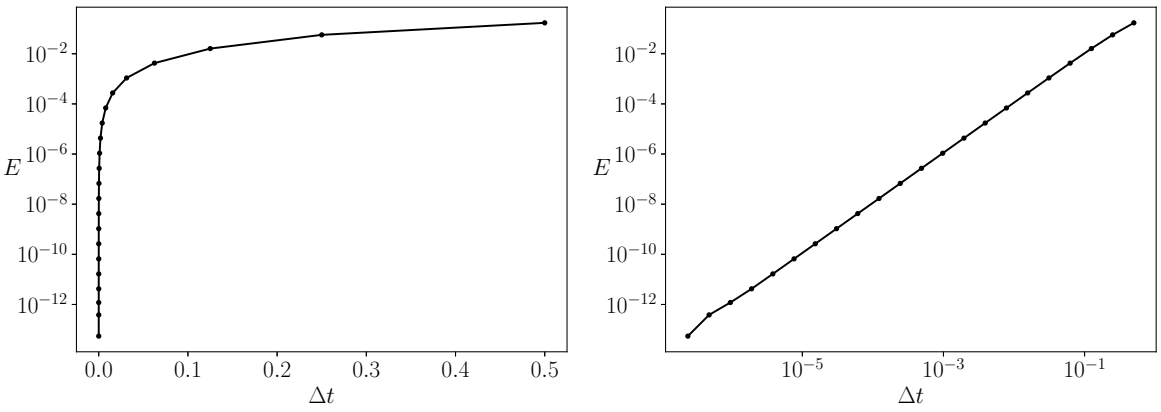

図 2　2 次アダムス・バッシュフォース法を用いて計算した $E(\Delta t)$。左は片対数、右は両対数グラフ

■3 次アダムス・バッシュフォース法

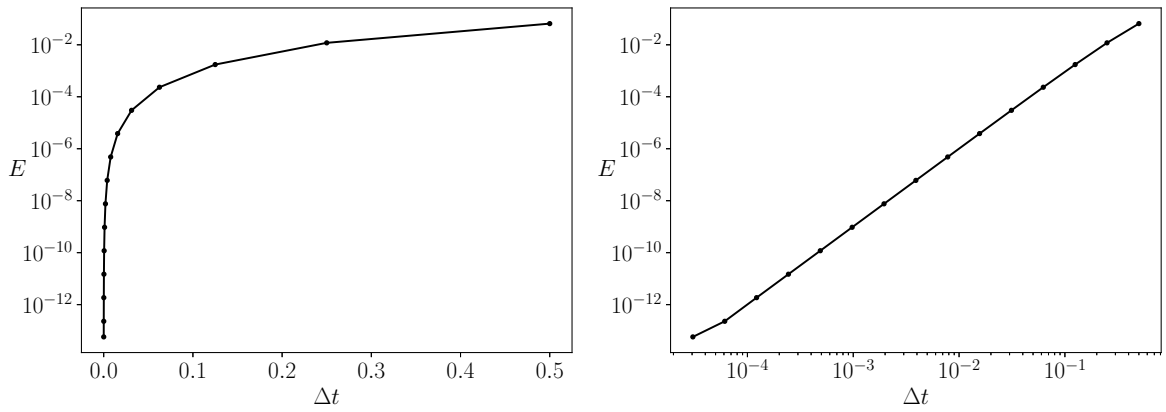

図 3　3 次アダムス・バッシュフォース法を用いて計算した $E(\Delta t)$。左は片対数、右は両対数グラフ

■ホイン法

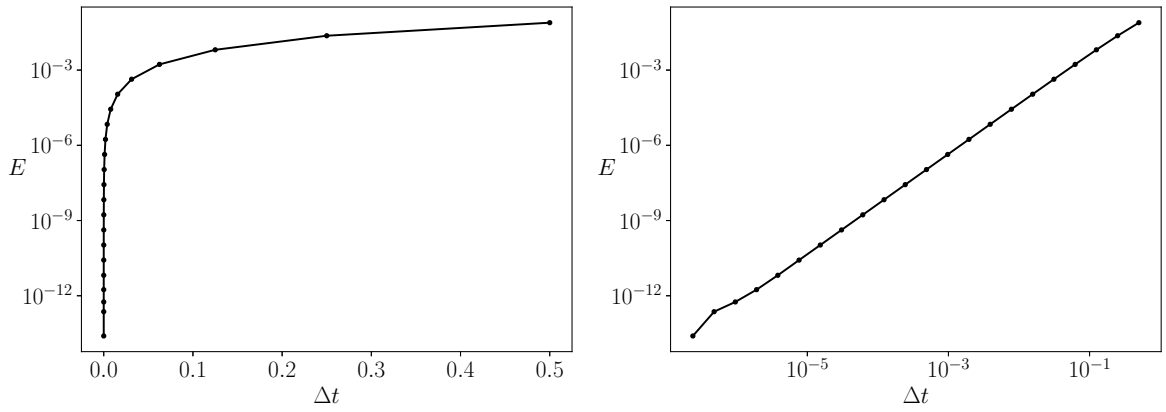

図 4　ホイン法を用いて計算した $E(\Delta t)$。左は片対数、右は両対数グラフ

■4 次ルンゲ・クッタ法

3.1.2　考察

どのアルゴリズムを用いて計算した $E(\Delta t)$ も、両対数グラフでは直線に近似していることがわかる。このことから、$E(\Delta t)$ の関数形は全て冪関数であることがわかる。次に、この冪関数それぞれの指数について考える。n 次のアルゴリズムでは、1 ステップあたりの数値解と厳密解の差は $O(\Delta t^{n+1})$ であるので、$N = \dfrac{1}{\Delta t}$ ス

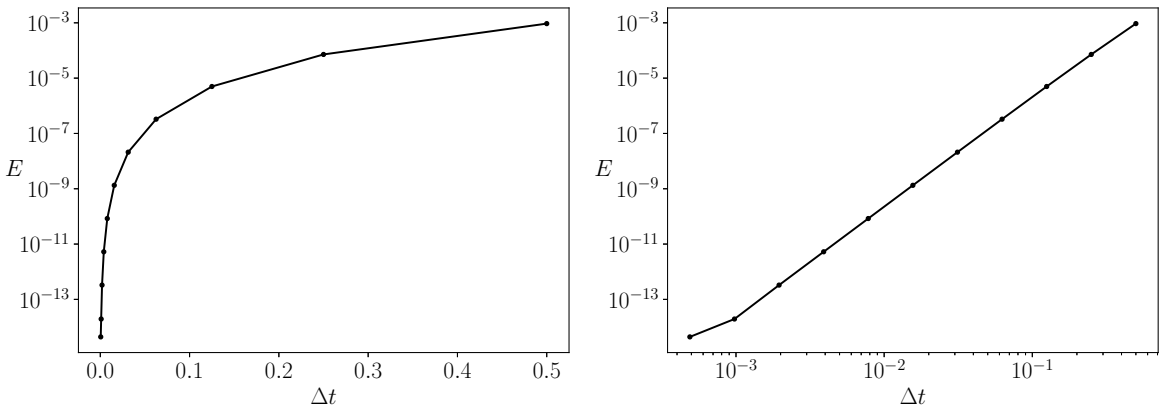

図 5　4 次ルンゲ・クッタ法を用いて計算した $E(\Delta t)$。左は片対数、右は両対数グラフ

テップ計算した際の差は、

$$O(N \times \Delta t^{n+1}) = O\left(\frac{1}{\Delta t} \times \Delta t^{n+1}\right) = O(\Delta t^n) \tag{3.3}$$

になると予想できる。

図 6 に、両対数グラフに上のグラフをまとめたものを示す。それぞれのアルゴリズムで計算した $E(\Delta t)$

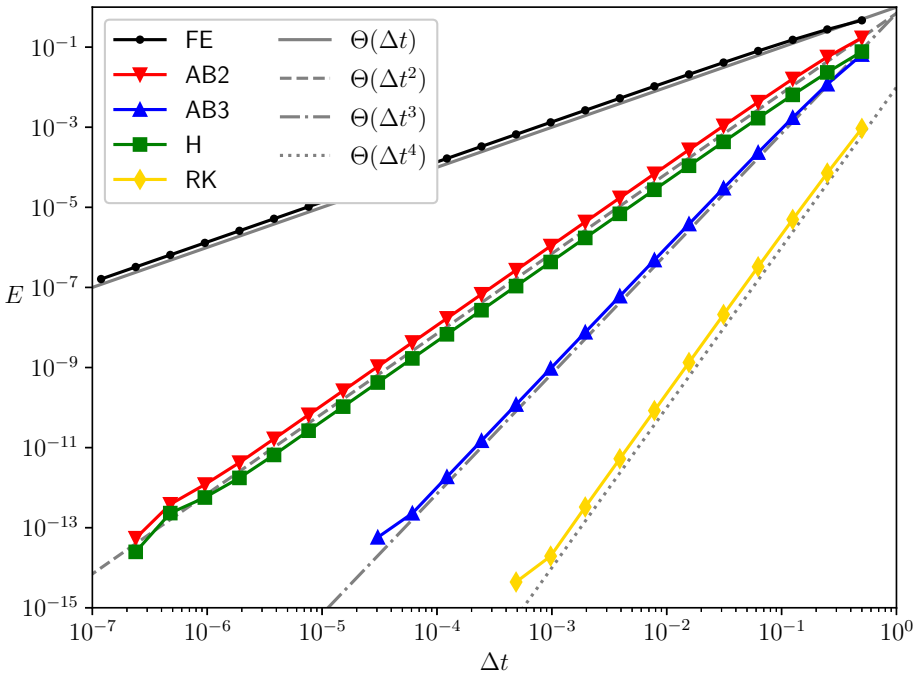

図 6　それぞれのアルゴリズムで計算した $E(\Delta t)$ (両対数グラフ)。$E(\Delta t)$ の他に、$n = 1, 2, 3, 4$ について、$\Theta(\Delta t^n)$ のグラフを重ねた。

と、$\Theta(\Delta t^n)$ の傾きを比較することで $E(\Delta t)$ の指数を観測することができる。例えば、ホイン法の $E(\Delta t)$ と $\Theta(\Delta t^2)$ は傾きが等しいため、定数 A を用いて $E(\Delta t) = A\Delta t^2$ と書けることがわかる。これを、式 (3.3) に示した予測とともにまとめると表 1 のようになり、予測が正しかったことがわかる。

表 1　$E(\Delta t)$ の指数の予測と計算結果

アルゴリズム	アルゴリズムの次数	予測される指数	実際の指数
オイラー法	1	1	1
2 次アダムス・バッシュフォース法	2	2	2
3 次アダムス・バッシュフォース法	3	3	3
ホイン法	2	2	2
4 次ルンゲ・クッタ法	4	4	4

3.1.3　結論

本課題で用いた 5 つのアルゴリズムでは、$E(\Delta t)$ の関数形は全て冪関数となり、その指数はアルゴリズムの次数と等しくなることがわかった。

3.2　課題 4 陰・陽解法の安定性

微分方程式の数値解法の安定性について考える。ここで、アルゴリズムが安定であることと、$\lim_{t\to\infty} |u(t)| \neq \infty$ となることは同じとする。アルゴリズムの差分式が式 (3.4) の漸化式で表される場合を考える。

$$u_{n+1} = au_n + b \ (a \neq 1) \tag{3.4}$$

これの一般項は、u の初期値 u_0 を用いて、

$$u_n = \left(u_0 - \frac{b}{1-a}\right) a^n + \frac{b}{1-a} \tag{3.5}$$

で与えられる [3]。アルゴリズムが安定であるとき、$\lim_{n\to\infty} u_n \neq \infty$ であるので、$|a| < 1$ であればアルゴリズムは安定であるといえる。

これを踏まえて、式 (3.6) に示す常微分方程式における以下に示すアルゴリズムの安定性を調べる。

$$\frac{\mathrm{d}u}{\mathrm{d}t} = -\alpha u + \beta,\ u(0) = u_0,\ \alpha > 0, \beta \in \mathbb{R} \tag{3.6}$$

- クランク・ニコルソン法 (陰解法)
- ホイン法 (陽解法)

3.2.1　クランク・ニコルソン法の安定性

式 (2.3) に式 (3.6) を代入すると、

$$\begin{aligned} u_{n+1} &\approx u_n + \frac{(-\alpha u_n + \beta) + (-\alpha u_{n+1} + \beta)}{2}\Delta t \\ \rightarrow u_{n+1} &\approx \frac{2 - \alpha\Delta t}{2 + \alpha\Delta t} u_n + \frac{2\beta}{2 + \alpha\Delta t}\Delta t \end{aligned}$$

を得る。上に示したように、$\left|\dfrac{2-\alpha\Delta t}{2+\alpha\Delta t}\right| < 1$ となればアルゴリズムは安定といえる。仮定より $\alpha, \Delta t > 0$ であるため、これは常に満たされ、このアルゴリズムは常に安定であるといえる。

3.2.2 ホイン法の安定性

式 (2.6) に式 (3.6) を代入すると、

$$u_{n+1} \approx u_n + \frac{(-\alpha u_n + \beta) + [-\alpha\{u_n + (-\alpha u_n + \beta)\Delta t\} + \beta]}{2}\Delta t$$
$$\rightarrow u_{n+1} \approx \left(1 - \alpha\Delta t + \frac{\alpha^2\Delta t^2}{2}\right) u_n + \left(1 - \frac{\alpha\Delta t}{2}\right)\beta\Delta t$$

を得る。3.2.1 節と同様に、$\left|1 - \alpha\Delta t + \dfrac{\alpha^2\Delta t^2}{2}\right| < 1$ となればアルゴリズムは安定といえる。式変形を行うと、

$$\left|1 - \alpha\Delta t + \frac{\alpha^2\Delta t^2}{2}\right| < 1$$
$$\rightarrow 0 < (2 - \alpha\Delta t)\alpha\Delta t < 4 \tag{3.7}$$

となるため、$0 < (2 - \alpha\Delta t)\alpha\Delta t < 4$ を満たす Δt について、アルゴリズムは安定であるといえる。

3.2.3 数値計算による確認

$u_0 = 1, \alpha = 10, \beta = 1$ として、上で述べた結果を数値計算により確認する。計算した結果を図 7, 8 に示す。

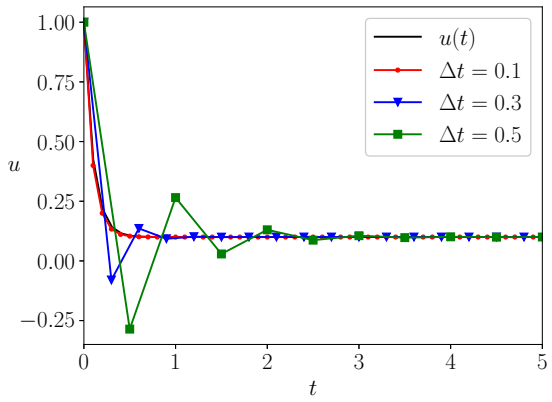

図 7　クランク・ニコルソン法による式 (3.6) の数値計算結果。$\Delta t = 0.1, 0.3, 0.5$ の場合の結果と、理論値をプロットした。

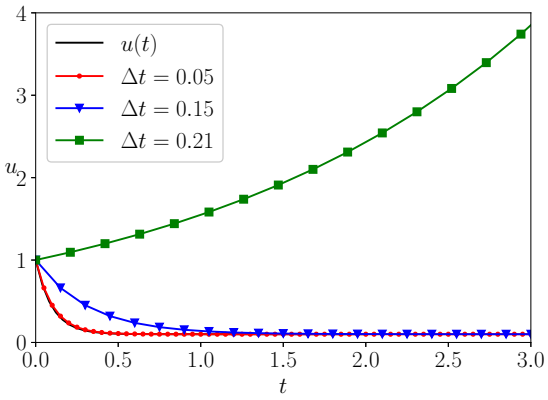

図 8　ホイン法による数値計算結果。式 (3.6) の数値計算結果。$\Delta t = 0.05, 0.15, 0.21$ の場合の結果と、理論値をプロットした。

3.2.4 考察

3.2.1 節での予測通り、クランク・ニコルソン法は、Δt を増大させても安定なアルゴリズムであることが図 7 より見てとれる。一方、図 8 を見ると、ホイン法は $\Delta t = 0.21$ の際に計算結果が発散し、不安定になっている。

式 (3.7) に 3.2.3 節で示したパラメータを代入すると、

$$\begin{aligned} &0 < (2 - 10\Delta t) \times 10\Delta t < 4 \\ \rightarrow\ &0 < (0.2 - \Delta t) \times \Delta t < 0.04 \\ \rightarrow\ &0 < \Delta t < 0.2 \end{aligned}$$

を得る。実際に、$\Delta t = 0.21 > 0.2$ で発散が起こっているため、3.2.3 節での予測は正しかったことになる。

3.2.5 結論

陰解法であるクランク・ニコルソン法では Δt を変化させてもアルゴリズムが不安定になることはなかったが、陽解法であるホイン法を用いると、Δt が一定の大きさを超えた際に不安定になることがわかった。

3.3 課題 5 アルゴリズムの安定性

式 (3.8) で表される方程式を考える。

$$\frac{\mathrm{d}u}{\mathrm{d}t} = -2u + 1 \tag{3.8}$$

これの厳密解を求める。式変形を行うと、

$$\begin{aligned} \frac{\mathrm{d}u}{\mathrm{d}t} &= -2u + 1 \\ \rightarrow \frac{\mathrm{d}u}{-2u+1} &= \mathrm{d}t \end{aligned}$$

となる。両辺を積分すると、

$$\begin{aligned} \int \frac{\mathrm{d}u}{-2u+1} &= \int \mathrm{d}t \\ \rightarrow -\frac{1}{2}\ln|-2u+1| &= t + C \ (C \text{ は定数}) \\ \rightarrow -2u + 1 &= \pm e^{-2t-2C} \\ &= C' e^{-2t} \ (C' \coloneqq \pm e^{-2C}) \\ \rightarrow u &= -\frac{C' e^{-2t} - 1}{2} \\ &= C'' e^{-2t} + \frac{1}{2} \ \left(C'' \coloneqq -\frac{C'}{2}\right) \end{aligned}$$

を得る。これの初期値 $u(0)$ は C'' によって決まるが、$t \to \infty$ において $\left|e^{-2t}\right| \to \infty$ であるので、u は任意の初期値について $t \to \infty$ で $\dfrac{1}{2}$ に収束する。

3.3.1 アルゴリズムの安定性

式 (3.9) に示すアルゴリズムを用いて式 (3.8) に示した方程式の解を数値計算することを考える。

$$u_n = u_{n-2} + 2f(t_{n-1}, u_{n-1})\Delta t \tag{3.9}$$

これに式 (3.8) を代入すると、

$$\begin{aligned} u_n &= u_{n-2} + 2(-2u_{n-1} + 1)\Delta t \\ \rightarrow u_n &= u_{n-2} - 4u_{n-1}\Delta t + 2\Delta t \end{aligned}$$

を得る。これの一般項は、定数 C_1, C_2 を用いて、

$$u_n = C_1 \left(-\sqrt{4\Delta t^2+1} - 2\Delta t\right)^n + C_2 \left(\sqrt{4\Delta t^2+1} - 2\Delta t\right)^n + \frac{1}{2}$$

と表せる [4]。$n \to \infty$ について、これが安定なアルゴリズムであるためには、

$$\begin{cases} \left|-\sqrt{4\Delta t^2+1} - 2\Delta t\right| < 1 \\ \left|\sqrt{4\Delta t^2+1} - 2\Delta t\right| < 1 \end{cases} \tag{3.10}$$

を同時に満たす必要がある。

式 (3.10) を変形すると、

$$\begin{aligned} &\left|-\sqrt{4\Delta t^2+1} - 2\Delta t\right| < 1 \\ \to &-1 < -\sqrt{4\Delta t^2+1} - 2\Delta t < 1 \\ \to &-1 + 2\Delta t < -\sqrt{4\Delta t^2+1} < 1 + 2\Delta t \end{aligned}$$

となる。右の不等号については、$\Delta t > 0$ より、常に成り立つ。左の不等式は、同様の理由により $\Delta t \geq 0.5$ の場合に成り立たないことがわかる。$\Delta t < 0.5$ の場合を考えると、

$$\begin{aligned} &-1 + 2\Delta t < -\sqrt{4\Delta t^2+1} \\ \to (-1+2\Delta t)^2 = 1 - 4\Delta t + 4\Delta t^2 &> 4\Delta t^2 + 1 \\ &\to -4\Delta t > 0 \end{aligned}$$

となり、これを満たす Δt は存在しないことがわかる。よって、式 (3.9) に示したアルゴリズムでは式 (3.8) の方程式の安定解を得ることは不可能である。

3.3.2 数値計算による確認

実際に、式 (3.9) に示したアルゴリズムを用いて式 (3.8) の方程式の解を Δt を変化させながら数値計算した様子を図 9 に示す。初期値は $u_0 = 1$ とした。

図より、Δt を小さくすれど、t が増加するにつれて数値解が発散していることがわかり、3.3.1 節で述べた予測が正しかったといえる。

3.3.3 結論

式 (3.9) に示したアルゴリズムを用いた場合、式 (3.8) の方程式の安定解を得ることは不可能であることがわかった。

3.4 課題 7 ローレンツ方程式

式 (3.11) で表されるローレンツ方程式を考える。

$$\begin{cases} \dfrac{\mathrm{d}x}{\mathrm{d}t} = \sigma(y - x) \\ \dfrac{\mathrm{d}y}{\mathrm{d}t} = rx - y - xz \\ \dfrac{\mathrm{d}z}{\mathrm{d}t} = xy - bz \end{cases} \tag{3.11}$$

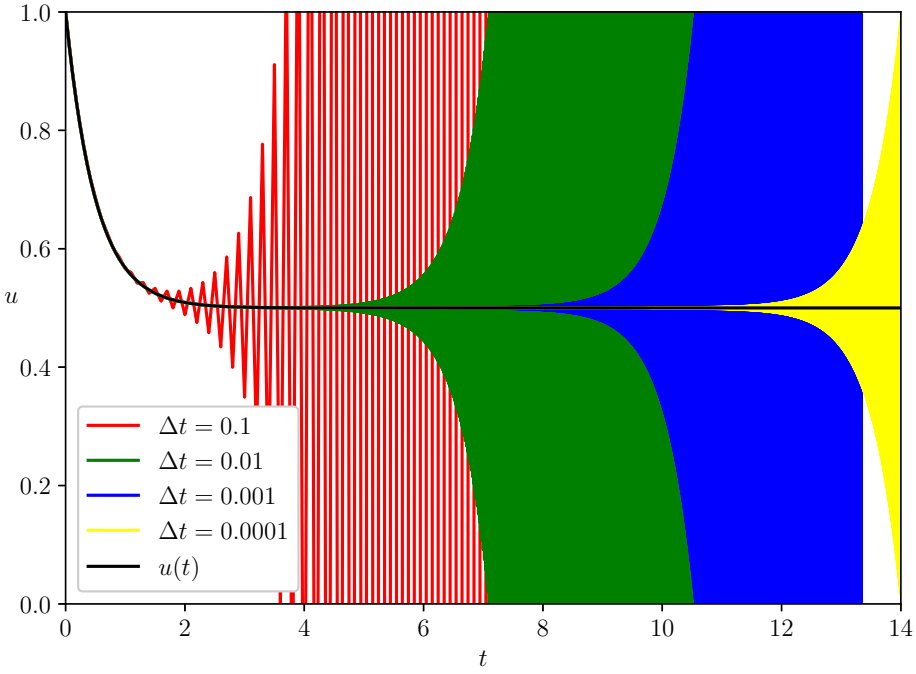

図 9　式 (3.8) の方程式の数値解。$\Delta t = 0.1, 0.01, 0.001, 0.0001$ の場合と、厳密解をプロットした。

初期値を $(x(0), y(0), z(0)) = (1 + \epsilon, 0, 0)$, 定数を $\sigma = 10, b = \dfrac{8}{3}, r = 28$ とする。以下では、ステップ幅 $\Delta t = 0.01$ の 4 次ルンゲ・クッタ法を用いて、$t \in [0, 100]$ の区間について数値計算を行う。

3.4.1　ローレンツ方程式の描画

$\epsilon = 0$ とし、$x(t)$ および、$(x(t), y(t), z(t))$ の xyz 空間への描画を行う。結果を図 10, 11 に示す。

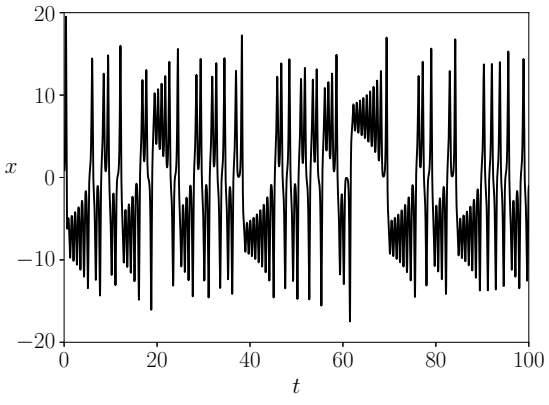

図 10　$x(t)$ の数値計算結果

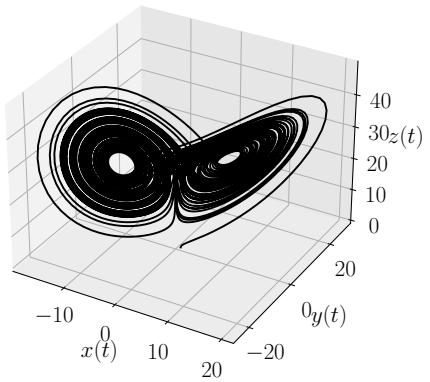

図 11　$(x(t), y(t), z(t))$ の数値計算結果

3.4.2　ϵ の変化による軌道の変化

$\epsilon = 0$ の場合と $\epsilon = 10^{-n}$ $(n = 2, 4, 6)$ の場合の解をそれぞれ

$$\boldsymbol{x}_0(t) = (x_0(t), y_0(t), z_0(t))$$
$$\boldsymbol{x}_n(t) = (x_n(t), y_n(t), z_n(t))$$

とし、これらの差

$$\Delta_n(t) \coloneqq \|\boldsymbol{x}_0(t) - \boldsymbol{x}_n(t)\|$$

を定義する。$\Delta_n(t)$ のグラフを作成し、これが漸近値に至る少し前の時間帯の関数形を調べる。

3.4.3　結果

$\Delta_n(t)$ の計算結果を図 12, 13 に示す。

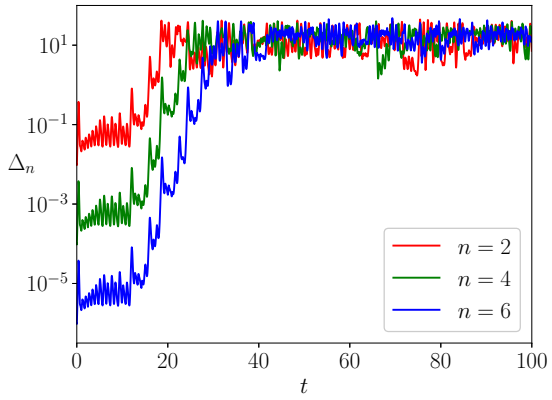

図 12　$\Delta_n(t)$ の計算結果 (片対数グラフ)

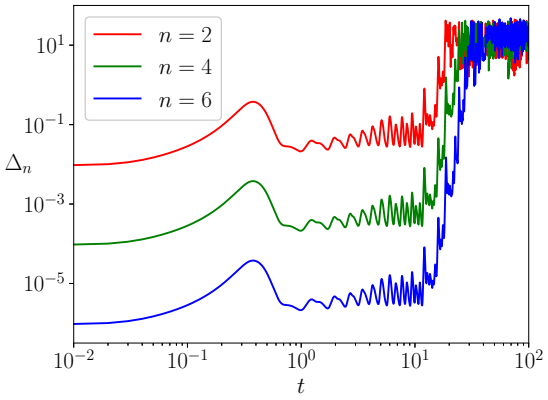

図 13　$\Delta_n(t)$ の計算結果 (両対数グラフ)

3.4.4　考察

$\Delta_n(t)$ の関数形を調べるために、冪関数と指数関数をそれぞれ図 12, 13 に重ねたグラフを図 14〜17 に示す。 図 15, 16 を見ると、冪関数と指数関数のどちらも綺麗に重なっているように見える。しかし、図 14 を見ると、n が増加するにつれ、漸近値付近で $\Delta_n(t)$ と冪関数の差が開いていることがわかる。一方、図 17 を見ると、指数関数は n が変化しても重なったままである。

以上より、$\Delta_n(t)$ の漸近値直前の関数形は指数関数であり、その指数は 0.8 であることがわかった。

3.4.5　結論

$\Delta_n(t)$ の漸近値直前の関数形は n に依存せず、$e^{-0.8t}$ である。

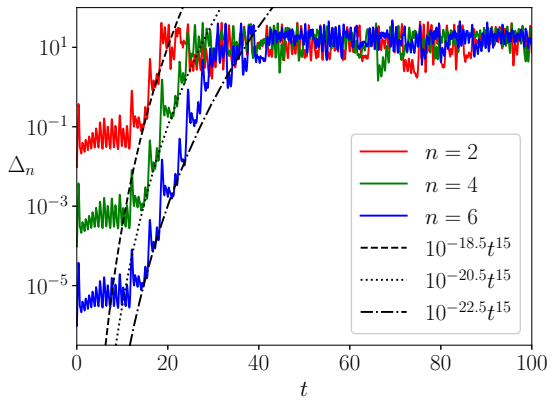

図 14　$\Delta_n(t)$ に冪関数を重ねた図 (片対数グラフ)

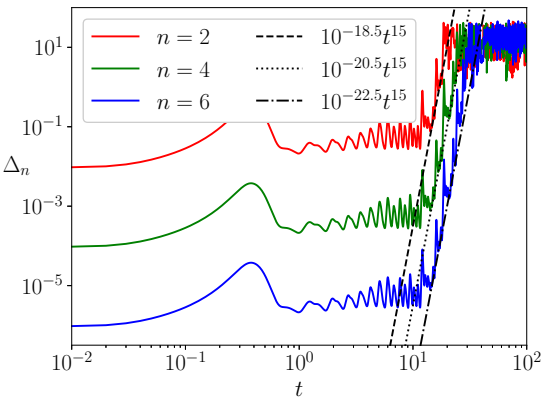

図 15　$\Delta_n(t)$ に冪関数を重ねた図 (両対数グラフ)

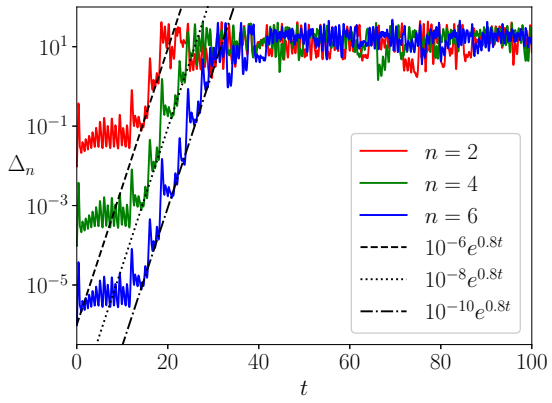

図 16　$\Delta_n(t)$ に指数関数を重ねた図 (片対数グラフ)

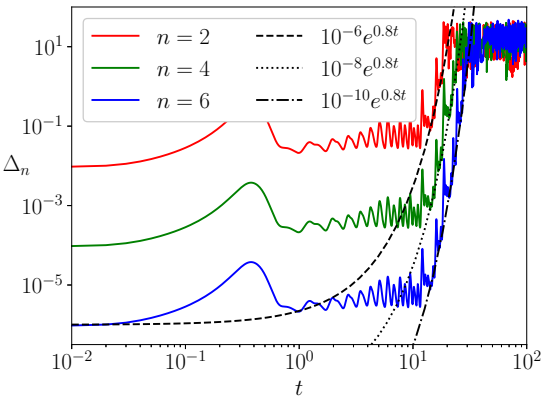

図 17　$\Delta_n(t)$ に指数関数を重ねた図 (両対数グラフ)

3.5　課題 8 連立微分方程式

式 (3.12) で表される連立微分方程式を考える。

$$\frac{\mathrm{d}x_i}{\mathrm{d}t} = \omega_i - \frac{K}{N}\sum_{j=1}^{N}\sin(x_i - x_j) \tag{3.12}$$
$$\omega_i \coloneqq \tan\left\{\pi\left(\frac{i}{N+1} - \frac{1}{2}\right)\right\}\ (i = 1, 2, \cdots, N)$$

ここで、K は定数である。また、初期値は次式で与えられる。

$$x_i(0) = y_i + 0.01\sin y_i,\ y_i \coloneqq \frac{i-1}{N}2\pi\ (i = 1, 2, \cdots, N)$$

また、式 (3.13) で定義される値 $R(t)$ を考える。

$$R(t) = \sqrt{R_x^2 + R_y^2},\ R_x \coloneqq \frac{1}{N}\sum_{j=1}^{N}\cos x_j,\ R_y \coloneqq \frac{1}{N}\sum_{j=1}^{N}\sin x_j \tag{3.13}$$

ステップ幅 $\Delta t = 0.01$ の 4 次ルンゲ・クッタ法を用いて数値計算を行う。$N = 10^2, 10^3$ に対して、$t \in [50, 100]$ の区間において $R(t)$ の時間平均を以下で定義する。

$$\bar{R} \coloneqq \frac{1}{50} \int_{50}^{100} R(t) dt$$

K を区間 $[1, 3]$ 内で 0.01 刻みで動かし、$\bar{R}$ を K の関数としてプロットする。

3.5.1 計算量の削減

式 (3.12) をそのまま計算すると、$O(N^2)$ の時間計算量を要するが、$O(N)$ で計算する方法を述べる。

右辺第 2 項の総和の中身に加法定理を適用すると、

$$\begin{aligned}
\frac{K}{N} \sum_{j=1}^{N} \sin(x_i - x_j) &= \frac{K}{N} \sum_{j=1}^{N} (\sin x_i \cos x_j - \cos x_i \sin x_j) \\
&= K \left(\sin x_i \frac{1}{N} \sum_{j=1}^{N} \cos x_j - \cos x_i \frac{1}{N} \sum_{j=1}^{N} \sin x_j \right) \\
&= K(R_x \sin x_i - R_y \cos x_i)
\end{aligned}$$

となり、式 (3.12) は

$$\frac{\mathrm{d}x_i}{\mathrm{d}t} = \omega_i - K(R_x \sin x_i - R_y \cos x_i)$$

と書ける。これは R_x, R_y の前計算を行うことで $O(1)$ で計算できるため、全体では $O(N)$ の時間計算量となる。

3.5.2 結果

計算した $\bar{R}(K)$ を図 18 に示す。

3.5.3 考察

$N \to \infty$ とした際、$K = 2$ は臨界点と呼ばれている。その理由を以下に考察する。

x_i を位相とすると、xy 平面における単位円上のその位相を持つ点の座標は $(\cos x_i, \sin x_i)$ と書ける。よって、式 (3.13) より、R_x, R_y はそれぞれ N 点の x, y 座標の平均であるといえ、R は平均座標のノルムである。

以上を踏まえると、x_i の分散が大きいほど R は 0 に近づき、分散が小さいほど R は 1 に近づくことがわかる。図 19, 20 に $N = 10^2$ について、$K = 1, 3$ のときの単位円上の N 点 $(\cos x_i(100), \sin x_i(100))$ の分布を示す。図 18 と図 19, 20 を見比べると、$\bar{R} \approx 0$ である $K = 1$ のときは実際に x_i の分散が大きく、逆に $\bar{R}$ が増加している $K = 3$ のときは x_i の分散が小さいことがわかる。

また、図 18 より、$N = 10^2, 10^3$ のどちらについても、$K \leq 2$ のときは $R \approx 0$ で、$K > 2$ のときに R が増加していることわかり、これが $K = 2$ が臨界点と呼ばれる要因であると考えられる。

4 付録

本節には 3 節で作成した c++ コードを一部抜粋して記述する。

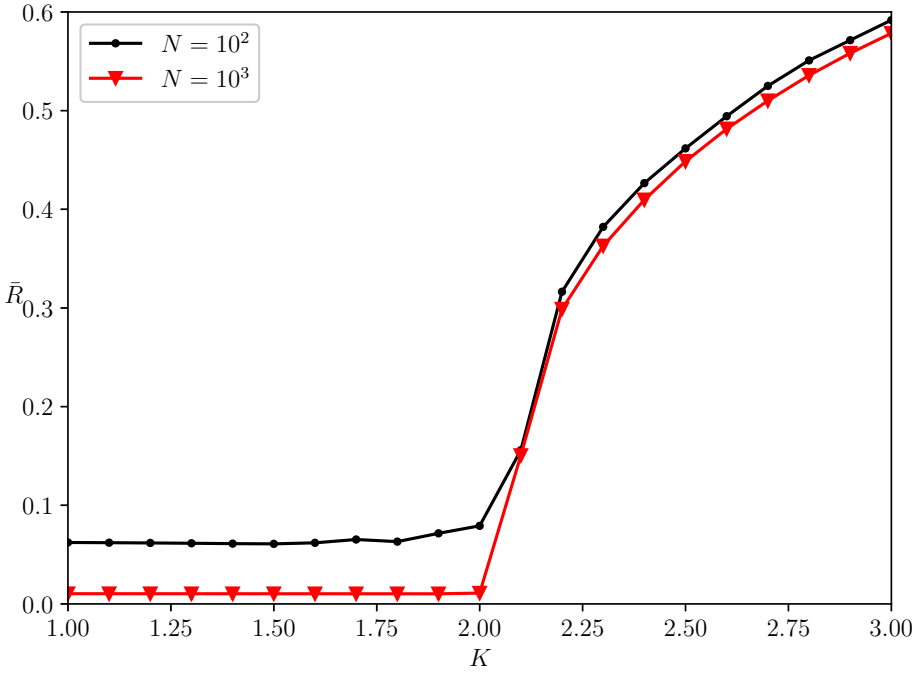

図 18　計算した $\bar{R}(K)$。

4.1　課題 3

コード 1　前進オイラー法

```
const double euler(
    const double (*f)(double , double), // 被積分関数
    double u, // 初期値
    const double dt, // ステップ幅
    const double start, const double end // 積分範囲
) {
    for (double t = start; t < end; t += dt)
        u += f(t, u) * dt;

    return u;
}
```

コード 2　2 次アダムス・バッシュフォース法

```
const double adams_second(
    const double (*f)(double , double), // 被積分関数
    const double (*uexact)(double), // 厳密解
    double u, // 初期値
    const double dt, // ステップ幅
    const double start, const double end // 積分範囲
) {
    double lastf = uexact(start - dt);
    for (double t = start; t < end; t += dt) {
```

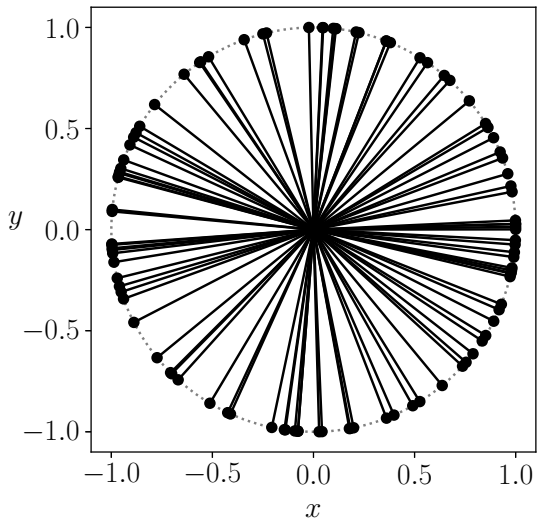

図 19　$K = 1$ のときの x_i $(i \in 1, 2, \cdots, N)$ の分布

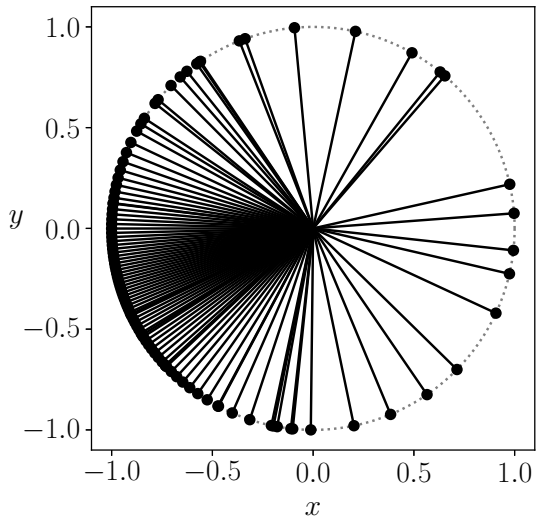

図 20　$K = 3$ のときの x_i $(i \in 1, 2, \cdots, N)$ の分布

```
        const double nowf = f(t, u);
        u += (3.0 * nowf - lastf) * dt / 2.0;

        lastf = nowf;
    }

    return u;
}
```

コード 3　3 次アダムス・バッシュフォース法

```
const double adams_third(
    const double (*f)(double , double), // 被積分関数
    const double (*uexact)(double), // 厳密解
    double u, // 初期値
    const double dt, // ステップ幅
    const double start, const double end // 積分範囲
) {
    double lastf = uexact(start - dt), lastlastf = uexact(start - dt - dt);
    for (double t = start; t < end; t += dt) {
        const double nowf = f(t, u);
        u += (23.0 * nowf - 16.0 * lastf + 5.0 * lastlastf) * dt / 12.0;

        lastlastf = lastf; lastf = nowf;
    }

    return u;
}
```

コード 4　ホイン法

```
const double heun(
    const double (*f)(double , double), // 被積分関数
```

```
    double u, // 初期値
    const double dt, // ステップ幅
    const double start, const double end // 積分範囲
) {
    for (double t = start; t < end; t += dt)
        u += (f(t, u) + f(t + dt, u + f(t, u) * dt)) * dt / 2.0;

    return u;
}
```

コード 5　4 次ルンゲ・クッタ法

```
const double runge_kutta(
    const double (*f)(double , double), // 被積分関数
    double u, // 初期値
    const double dt, // ステップ幅
    const double start, const double end // 積分範囲
) {
    for (double t = start; t < end; t += dt) {
        const double f1 = f(t, u);
        const double f2 = f(t + dt / 2.0, u + f1 * dt / 2.0);
        const double f3 = f(t + dt / 2.0, u + f2 * dt / 2.0);
        const double f4 = f(t + dt, u + f3 * dt);

        u += (f1 + 2 * f2 + 2 * f3 + f4) * dt / 6.0;
    }

    return u;
}
```

4.2　課題 4

コード 6　クランク・ニコルソン法

```
constexpr double ALPHA = 10;
constexpr double BETA = 1;

void crank(
    const double (*f)(double , double), // 被積分関数
    double u, // 初期値
    const double dt, // ステップ幅
    const double start, const double end // 積分範囲
) {
    double t;
    for (t = start; t < end; t += dt) {
        u = ((2.0 - ALPHA * dt) * u + 2.0 * BETA * dt) / (2.0 + ALPHA * dt);
    }
}
```

ホイン法はコード 4 と同じため、略。

4.3　課題 5

コード 7　式 (3.9) による数値計算

```
void simulate(
    const double (*f)(double , double), // 被積分関数
    const double (*uexact)(double), // 厳密解
    double u, // 初期値
    const double dt, // ステップ幅
    const double start, const double end // 積分範囲
) {
    double lastu = uexact(start - dt);
    for (double t = start; t < end; t += dt) {
        const double utemp = u;
        u = lastu + 2 * dt * f(t, u);

        lastu = utemp;
    }
}
```

4.4　課題 7

コード 8　4 次ルンゲ・クッタ法によるローレンツ方程式の数値計算

```
void runge_kutta(
    const double (*fx)(double , double, double),
    const double (*fy)(double , double, double),
    const double (*fz)(double , double, double), // 被積分関数
    double x, double y, double z, // 初期値
    const double dt, // ステップ幅
    const double start, const double end // 積分範囲
) {
    for (double t = start; t < end; t += dt) {
        const double fx1 = fx(x, y, z);
        const double fy1 = fy(x, y, z);
        const double fz1 = fz(x, y, z);

        const double fx2 = fx(x + fx1 * dt / 2.0, y + fy1 * dt / 2.0, z + fz1 * dt / 2.0);
        const double fy2 = fy(x + fx1 * dt / 2.0, y + fy1 * dt / 2.0, z + fz1 * dt / 2.0);
        const double fz2 = fz(x + fx1 * dt / 2.0, y + fy1 * dt / 2.0, z + fz1 * dt / 2.0);

        const double fx3 = fx(x + fx2 * dt / 2.0, y + fy2 * dt / 2.0, z + fz2 * dt / 2.0);
        const double fy3 = fy(x + fx2 * dt / 2.0, y + fy2 * dt / 2.0, z + fz2 * dt / 2.0);
        const double fz3 = fz(x + fx2 * dt / 2.0, y + fy2 * dt / 2.0, z + fz2 * dt / 2.0);

        const double fx4 = fx(x + fx3 * dt, y + fy3 * dt, z + fz3 * dt);
        const double fy4 = fy(x + fx3 * dt, y + fy3 * dt, z + fz3 * dt);
        const double fz4 = fz(x + fx3 * dt, y + fy3 * dt, z + fz3 * dt);

        x += (fx1 + 2 * fx2 + 2 * fx3 + fx4) * dt / 6.0;
        y += (fy1 + 2 * fy2 + 2 * fy3 + fy4) * dt / 6.0;
        z += (fz1 + 2 * fz2 + 2 * fz3 + fz4) * dt / 6.0;
    }
}
```

4.5　課題 8

コード 9　4 次ルンゲ・クッタ法による $\bar{R}(K)$ の計算

```cpp
#include <cmath>
#include <vector>

double runge_kutta(
    const double (*f)(double, int, double, int, double, double), // 被積分関数
    std::vector<double> x, // 初期値
    const double K, // 定数
    const int N, // 元数
    const double dt, // ステップ幅
    const double start, const double end, // 積分範囲
    double calc_t // 平均の計算開始時刻
) {
    double Rx, Ry;
    double Rsum = 0;
    for (double t = start; t < end; t += dt) {
        Rx = Ry = 0;
        for (int i = 0; i < N; i++) {
            Rx += cos(x[i]) / N;
            Ry += sin(x[i]) / N;
        }

        for (int i = 0; i < N; i++) {
            const double f1 = f(x[i], i, K, N, Rx, Ry);
            const double f2 = f(x[i] + f1 * dt / 2.0, i, K, N, Rx, Ry);
            const double f3 = f(x[i] + f2 * dt / 2.0, i, K, N, Rx, Ry);
            const double f4 = f(x[i] + f3 * dt, i, K, N, Rx, Ry);

            x[i] += (f1 + 2 * f2 + 2 * f3 + f4) * dt / 6.0;
        }

        if (t >= calc_t) {
            Rsum += sqrt(Rx * Rx + Ry * Ry) * dt;
        }
    }

    return Rsum / (end - calc_t);
}

void calculate_Rbar(
    int N // 元数
) {
    const double start = 0, end = 100, calc_t = 50, dt = 0.01;
    std::vector<double> x(N);
    for (int i = 0; i < N; i++) {
        const double y = 2 * M_PI / N * (i - 1);
        x[i] = y + 0.01 * sin(y);
    }

    for (double K = 1; K <= 5 + 1e-7; K += 0.1)
        const double R = runge_kutta(f, x, K, N, dt, start, end, calc_t);
}
```

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)

[2] 斉藤宣一、“数値解析入門”、東京大学出版 (2012)

[3] 高校数学の美しい物語、“$f(n)$ を含む二項間漸化式の 2 通りの解法”、“https://manabitimes.jp/math/687”, (最終更新: 2021/3/7)

[4] 高校数学の美しい物語、“三項間漸化式の 3 通りの解き方”、“https://manabitimes.jp/math/697”, (最終更新: 2021/3/7)