

# 数理工学実験レポート

## 第4章（常微分方程式の数値的解法）

学籍番号 1029366161 中塚一瑳

2025年11月16日

### 概要

## 目次

1	はじめに	3
2	課題1 身の回りの現象の定式化	3
2.1	現象の説明	3
2.2	モデル化のための仮定	3
2.3	常微分方程式の初期値問題	4
2.4	解析解	4
2.5	パラメータ $\beta$ の解釈と考察	5
3	課題2 4次のアダムス・バッシュフォース法とアダムス・ムルトン法の導出	6
3.1	4次アダムス・バッシュフォース法の導出	6
3.2	4次アダムス・ムルトン法の導出	10
3.3	4次予測子・修正子法	11
4	課題3 指数関数の微分方程式の数値解法	11
4.1	原理・方法	11
4.2	結果	15
4.3	考察	15
5	課題4 各手法の安定性と安定性な領域の確認	16
5.1	原理・方法	16
5.2	解析的結果のまとめ	20
5.3	結果	20
5.4	考察	22
6	課題5 不安定性の確認	23
6.1	問題設定	23

6.2	差分スキームの形	24
6.3	特性方程式による安定性解析	24
6.4	数値解析による結果	25
7	課題 6 爆発解の数値計算と弧長パラメータによる変数変換	27
7.1	原理と方法	27
7.2	数値解法の実装	27
7.3	結果	28
8	課題 7 ローレンツ方程式の数値解法とカオス現象の観察	30
8.1	原理, 方法	30
8.2	結果	31
8.3	考察	35
9	結論	35
A	付録 A 使用コード一覧	35
A.1	課題 3 のコード	35
A.2	課題 4 のコード	37
A.3	課題 5 のコード	39
A.4	課題 6 のコード	40
A.5	課題 7 のコード	43

# 1 はじめに

本レポートでは、常微分方程式の数値的解法について、オイラー法、ルンゲ・クッタ法、アダムス法などの代表的な手法を実装し、その精度や安定性を検証する。さらに、ローレンツ方程式を用いたカオス現象の観察を通じて、数値解法の応用と限界について考察を行う。

## 2 課題 1 身の回りの現象の定式化

### 2.1 現象の説明

本課題では、身の回りの現象として SNS (X, 旧 Twitter) におけるツイートの「バズり方」を取り上げる。あるユーザが投稿したツイートが、他のユーザによるいいねやリポスト（拡散）によって時間とともに広がっていく様子を、常微分方程式の初期値問題としてモデル化する。

ここではさらに、「どのくらいの界限を越えて受け入れられるツイートか」を表すパラメータとして  $\beta$  を導入する。一般受けしやすいツイートほど、フォロワーの界限から離れても反応が落ちにくく、身内ネタや日常報告のようなツイートほど、界限が少し離れるだけで反応しにくくなる、という直感を数式に反映させることを目指す。

### 2.2 モデル化のための仮定

ツイートの広がり方を簡単に記述するために、次のような仮定をおく。

- 時刻  $t$  をツイート投稿からの経過時間とする。
- $x(t)$  を、時刻  $t$  までにそのツイートに対していいねまたはリポストしたユーザの累積人数と定める。
- すでに反応したユーザは、界限のそこからさらに外側の界限へとツイートを伝播させる。界限が遠くなるほど反応しにくくなる効果を、 $x$  に対する幕  $x^{1-\beta}$  の形で表し、 $0 < \beta \leq 1$  とする。
  - $\beta \approx 0$ ：動物系、ニュース・政治系など界限を問わず受け入れられやすい「一般受け」ツイート。
  - $\beta \approx 1$ ：身内ネタや日常報告など、特定の界限にしか届きにくいツイート。
- ツイートを見たユーザがいいね／リポストする確率は、界限の内部では一定であり、界限内外で平均すると、 $x^{1-\beta}$  に比例する総合的な効果にまとめられると仮定する。
- アルゴリズムの影響により、ツイートは時間が経つほどおすすめやタイムラインに表示されにくくなる。その効果を「実効的な拡散力」 $k(t)$  が指数関数的に減衰する

$$k(t) = k_0 e^{-\alpha t} \quad (k_0 > 0, \alpha > 0)$$

として表現する。

- ツイートを最終的に見うる潜在的ユーザ数  $N$  は非常に大きく、実際の反応人数  $x(t)$  は  $x(t) \ll N$  の範囲にとどまるとして仮定し、飽和効果は無視する。

これらの仮定のもとで、時刻  $t$  における「新たに反応する人数の増加率」は

$$\{すでに反応した人数\}^{1-\beta} \times \{\text{時間に応じた拡散力}\} \approx k(t) x(t)^{1-\beta}$$

で与えられると考える。

### 2.3 常微分方程式の初期値問題

以上より、反応人数  $x(t)$  の時間発展は次の常微分方程式でモデル化できる：

$$\frac{dx}{dt} = k(t) x(t)^{1-\beta} = k_0 e^{-\alpha t} x(t)^{1-\beta}, \quad 0 < \beta \leq 1, k_0 > 0, \alpha > 0. \quad (1)$$

初期条件として、ツイート直後  $t = 0$  の時点では、投稿者本人や一部のフォロワーによる反応がすでに  $x_0$  件存在していると仮定すると、

$$x(0) = x_0 \quad (x_0 > 0) \quad (2)$$

と書ける。

したがって、ツイートのバズり方を表す初期値問題は

$$\begin{cases} \frac{dx}{dt} = k_0 e^{-\alpha t} x^{1-\beta}(t), \\ x(0) = x_0, \end{cases} \quad (3)$$

となる。

### 2.4 解析解

式 (3) は変数分離形であり、 $x > 0$  の範囲で

$$x^{\beta-1} \frac{dx}{dt} = k_0 e^{-\alpha t}$$

と書けるので、

$$x^{\beta-1} dx = k_0 e^{-\alpha t} dt$$

を積分する：

$$\int x^{\beta-1} dx = \int k_0 e^{-\alpha t} dt.$$

左辺は

$$\int x^{\beta-1} dx = \frac{x^\beta}{\beta},$$

右辺は

$$\int k_0 e^{-\alpha t} dt = -\frac{k_0}{\alpha} e^{-\alpha t} + C$$

であるから、

$$\frac{x^\beta}{\beta} = -\frac{k_0}{\alpha} e^{-\alpha t} + C.$$

初期条件  $t = 0$ ,  $x(0) = x_0$  を用いると

$$\frac{x_0^\beta}{\beta} = -\frac{k_0}{\alpha} + C$$

より

$$C = \frac{x_0^\beta}{\beta} + \frac{k_0}{\alpha}.$$

したがって

$$\frac{x^\beta}{\beta} = -\frac{k_0}{\alpha}e^{-\alpha t} + \frac{x_0^\beta}{\beta} + \frac{k_0}{\alpha} = \frac{x_0^\beta}{\beta} + \frac{k_0}{\alpha}(1 - e^{-\alpha t}),$$

すなわち

$$x(t)^\beta = x_0^\beta + \frac{\beta k_0}{\alpha}(1 - e^{-\alpha t}). \quad (4)$$

よって解は

$$x(t) = \left[ x_0^\beta + \frac{\beta k_0}{\alpha}(1 - e^{-\alpha t}) \right]^{1/\beta} \quad (5)$$

となる（括弧内が正の範囲で有効）。

特に、十分時間が経った極限  $t \rightarrow \infty$  では  $e^{-\alpha t} \rightarrow 0$  より

$$x_\infty := \lim_{t \rightarrow \infty} x(t) = \left[ x_0^\beta + \frac{\beta k_0}{\alpha} \right]^{1/\beta} \quad (6)$$

となる。これは「ツイートの寿命が尽きた後の最終的な反応数」に対応する。

## 2.5 パラメータ $\beta$ の解釈と考察

ツイートの「面白さ」や「受けやすさ」を表す指標を  $s$  とし、面白いツイートほど初期拡散力が大きくなると仮定して

$$k_0 = k_{\text{base}} s$$

とおく。ここで  $k_{\text{base}}$  は定数である。このとき (6) より、最終的な反応数は

$$x_\infty(s) = \left[ x_0^\beta + \beta K s \right]^{1/\beta}, \quad K := \frac{k_{\text{base}}}{\alpha} \quad (7)$$

と書ける。

この式から、パラメータ  $\beta$  の値によって、「面白さ  $s$  の変化が最終的な反応数にどう効くか」が大きく変わることが分かる。

### $\beta \rightarrow 0$ の極限（界隈を越えてバズりやすいツイート）

$\beta$  が十分小さいとき、 $\log x_\infty(s)$  を  $\beta$  について一次まで近似すると

$$\log x_\infty(s) = \frac{1}{\beta} \log(x_0^\beta + \beta K s) \approx \log x_0 + K s$$

となる。従って

$$x_\infty(s) \approx x_0 \exp(Ks) \quad (\beta \ll 1). \quad (8)$$

すなわち、 $\beta$  が 0 に近いツイート（界隈をまたいで受け入れられる、いわゆる一般受けするツイート）では、面白さ  $s$  が 1 増えるごとに、最終的な反応数がほぼ指数関数的に増加する。この意味で、「面白さに対して指数的に効く」モードになっていると解釈できる。

$\beta \rightarrow 1$  の極限（身内ネタ寄りのツイート）

一方,  $\beta \rightarrow 1$  とすると

$$x_\infty(s) = \left[ x_0^\beta + \beta K s \right]^{1/\beta} \xrightarrow[\beta \rightarrow 1]{} x_0 + K s$$

となり,

$$x_\infty(s) \approx x_0 + K s \quad (\beta \approx 1) \quad (9)$$

とほぼ線形に増加する。これは、身内ネタや日常報告のように、ごく限られた界隈にしか届かないツイートでは、面白さ  $s$  を上げても、反応数はほぼ比例程度しか増えないことを意味している。

## まとめ

以上より、パラメータ  $\beta$  は

- $\beta$  が 0 に近いほど「界隈を越えて受け入れられやすい」ツイートであり、面白さ  $s$  に対して最終的な反応数  $x_\infty(s)$  は式 (8) のように ほぼ指数関数的 に増加する。
- $\beta$  が 1 に近いほど「特定の界隈にしか刺さらない」ツイートであり、 $x_\infty(s)$  は式 (9) のように ほぼ線形 にしか増加しない。

このように、常微分方程式 (3) の解の形を解析することで、

一般受けするツイートほど、小さな「面白さ」の差が最終的なバズり具合を非常に大きく変えるのに対し、身内ネタに近づくほど、面白さはほぼ線形にしか効かない

という直感的な性質を、数学的に説明することができる。

## 3 課題 2 4 次のアダムス・バッシュフォース法とアダムス・ムルトン法の導出

Adams 型多段法の一般的構成に基づき、4 次のアダムス・バッシュフォース法 (Adams – Bashforth, AB4) と 4 次のアダムス・ムルトン法 (Adams – Moulton, AM4) を *Lagrange 補間の積分を明示的に行うこと* で導出する [1]。さらに、これらを組み合わせて 4 次の予測子・修正子法を構成する。

対象とする常微分方程式は

$$u' = f(t, u), \quad u(t_n) = u_n \quad (10)$$

とし、時間刻みを一定  $\Delta t$  とおく。資料と同様に

$$f_n = f(t_n, u_n)$$

と略記する。

### 3.1 4 次アダムス・バッシュフォース法の導出

まず、

$$u_n = u_{n-1} + \int_{t_{n-1}}^{t_n} f(\tau, u(\tau)) d\tau$$

を数値積分で近似することを考える。4次AB法では、区間  $[t_{n-1}, t_n]$  上の  $f$  を

$$(t_{n-1}, f_{n-1}), (t_{n-2}, f_{n-2}), (t_{n-3}, f_{n-3}), (t_{n-4}, f_{n-4})$$

の4点を通る3次Lagrange多項式で近似し、その積分をとる。

## 変数変換

等間隔刻み  $\Delta t$  を用いて

$$\tau = t_{n-1} + \theta \Delta t, \quad \theta \in [0, 1] \quad (11)$$

とおくと、

$$d\tau = \Delta t d\theta$$

より

$$\int_{t_{n-1}}^{t_n} f(\tau, u(\tau)) d\tau = \Delta t \int_0^1 f(t_{n-1} + \theta \Delta t, u(t_{n-1} + \theta \Delta t)) d\theta \quad (12)$$

$$= \Delta t \int_0^1 g(\theta) d\theta, \quad (13)$$

ただし

$$g(\theta) = f(t_{n-1} + \theta \Delta t, u(t_{n-1} + \theta \Delta t))$$

とおいた。

$\theta$  の節点は

$$t_{n-1} \leftrightarrow \theta = 0, \quad t_{n-2} \leftrightarrow \theta = -1, \quad t_{n-3} \leftrightarrow \theta = -2, \quad t_{n-4} \leftrightarrow \theta = -3$$

となる。

## Lagrange多項式の構成

節点

$$\theta_0 = 0, \quad \theta_1 = -1, \quad \theta_2 = -2, \quad \theta_3 = -3$$

に対し、

$$g(\theta_k) = f_{n-1-k} \quad (k = 0, 1, 2, 3)$$

となる。Lagrange基底多項式を

$$\ell_k(\theta) = \prod_{\substack{j=0 \\ j \neq k}}^3 \frac{\theta - \theta_j}{\theta_k - \theta_j}$$

とおくと、

$$g(\theta) \approx \ell_0(\theta)f_{n-1} + \ell_1(\theta)f_{n-2} + \ell_2(\theta)f_{n-3} + \ell_3(\theta)f_{n-4}$$

と表される。

各  $\ell_k(\theta)$  を具体的に書くと

$$\ell_0(\theta) = \frac{(\theta - \theta_1)(\theta - \theta_2)(\theta - \theta_3)}{(\theta_0 - \theta_1)(\theta_0 - \theta_2)(\theta_0 - \theta_3)} \quad (14)$$

$$= \frac{(\theta + 1)(\theta + 2)(\theta + 3)}{(0 + 1)(0 + 2)(0 + 3)} = \frac{(\theta + 1)(\theta + 2)(\theta + 3)}{6}, \quad (15)$$

$$\ell_1(\theta) = \frac{(\theta - \theta_0)(\theta - \theta_2)(\theta - \theta_3)}{(\theta_1 - \theta_0)(\theta_1 - \theta_2)(\theta_1 - \theta_3)} \quad (16)$$

$$= \frac{\theta(\theta+2)(\theta+3)}{(-1-0)(-1+2)(-1+3)} = -\frac{\theta(\theta+2)(\theta+3)}{2}, \quad (17)$$

$$\ell_2(\theta) = \frac{(\theta - \theta_0)(\theta - \theta_1)(\theta - \theta_3)}{(\theta_2 - \theta_0)(\theta_2 - \theta_1)(\theta_2 - \theta_3)} \quad (18)$$

$$= \frac{\theta(\theta+1)(\theta+3)}{(-2-0)(-2+1)(-2+3)} = \frac{\theta(\theta+1)(\theta+3)}{2}, \quad (19)$$

$$\ell_3(\theta) = \frac{(\theta - \theta_0)(\theta - \theta_1)(\theta - \theta_2)}{(\theta_3 - \theta_0)(\theta_3 - \theta_1)(\theta_3 - \theta_2)} \quad (20)$$

$$= \frac{\theta(\theta+1)(\theta+2)}{(-3-0)(-3+1)(-3+2)} = -\frac{\theta(\theta+1)(\theta+2)}{6}. \quad (21)$$

## 係数の積分計算

$$\int_0^1 g(\theta) d\theta \approx \sum_{k=0}^3 \left( \int_0^1 \ell_k(\theta) d\theta \right) f_{n-1-k}$$

であるから、各

$$\alpha_k = \int_0^1 \ell_k(\theta) d\theta$$

を計算すればよい。

### ■ $\ell_0$ の積分

$$\ell_0(\theta) = \frac{(\theta+1)(\theta+2)(\theta+3)}{6} \quad (22)$$

$$= \frac{(\theta^2 + 3\theta + 2)(\theta + 3)}{6} \quad (23)$$

$$= \frac{\theta^3 + 6\theta^2 + 11\theta + 6}{6}. \quad (24)$$

したがって

$$\alpha_0 = \int_0^1 \ell_0(\theta) d\theta \quad (25)$$

$$= \frac{1}{6} \int_0^1 (\theta^3 + 6\theta^2 + 11\theta + 6) d\theta \quad (26)$$

$$= \frac{1}{6} \left[ \frac{\theta^4}{4} + 6 \frac{\theta^3}{3} + 11 \frac{\theta^2}{2} + 6\theta \right]_0^1 \quad (27)$$

$$= \frac{1}{6} \left( \frac{1}{4} + 2 + \frac{11}{2} + 6 \right) = \frac{55}{24}. \quad (28)$$

### ■ $\ell_1$ の積分

$$\ell_1(\theta) = -\frac{\theta(\theta+2)(\theta+3)}{2} \quad (29)$$

$$= -\frac{\theta(\theta^2 + 5\theta + 6)}{2} = -\frac{\theta^3 + 5\theta^2 + 6\theta}{2}. \quad (30)$$

よって

$$\alpha_1 = \int_0^1 \ell_1(\theta) d\theta \quad (31)$$

$$= -\frac{1}{2} \int_0^1 (\theta^3 + 5\theta^2 + 6\theta) d\theta \quad (32)$$

$$= -\frac{1}{2} \left[ \frac{\theta^4}{4} + 5 \frac{\theta^3}{3} + 6 \frac{\theta^2}{2} \right]_0^1 \quad (33)$$

$$= -\frac{1}{2} \left( \frac{1}{4} + \frac{5}{3} + 3 \right) = -\frac{59}{24}. \quad (34)$$

### ■ $\ell_2$ の積分

$$\ell_2(\theta) = \frac{\theta(\theta+1)(\theta+3)}{2} \quad (35)$$

$$= \frac{\theta(\theta^2 + 4\theta + 3)}{2} = \frac{\theta^3 + 4\theta^2 + 3\theta}{2}. \quad (36)$$

したがって

$$\alpha_2 = \int_0^1 \ell_2(\theta) d\theta \quad (37)$$

$$= \frac{1}{2} \int_0^1 (\theta^3 + 4\theta^2 + 3\theta) d\theta \quad (38)$$

$$= \frac{1}{2} \left[ \frac{\theta^4}{4} + 4 \frac{\theta^3}{3} + 3 \frac{\theta^2}{2} \right]_0^1 \quad (39)$$

$$= \frac{1}{2} \left( \frac{1}{4} + \frac{4}{3} + \frac{3}{2} \right) = \frac{37}{24}. \quad (40)$$

### ■ $\ell_3$ の積分

$$\ell_3(\theta) = -\frac{\theta(\theta+1)(\theta+2)}{6} \quad (41)$$

$$= -\frac{\theta(\theta^2 + 3\theta + 2)}{6} = -\frac{\theta^3 + 3\theta^2 + 2\theta}{6}. \quad (42)$$

したがって

$$\alpha_3 = \int_0^1 \ell_3(\theta) d\theta \quad (43)$$

$$= -\frac{1}{6} \int_0^1 (\theta^3 + 3\theta^2 + 2\theta) d\theta \quad (44)$$

$$= -\frac{1}{6} \left[ \frac{\theta^4}{4} + 3 \frac{\theta^3}{3} + 2 \frac{\theta^2}{2} \right]_0^1 \quad (45)$$

$$= -\frac{1}{6} \left( \frac{1}{4} + 1 + 1 \right) = -\frac{3}{8} = -\frac{9}{24}. \quad (46)$$

### ■AB4 の最終形 以上より

$$\int_{t_{n-1}}^{t_n} f(\tau, u(\tau)) d\tau \approx \Delta t \left( \alpha_0 f_{n-1} + \alpha_1 f_{n-2} + \alpha_2 f_{n-3} + \alpha_3 f_{n-4} \right)$$

であり,

$$u_n = u_{n-1} + \Delta t \left( \frac{55}{24} f_{n-1} - \frac{59}{24} f_{n-2} + \frac{37}{24} f_{n-3} - \frac{9}{24} f_{n-4} \right) \quad (47)$$

$$= u_{n-1} + \frac{\Delta t}{24} (55f_{n-1} - 59f_{n-2} + 37f_{n-3} - 9f_{n-4}). \quad (48)$$

これが 4 次アダムス・バッシュフォース法である。

### 3.2 4 次アダムス・ムルトン法の導出

同様にして AM4 を導出する。今度は節点として

$$(t_n, f_n), (t_{n-1}, f_{n-1}), (t_{n-2}, f_{n-2}), (t_{n-3}, f_{n-3})$$

の 4 点を用いる。先ほどと同じ変数変換

$$\tau = t_{n-1} + \theta \Delta t, \quad \theta \in [0, 1]$$

を用いると

$$t_n \leftrightarrow \theta = 1, \quad t_{n-1} \leftrightarrow \theta = 0, \quad t_{n-2} \leftrightarrow \theta = -1, \quad t_{n-3} \leftrightarrow \theta = -2$$

となる。

節点

$$\theta_0 = 1, \quad \theta_1 = 0, \quad \theta_2 = -1, \quad \theta_3 = -2$$

に対し

$$g(\theta_k) = \begin{cases} f_n & (k = 0), \\ f_{n-1} & (k = 1), \\ f_{n-2} & (k = 2), \\ f_{n-3} & (k = 3) \end{cases}$$

である。Lagrange 基底は

$$g(\theta) \approx \tilde{\ell}_0(\theta)f_n + \tilde{\ell}_1(\theta)f_{n-1} + \tilde{\ell}_2(\theta)f_{n-2} + \tilde{\ell}_3(\theta)f_{n-3}$$

と書ける。

同様に

$$\tilde{\ell}_0(\theta) = \frac{(\theta - \theta_1)(\theta - \theta_2)(\theta - \theta_3)}{(\theta_0 - \theta_1)(\theta_0 - \theta_2)(\theta_0 - \theta_3)} \quad (49)$$

$$= \frac{\theta(\theta+1)(\theta+2)}{(1-0)(1+1)(1+2)} = \frac{\theta(\theta+1)(\theta+2)}{6}, \quad (50)$$

$$\tilde{\ell}_1(\theta) = \frac{(\theta - \theta_0)(\theta - \theta_2)(\theta - \theta_3)}{(\theta_1 - \theta_0)(\theta_1 - \theta_2)(\theta_1 - \theta_3)} \quad (51)$$

$$= -\frac{(\theta - 1)(\theta + 1)(\theta + 2)}{2}, \quad (52)$$

$$\tilde{\ell}_2(\theta) = \frac{(\theta - \theta_0)(\theta - \theta_1)(\theta - \theta_3)}{(\theta_2 - \theta_0)(\theta_2 - \theta_1)(\theta_2 - \theta_3)} \quad (53)$$

$$= \frac{\theta(\theta - 1)(\theta + 2)}{2}, \quad (54)$$

$$\tilde{\ell}_3(\theta) = \frac{(\theta - \theta_0)(\theta - \theta_1)(\theta - \theta_2)}{(\theta_3 - \theta_0)(\theta_3 - \theta_1)(\theta_3 - \theta_2)} \quad (55)$$

$$= \frac{\theta(1 - \theta^2)}{6}. \quad (56)$$

同様に

$$\tilde{\alpha}_k = \int_0^1 \tilde{\ell}_k(\theta) d\theta$$

を計算すると

$$\tilde{\alpha}_0 = \frac{3}{8} = \frac{9}{24}, \quad (57)$$

$$\tilde{\alpha}_1 = \frac{19}{24}, \quad (58)$$

$$\tilde{\alpha}_2 = -\frac{5}{24}, \quad (59)$$

$$\tilde{\alpha}_3 = \frac{1}{24} \quad (60)$$

が得られる（展開・積分は AB4 の場合と同様に多項式を展開して項別に行う）。

したがって

$$u_n = u_{n-1} + \Delta t \left( \tilde{\alpha}_0 f_n + \tilde{\alpha}_1 f_{n-1} + \tilde{\alpha}_2 f_{n-2} + \tilde{\alpha}_3 f_{n-3} \right) \quad (61)$$

$$= u_{n-1} + \frac{\Delta t}{24} \left( 9f_n + 19f_{n-1} - 5f_{n-2} + f_{n-3} \right). \quad (62)$$

これが 4 次アダムス・ムルトン法である。

### 3.3 4 次予測子・修正子法

AM4 は  $f_n = f(t_n, u_n)$  を含む陰的法である。そこで、まず AB4 を用いて  $u_n$  の予測値  $\tilde{u}_n$  を計算し、AM4 の中の  $f_n$  を  $f(t_n, \tilde{u}_n)$  に置き換えることで完全に陽的な 4 次予測子・修正子法を得る。

#### ■予測 (Predictor: AB4)

$$\tilde{u}_n = u_{n-1} + \frac{\Delta t}{24} \left( 55f_{n-1} - 59f_{n-2} + 37f_{n-3} - 9f_{n-4} \right). \quad (63)$$

#### ■修正 (Corrector: AM4)

$$u_n = u_{n-1} + \frac{\Delta t}{24} \left( 9f(t_n, \tilde{u}_n) + 19f_{n-1} - 5f_{n-2} + f_{n-3} \right). \quad (64)$$

必要に応じて修正ステップを繰り返せば、AM4 の陰的方程式に対する反復解法として解釈することもできる。

## 4 課題 3 指数関数の微分方程式の数値解法

### 4.1 原理・方法

本課題では、常微分方程式の初期値問題

$$u'(t) = u(t), \quad u(0) = 1 \quad (65)$$

を区間  $t \in [0, 1]$  で数値的に解き、いくつかの代表的な数値解法の収束次数を、誤差の振る舞いから確認する [1]。

式 (65) の解析解は

$$u_{\text{exact}}(t) = \exp(t) \quad (66)$$

であり、特に  $t = 1$  における真値は  $u_{\text{exact}}(1) = e$  である。

### 離散化と誤差の定義

時間刻み幅  $\Delta t$  を

$$\Delta t = \frac{1}{2^i}, \quad i = 1, 2, \dots, i_{\max} \quad (67)$$

とし、ステップ数を  $N = 2^i$ 、離散化した時刻を

$$t_n = n\Delta t, \quad n = 0, 1, \dots, N \quad (68)$$

とおく。数値解を  $u_n \approx u(t_n)$  と表す。

各  $i$  に対し、 $t = 1$  に対応する  $n = N$  での数値解  $u_N^{(i)}$  を求め、絶対誤差

$$E^{(i)} = \left| u_N^{(i)} - u_{\text{exact}}(1) \right| = \left| u_N^{(i)} - e \right| \quad (69)$$

を定義する。

さらに、連続する刻み幅に対する誤差比

$$E_r^{(i)} = \frac{E^{(i)}}{E^{(i-1)}} \quad (i \geq 2) \quad (70)$$

を考える。 $p$  次精度の手法では、 $\Delta t$  を半分にすると誤差が理論的に

$$E^{(i)} \approx C(\Delta t)^p = C2^{-ip} \quad (71)$$

と振る舞うので、

$$E_r^{(i)} \approx \frac{C2^{-ip}}{C2^{-(i-1)p}} = 2^{-p} \quad (72)$$

が期待される。したがって、数値的に  $E_r^{(i)}$  が  $2^{-p}$  に近づくかどうかを調べることで、各手法の収束次数  $p$  を検証できる。

### 対象とする数値解法

本課題では、次の 5 種類の手法を比較する。

#### ■(1) 前進オイラー法 (Forward Euler, FE)

式 (65) に対する前進オイラー法は

$$u_{n+1} = u_n + \Delta t f(t_n, u_n), \quad (73)$$

$$f(t, u) = u \quad (74)$$

で与えられる。本問題では  $f(t_n, u_n) = u_n$  であるから、

$$u_{n+1} = (1 + \Delta t) u_n \quad (75)$$

となる。これは 1 次精度の陽的一段法である。

■(2) 2 次アダムス・バッシュフォース法 (AB2) 2 次の Adams - Bashforth 法は陽的二段法であり,

$$u_{n+1} = u_n + \Delta t \left( \frac{3}{2} f_n - \frac{1}{2} f_{n-1} \right), \quad f_n = f(t_n, u_n) \quad (76)$$

と表される。本問題では  $f_n = u_n$  である。

多段法の性質上,  $u_0, u_1$  の 2 点が初期値として必要になる。本レポートでは,  $u_0$  に加えて  $u_1 = u_{\text{exact}}(t_1)$  を解析解から与えている。

■(3) 3 次アダムス・バッシュフォース法 (AB3) 3 次の Adams - Bashforth 法は陽的三段法であり,

$$u_{n+1} = u_n + \Delta t \left( \frac{23}{12} f_n - \frac{16}{12} f_{n-1} + \frac{5}{12} f_{n-2} \right) \quad (77)$$

で与えられる。本問題では  $f_n = u_n$  である。

三段法のため,  $u_0, u_1, u_2$  の 3 点が初期値として必要になる。本レポートでは,  $u_1, u_2$  を

$$u_1 = u_{\text{exact}}(t_1), \quad u_2 = u_{\text{exact}}(t_2) \quad (78)$$

と解析解から与えている。

■(4) ホイン法 (Heun 法, 2 次ルンゲ・クッタ法) ホイン法は 2 次精度の陽的一段法であり, 予測子・修正子の形で

$$\text{予測: } u_{n+1}^* = u_n + \Delta t f(t_n, u_n), \quad (79)$$

$$\text{修正: } u_{n+1} = u_n + \frac{\Delta t}{2} (f(t_n, u_n) + f(t_{n+1}, u_{n+1}^*)) \quad (80)$$

と書ける。本問題では  $f(t, u) = u$  である。

■(5) 4 次ルンゲ・クッタ法 (RK4) 4 次のルンゲ・クッタ法は

$$k_1 = f(t_n, u_n), \quad (81)$$

$$k_2 = f \left( t_n + \frac{\Delta t}{2}, u_n + \frac{\Delta t}{2} k_1 \right), \quad (82)$$

$$k_3 = f \left( t_n + \frac{\Delta t}{2}, u_n + \frac{\Delta t}{2} k_2 \right), \quad (83)$$

$$k_4 = f(t_n + \Delta t, u_n + \Delta t k_3), \quad (84)$$

$$u_{n+1} = u_n + \frac{\Delta t}{6} (k_1 + 2k_2 + 2k_3 + k_4) \quad (85)$$

と定義される。本問題では  $f(t, u) = u$  であり, 4 次精度の陽的一段法となる。

## 数値実験の手順

式

$$u'(t) = u(t), \quad u(0) = 1$$

を区間  $[0, 1]$  で数値的に解き,  $t = 1$  における解の誤差を比較する。

時間刻みは

$$n = 2^i, \quad i = 2, 3, \dots, 9, \quad \Delta t = \frac{1}{n}$$

とし,  $t_0 = 0, u_0 = 1$  から  $n$  ステップ進めて  $t_n = 1$  の値  $u_n$  を求める。

実装では, 以下の 5 種類の手法を用いた。

- 前進オイラー法 (Forward Euler)

更新式

$$u_{k+1} = u_k + \Delta t f(t_k, u_k), \quad f(t, u) = u$$

を  $k = 0, \dots, n - 1$  について適用する関数 `foward_euler` を用いた。

- 2 次アダムス・バッシュフォース法 (AB2)

更新式

$$u_{k+1} = u_k + \Delta t \left( \frac{3}{2} f_k - \frac{1}{2} f_{k-1} \right), \quad f_k = f(t_k, u_k)$$

を用いる関数 `adam_bashforth2` を実装した。多段法の初期化のため,  $u_0 = 1$  に加えて

$$u_1 = u_{\text{exact}}(t_1) = \exp(\Delta t)$$

を解析解から与え,  $k = 1, \dots, n - 1$  で上式を適用した。

- 3 次アダムス・バッシュフォース法 (AB3)

更新式

$$u_{k+1} = u_k + \Delta t \left( \frac{23}{12} f_k - \frac{16}{12} f_{k-1} + \frac{5}{12} f_{k-2} \right)$$

を用いる関数 `adam_bashforth3` を実装した。初期化のため  $u_0 = 1$  に加え

$$u_1 = u_{\text{exact}}(t_1), \quad u_2 = u_{\text{exact}}(t_2)$$

を解析解から与え,  $k = 2, \dots, n - 1$  で上式を適用した。

- ホイン法 (Heun 法, 2 次ルンゲ・クッタ法)

関数 `heun` で

$$\text{予測: } u_{k+1}^* = u_k + \Delta t f(t_k, u_k), \tag{86}$$

$$\text{修正: } u_{k+1} = u_k + \frac{\Delta t}{2} (f(t_k, u_k) + f(t_{k+1}, u_{k+1}^*)) \tag{87}$$

を  $k = 0, \dots, n - 1$  について適用した。

- 4 次ルンゲ・クッタ法 (RK4)

関数 `runge_kutta4` で

$$k_1 = f(t_k, u_k), \tag{88}$$

$$k_2 = f\left(t_k + \frac{\Delta t}{2}, u_k + \frac{\Delta t}{2} k_1\right), \tag{89}$$

$$k_3 = f\left(t_k + \frac{\Delta t}{2}, u_k + \frac{\Delta t}{2} k_2\right), \tag{90}$$

$$k_4 = f(t_k + \Delta t, u_k + \Delta t k_3), \tag{91}$$

$$u_{k+1} = u_k + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

を  $k = 0, \dots, n - 1$  について適用した。

各  $n$  に対して,  $t = 1$  における真値  $u_{\text{exact}}(1) = e$  との差

$$E(n) = |u_n - e|$$

を計算し, 誤差の大きさと減少の仕方を比較した。さらに, 誤差比

$$\frac{E(n)}{E(n/2)}$$

を用いて, 各手法の収束次数を推定した。

## 4.2 結果

$n = 4, 8, \dots, 512$  に対して得られた  $t = 1$  における誤差  $E(n)$  を表 1 に示す (有効数字 4 術)。

表 1:  $t = 1$  における絶対誤差  $E(n) = |u_n - e|$

$n$	前進オイラー	AB2	AB3	ホイン	RK4
4	0.2769	0.04240	0.005826	0.02343	$7.189 \times 10^{-5}$
8	0.1525	0.01407	0.001300	0.006441	$4.984 \times 10^{-6}$
16	0.08035	0.003973	$2.035 \times 10^{-4}$	0.001688	$3.281 \times 10^{-7}$
32	0.04129	0.001050	$2.820 \times 10^{-5}$	$4.322 \times 10^{-4}$	$2.105 \times 10^{-8}$
64	0.02094	$2.696 \times 10^{-4}$	$3.704 \times 10^{-6}$	$1.093 \times 10^{-4}$	$1.333 \times 10^{-9}$
128	0.01054	$6.826 \times 10^{-5}$	$4.745 \times 10^{-7}$	$2.749 \times 10^{-5}$	$8.384 \times 10^{-11}$
256	0.005290	$1.717 \times 10^{-5}$	$6.003 \times 10^{-8}$	$6.893 \times 10^{-6}$	$5.261 \times 10^{-12}$
512	0.002650	$4.307 \times 10^{-6}$	$7.549 \times 10^{-9}$	$1.726 \times 10^{-6}$	$3.286 \times 10^{-13}$

また, 誤差比  $E(n)/E(n/2)$  の典型的な値として,  $n = 256$  と 512 の組に対しては

前進オイラー	$E(512)/E(256) \approx 0.5009,$
AB2	$E(512)/E(256) \approx 0.2508,$
AB3	$E(512)/E(256) \approx 0.1258,$
ホイン	$E(512)/E(256) \approx 0.2504,$
RK4	$E(512)/E(256) \approx 0.0625$

が得られた。

## 4.3 考察

表 1 から, 刻み数  $n$  を 2 倍にすると各手法の誤差はほぼ一定の比率で減少していることがわかる。誤差比  $E(n)/E(n/2)$  の極限値は

$$2^{-p}$$

となることから,  $p$  次精度の手法では

$$E(n)/E(n/2) \rightarrow 2^{-p}$$

が期待される。本数値結果では、 $n$  が大きい領域で

- 前進オイラー法 :  $E(n)/E(n/2) \rightarrow 0.5 \approx 2^{-1}$ ,
- AB2, ホイン法 :  $E(n)/E(n/2) \rightarrow 0.25 \approx 2^{-2}$ ,
- AB3 :  $E(n)/E(n/2) \rightarrow 0.125 \approx 2^{-3}$ ,
- RK4 :  $E(n)/E(n/2) \rightarrow 0.0625 = 2^{-4}$

となっており、それぞれ 1 次、2 次、3 次、4 次精度であるという理論的な収束次数と整合的な結果になっている。

$n = 512$  での誤差の大きさを比較すると、前進オイラー法は  $E \approx 2.65 \times 10^{-3}$  であるのに対し、AB2, ホイン法はそれぞれ  $10^{-6}$  オーダー、AB3 は  $10^{-9}$  オーダー、RK4 は  $10^{-13}$  オーダーまで誤差が減少している。同じ刻み幅でも、高次の手法ほど誤差が急速に小さくなることが確認できる。

また、多段法である AB2, AB3 は開始ステップで解析解を用いて初期値を与えていたため、この問題設定では一段法よりも有利な初期条件になっている。それにもかかわらず、AB2 とホイン法（ともに 2 次精度）の誤差比はほぼ同じ値に収束しており、「次数が同じであれば刻み幅に対する誤差の減少率も同程度」であることが数値的に確かめられる。RK4 は計算あたりの評価回数が多い一方で、同じ  $n$  に対して最も小さい誤差を与えており、高次の一段法を用いることにより、粗い刻みでも高精度な解が得られることがわかる。

## 5 課題 4 各手法の安定性と安定性な領域の確認

### 5.1 原理・方法

本課題では、線形常微分方程式

$$u'(t) = -\alpha u(t) + \beta, \quad u(0) = u_0, \quad \alpha > 0 \quad (92)$$

に対し、クランク・ニコルソン法、予測子・修正子法、ホイン法を適用したときの**安定性領域**を解析的に求める [1]。

以下では

$$z = \alpha \Delta t$$

とおき、時間刻み幅  $\Delta t$  のみを変化させたときの**增幅因子**の条件から安定性を議論する。

式 (92) の定常解は  $u_* = \beta/\alpha$  であり、 $v(t) = u(t) - u_*$  とおけば

$$v'(t) = -\alpha v(t)$$

となる。したがって、安定性解析では定数項を無視して

$$u'(t) = -\alpha u(t)$$

とした場合の更新式の係数の絶対値を調べれば十分である。

一般に、数値解法により

$$u_{n+1} = R(z) u_n + C(z) \beta$$

と書けるとき、式 (5.1) に対する安定性条件は

$$|R(z)| < 1$$

である。以下、各手法について  $R(z)$  を求め、 $|R(z)| < 1$  から  $z$  の許容範囲を導く。

### クランク・ニコルソン法

クランク・ニコルソン法は

$$u_{n+1} = u_n + \frac{\Delta t}{2}(u'_{n+1} + u'_n) \quad (93)$$

と書ける。式 (92) を代入すると

$$u_{n+1} = u_n + \frac{\Delta t}{2}(-\alpha u_{n+1} + \beta - \alpha u_n + \beta) \quad (94)$$

$$= u_n - \frac{\alpha \Delta t}{2}(u_{n+1} + u_n) + \Delta t \beta \quad (95)$$

$$= u_n - \frac{z}{2}(u_{n+1} + u_n) + \Delta t \beta. \quad (96)$$

これを  $u_{n+1}$  について解くと

$$\left(1 + \frac{z}{2}\right) u_{n+1} = \left(1 - \frac{z}{2}\right) u_n + \Delta t \beta, \quad (97)$$

$$u_{n+1} = \frac{1 - \frac{z}{2}}{1 + \frac{z}{2}} u_n + \frac{\Delta t}{1 + \frac{z}{2}} \beta. \quad (98)$$

したがって增幅因子は

$$R_{CN}(z) = \frac{1 - \frac{z}{2}}{1 + \frac{z}{2}}.$$

安定性条件は

$$\left| \frac{1 - \frac{z}{2}}{1 + \frac{z}{2}} \right| < 1.$$

$z > 0$  のもとでは分母  $1 + z/2 > 0$  なので、これは

$$\left|1 - \frac{z}{2}\right| < 1 + \frac{z}{2}$$

と同値である。両辺を二乗しても不等号の向きは変わらず

$$\left(1 - \frac{z}{2}\right)^2 < \left(1 + \frac{z}{2}\right)^2 \quad (99)$$

$$1 - z + \frac{z^2}{4} < 1 + z + \frac{z^2}{4} \quad (100)$$

$$-z < z \quad (101)$$

となる。 $z > 0$  ならばこれは常に成り立つから、クランク・ニコルソン法は

$$z = \alpha \Delta t > 0$$

に対して常に安定であり、時間刻み幅に制限はない（無条件安定）。

### 予測子・修正子法（前進オイラー+後退オイラー）

ここでは予測子に前進オイラー法、修正子に後退オイラー法を用いた1段の予測子・修正子法を考える。まず前進オイラー法で予測値  $u_n^*$  を求める：

$$u_n^* = u_{n-1} + \Delta t u'_{n-1} \quad (102)$$

$$= u_{n-1} + \Delta t(-\alpha u_{n-1} + \beta) \quad (103)$$

$$= (1 - z)u_{n-1} + \Delta t \beta. \quad (104)$$

次に、この予測値を用いて後退オイラー法を陽的化し、

$$u_n = u_{n-1} + \Delta t (-\alpha u_n^* + \beta) \quad (105)$$

とする ( $f(t_n, u_n)$  を  $f(t_n, u_n^*)$  で近似した形)。

$u_n^*$  を代入して

$$u_n = u_{n-1} + \Delta t \left( -\alpha((1 - z)u_{n-1} + \Delta t \beta) + \beta \right) \quad (106)$$

$$= u_{n-1} - \alpha \Delta t (1 - z)u_{n-1} - \alpha \Delta t^2 \beta + \Delta t \beta \quad (107)$$

$$= (1 - z + z^2)u_{n-1} + (1 - z)\Delta t \beta. \quad (108)$$

したがって增幅因子は

$$R_{\text{PC}}(z) = 1 - z + z^2.$$

安定性条件は

$$|1 - z + z^2| < 1.$$

左辺は

$$1 - z + z^2 = \left( z - \frac{1}{2} \right)^2 + \frac{3}{4} > 0$$

なので、絶対値を外して

$$0 < 1 - z + z^2 < 1$$

に等しい。右の不等式だけを解けばよく、

$$1 - z + z^2 < 1 \quad (109)$$

$$-z + z^2 < 0 \quad (110)$$

$$z(z - 1) < 0 \quad (111)$$

より

$$0 < z < 1$$

となる。したがって予測子・修正子法は

$$0 < \alpha \Delta t < 1 \iff \Delta t < \frac{1}{\alpha}$$

の範囲でのみ安定であり、条件付き安定である。

## ホイン法 (Heun 法)

ホイン法は 2 次のルンゲ・クッタ法であり、予測値  $u_n^*$  を

$$u_n^* = u_{n-1} + \Delta t u'_{n-1} = (1 - z)u_{n-1} + \Delta t \beta \quad (112)$$

とおいたあと、

$$u_n = u_{n-1} + \frac{\Delta t}{2} \left( u'_{n-1} + u_n^{* *} \right) \quad (113)$$

で修正を行う。ここで

$$u'_{n-1} = -\alpha u_{n-1} + \beta, \quad (114)$$

$$u_n^{* *} = -\alpha u_n^* + \beta = -\alpha((1 - z)u_{n-1} + \Delta t \beta) + \beta \quad (115)$$

であるから

$$u_n = u_{n-1} + \frac{\Delta t}{2} \left[ -\alpha u_{n-1} + \beta - \alpha(1 - z)u_{n-1} - \alpha \Delta t \beta + \beta \right] \quad (116)$$

$$= u_{n-1} - \frac{\alpha \Delta t}{2} (2 - z)u_{n-1} + \frac{\Delta t}{2} (2 - z)\beta \quad (117)$$

$$= (1 - z + \frac{z^2}{2})u_{n-1} + \left(1 - \frac{z}{2}\right) \Delta t \beta. \quad (118)$$

したがって增幅因子は

$$R_{\text{Heun}}(z) = 1 - z + \frac{z^2}{2}$$

である。

安定性条件は

$$\left|1 - z + \frac{z^2}{2}\right| < 1.$$

同様に

$$1 - z + \frac{z^2}{2} = \frac{1}{2}(z - 1)^2 + \frac{1}{2} > 0$$

なので、これは

$$0 < 1 - z + \frac{z^2}{2} < 1$$

と同値である。右の不等式を解くと

$$1 - z + \frac{z^2}{2} < 1 \quad (119)$$

$$-z + \frac{z^2}{2} < 0 \quad (120)$$

$$z \left( \frac{z}{2} - 1 \right) < 0 \quad (121)$$

より

$$0 < z < 2$$

を得る。したがってホイン法は

$$0 < \alpha \Delta t < 2 \iff \Delta t < \frac{2}{\alpha}$$

の範囲で安定である。

## 5.2 解析的結果のまとめ

以上より、式 (92) に対して得られた各手法の安定性領域 ( $z = \alpha\Delta t > 0$ ) は次のようにまとめられる：

- クランク・ニコルソン法： 全ての  $z > 0$  で安定（無条件安定）。
- 予測子・修正子法（前進オイラー+後退オイラー）：  $0 < z < 1$  で安定。
- ホイン法：  $0 < z < 2$  で安定。

これらの理論的安定性領域と、実際に数値計算で観測される解の振る舞いを比較することで、次節で数値的な確認を行う。

## 5.3 結果

本課題では、線形常微分方程式

$$u'(t) = -\alpha u(t) + \beta, \quad u(0) = 1, \quad (122)$$

に対して  $\alpha = 10$ ,  $\beta = 0$  を固定し、時間区間  $[0, 10]$  を各種の刻み幅  $\Delta t$  で分割して数値解を求めた。真値は

$$u_{\text{exact}}(t) = e^{-10t} \quad (123)$$

である。

クランク・ニコルソン法については

$$\Delta t = 0.01, 0.19, 0.205$$

の場合を計算し、図 1 に数値解と真値の比較を示した。縦軸は  $u \in [-1, 1]$  の範囲に固定している。 $\Delta t = 0.01$  の場合、真値とほぼ重なる減衰曲線が得られた。 $\Delta t = 0.19, 0.205$  では時間刻みが粗くなるため初期付近で真値との差が大きくなるが、いずれの場合も数値解は急速に原点へ収束し、発散や振動は観測されなかった。

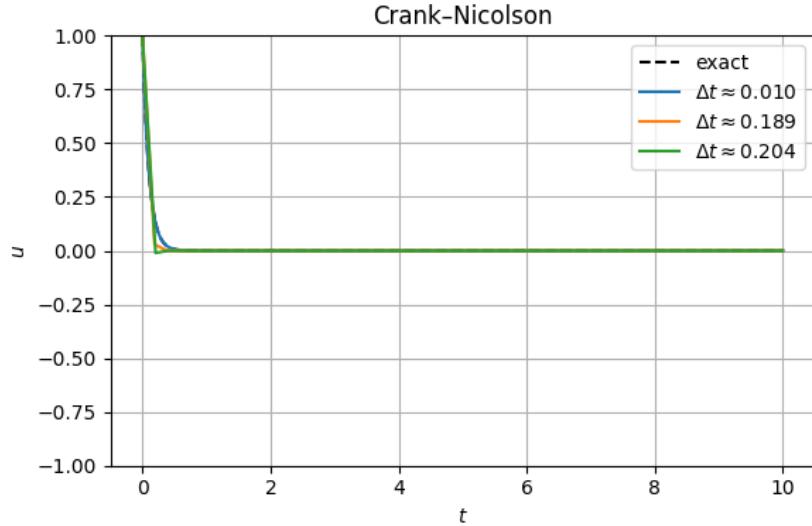


図 1: クランク・ニコルソン法による式 (4.28) の数値解 ( $\Delta t = 0.01, 0.19, 0.205$ )。

予測子・修正子法（前進オイラー+後退オイラー）については、安定性条件  $0 < z = \alpha\Delta t < 1$  の境界付近の挙動を観察するため、

$$\Delta t = 0.01, 0.095, 0.105$$

とした。図 2 に、縦軸  $u \in [-1, 10]$  の範囲で描いた結果を示す。 $\Delta t = 0.01$  では真値と同様に滑らかにゼロへ減衰する。 $\Delta t = 0.095$  ( $z = 0.95$ ) では、減衰は続くものの、 $\Delta t = 0.01$  に比べてゼロへの収束が非常に緩やかになっている。一方、 $\Delta t = 0.105$  ( $z = 1.05$ ) では、時間の経過とともに振幅が増大し、 $t = 10$  の時点では  $u$  が  $10^2$  オーダーまで成長する発散解が得られた。

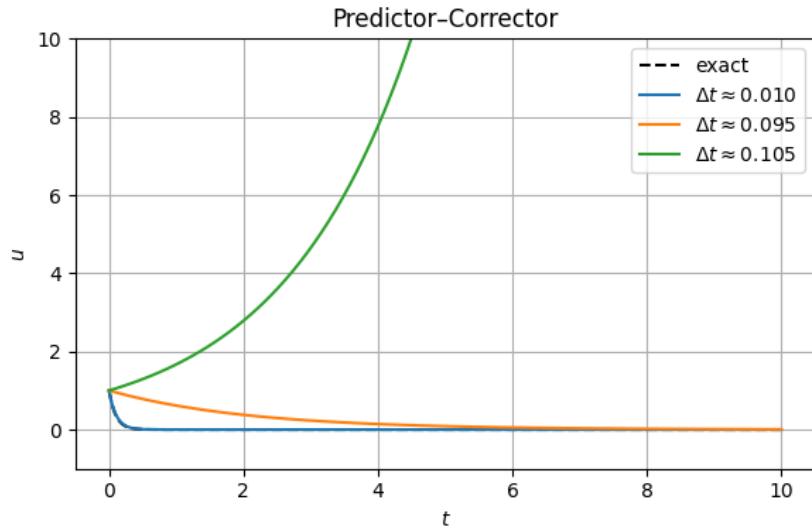


図 2: 予測子・修正子法（前進オイラー+後退オイラー）による式 (4.28) の数値解 ( $\Delta t = 0.01, 0.095, 0.105$ )。

ホイン法については、クランク・ニコルソン法と同じ

$$\Delta t = 0.01, 0.19, 0.205$$

を用いて計算し、結果を図 3 に示した。縦軸は  $u \in [-1, 10]$  に固定している。 $\Delta t = 0.01$  では真値とほとんど一致する減衰が得られた。 $\Delta t = 0.19$  ( $z = 1.9$ ) では理論上はまだ安定領域内であり、数値解は緩やかにゼロに近づく様子が見られた。これに対し、 $\Delta t = 0.205$  ( $z \approx 2.05$ ) では安定限界  $z = 2$  をわずかに超えているため、数値解は  $t = 10$  の時点で  $u$  が  $10^1$  オーダーに達する発散挙動を示した。

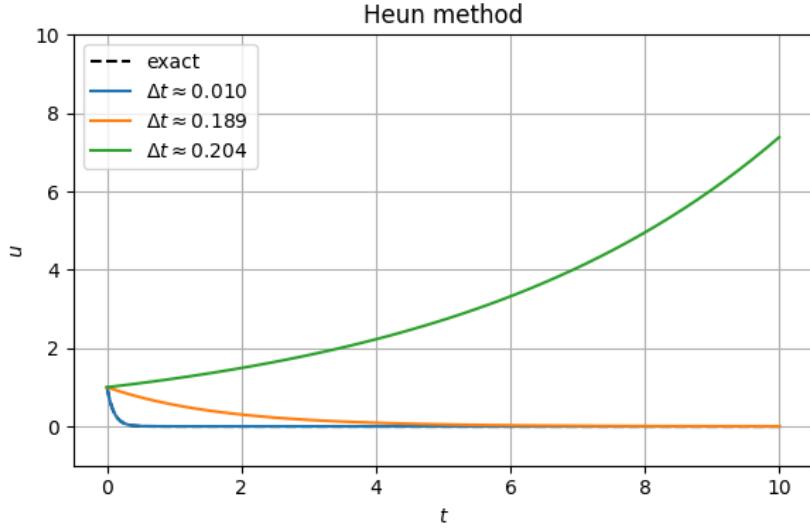


図 3: ホイン法による式 (4.28) の数値解 ( $\Delta t = 0.01, 0.19, 0.205$ )。

## 5.4 考察

まず、クランク・ニコルソン法について解析的に得られた增幅因子

$$R_{\text{CN}}(z) = \frac{1 - \frac{z}{2}}{1 + \frac{z}{2}}, \quad z = \alpha \Delta t$$

は、 $z > 0$  に対して常に  $|R_{\text{CN}}(z)| < 1$  を満たすため無条件安定である。数値結果でも、 $\Delta t = 0.19, 0.205$  のように  $\alpha \Delta t$  が 1 を大きく超える場合であっても、数値解はすべての時刻で有界に保たれ、時間とともにゼロへ収束している。ただし、 $z$  が大きくなると  $R_{\text{CN}}(z)$  は真の減衰係数  $e^{-z}$  から大きく外れ、過度に強い減衰や ( $z > 2$  の領域では) 符号反転を伴う振動が生じる。したがって、クランク・ニコルソン法は安定性の観点からは刻み幅に制限を受けない一方で、精度の観点からは  $\alpha \Delta t$  を適度に小さく取る必要があることが確認できる。

予測子・修正子法（前進オイラー+後退オイラー）の增幅因子は

$$R_{\text{PC}}(z) = 1 - z + z^2$$

であり、 $|R_{\text{PC}}(z)| < 1$  より  $0 < z < 1$  の範囲のみ安定である。数値結果では、 $z = 0.1$  に対応する  $\Delta t = 0.01$  では真値と同様に指数減衰が再現され、 $z = 0.95$  に対応する  $\Delta t = 0.095$  では減衰はするものの、 $R_{\text{PC}}(0.95) \approx 0.95$  と 1 に近いため、ゼロへの収束が非常に遅いことが観察された。これに対し、 $z = 1.05$  の  $\Delta t = 0.105$  では  $R_{\text{PC}}(1.05) \approx 1.05 > 1$  となり、理論通り発散解が得られた。安定限界をわずかに越えただけで振幅が指数的に増大し、 $t = 10$  の段階で  $10^2$  オーダーにまで達し

ていることから、この手法はクランク・ニコルソン法に比べてかなり厳しい刻み幅制限を受けることが分かる。

ホイン法の増幅因子は

$$R_{\text{Heun}}(z) = 1 - z + \frac{z^2}{2}$$

であり、安定範囲は  $0 < z < 2$  となる。数値結果でも、 $z = 1$  に相当するような  $\Delta t$  では安定で、 $\Delta t = 0.19$  ( $z \approx 1.9$ ) では緩やかな減衰が観測された。一方、 $\Delta t = 0.205$  ( $z \approx 2.05$ ) では  $R_{\text{Heun}}(z) > 1$  となり、数値解がゼロから離れて増大していった。すなわち、ホイン法も予測子・修正子法と同様に条件付き安定であり、明示的手法として典型的な「有限の安定領域」を持つことが数値的にも確認された。

以上をまとめると、今回の一次元線形問題に関して

- クランク・ニコルソン法は無条件安定であり、大きな  $\Delta t$  に対しても解は発散しないが、減衰の速さや位相の再現性は  $\alpha\Delta t$  に強く依存する。
- 予測子・修正子法は高次化された陽的手法であるにもかかわらず、安定領域は  $0 < \alpha\Delta t < 1$  と比較的狭く、安定限界をわずかに上回るだけで解が急激に発散する。
- ホイン法は安定領域  $0 < \alpha\Delta t < 2$  を持ち、予測子・修正子法よりは広いものの、やはり刻み幅を安定限界近くまで大きくすると解が発散する。

このように、安定性解析で導かれた増幅因子  $R(z)$  の条件と、実際の数値解の振る舞いが定性的にも定量的にも一致していることが確認できた。より高次・高精度な手法であっても、陽的である限りは安定領域の制約を回避できないため、剛性の強い問題や固有値の実部が大きく負の問題では、クランク・ニコルソン法のような陰的手法を用いる方が実用上有利であるといえる。

## 6 課題 5 不安定性の確認

### 6.1 問題設定

常微分方程式

$$u'(t) = -2u(t) + 1, \quad u(0) = 1 \quad (124)$$

を考える。本課題では、ステップ幅  $\Delta t > 0$  を用いて

$$t_n = n\Delta t, \quad u_n \approx u(t_n)$$

とし、2段法

$$u_n = u_{n-2} + 2\Delta t f(t_{n-1}, u_{n-1}), \quad f(t, u) = -2u + 1 \quad (125)$$

による数値解の安定性を解析する。

式 (125) は、区間  $[t_{n-2}, t_n]$  で中点  $t_{n-1}$  における微分係数  $u'(t_{n-1})$  を  $2\Delta t$  だけ積分した形になつておる、いわゆる中点公式（明示 2 段法）の一種である [1]。

## 6.2 差分スキームの形

(124) を (125) に代入すると

$$\begin{aligned} u_n &= u_{n-2} + 2\Delta t (-2u_{n-1} + 1) \\ &= u_{n-2} - 4\Delta t u_{n-1} + 2\Delta t \end{aligned} \quad (126)$$

を得る。

この差分方程式の定常解を考えると、連続系の平衡解  $u_*$  は

$$-2u_* + 1 = 0 \Rightarrow u_* = \frac{1}{2}$$

であるから、数値解についても

$$u_n \equiv \frac{1}{2}$$

が定常解になっていることが分かる。

数値解の安定性を見るため、平衡解からの偏差

$$v_n = u_n - u_* = u_n - \frac{1}{2}$$

を導入する。(126) に  $u_n = v_n + \frac{1}{2}$  を代入すると

$$v_n + \frac{1}{2} = v_{n-2} + \frac{1}{2} - 4\Delta t \left( v_{n-1} + \frac{1}{2} \right) + 2\Delta t. \quad (127)$$

両辺から  $\frac{1}{2}$  を消去すると

$$v_n = v_{n-2} - 4\Delta t v_{n-1} \quad (128)$$

を得る。したがって誤差  $v_n$  は線形 2 階の齊次漸化式

$$v_n + 4\Delta t v_{n-1} - v_{n-2} = 0 \quad (129)$$

に従う。

## 6.3 特性方程式による安定性解析

漸化式 (129) の特性方程式は

$$D(\lambda) = \lambda^2 + 4\Delta t \lambda - 1 = 0 \quad (130)$$

である。(130) の 2 つの根を  $\lambda_1, \lambda_2$  とすると、係数比較から

$$\lambda_1 + \lambda_2 = -4\Delta t, \quad (131)$$

$$\lambda_1 \lambda_2 = -1 \quad (132)$$

が成り立つ。

特に、(132) より

$$|\lambda_1 \lambda_2| = 1$$

であるから、もし一方の根が  $|\lambda_1| < 1$  を満たすならば、もう一方は

$$|\lambda_2| = \frac{1}{|\lambda_1|} > 1$$

を満たさざるを得ない。すなわち、2つの根の少なくとも一方は必ず  $|\lambda| > 1$  となる。

根の位置をもう少し具体的に確認するため、 $D(\lambda)$  の符号を調べる。 $\Delta t > 0$  とすると

$$D(-1) = (-1)^2 + 4\Delta t(-1) - 1 = -4\Delta t < 0, \quad (133)$$

$$D(0) = -1 < 0, \quad (134)$$

$$D(1) = 1 + 4\Delta t - 1 = 4\Delta t > 0. \quad (135)$$

したがって、連続性より

$$0 < \lambda_1 < 1, \quad \lambda_2 < -1$$

となる2つの実根が存在する。このうち  $\lambda_2$  は必ず  $|\lambda_2| > 1$  を満たす。

漸化式 (129) の一般解は

$$v_n = C_1 \lambda_1^n + C_2 \lambda_2^n \quad (136)$$

と書けるので、初期値  $(v_0, v_1)$  が特別に選ばれて  $C_2 = 0$  となる場合を除き、 $\lambda_2^n$  の項が支配的になり

$$|v_n| \sim |C_2| |\lambda_2|^n \rightarrow \infty \quad (n \rightarrow \infty)$$

となる。

以上から、差分スキーム (125) を (124) に適用すると、平衡解からの偏差  $v_n$  は必ず

$$|\lambda_2| > 1$$

を持つ固有値成分を含むため、一般的な初期値に対して  $n \rightarrow \infty$  で発散する。

すなわち、任意の  $\Delta t > 0$  に対して、この2段法は式 (124) に対して安定ではなく、 $u_n$  は必ず発散する。

## 6.4 数値解析による結果

常微分方程式

$$u'(t) = -2u(t) + 1, \quad u(0) = 1 \quad (137)$$

に対して、2段法

$$u_n = u_{n-2} + 2\Delta t (-2u_{n-1} + 1) \quad (138)$$

を用いて数値解の時間発展を調べた。

時間区間は  $[0, 10]$  とし、刻み幅

$$\Delta t = 0.01, 0.05, 0.10$$

の3通りについて計算を行った。2段法であるため、 $u_0 = 1$  に加え、 $u_1$  は解析解

$$u_{\text{exact}}(t) = \frac{1}{2} + \frac{1}{2}e^{-2t}$$

を用いて

$$u_1 = u_{\text{exact}}(\Delta t)$$

と与えた。その後は式 (138) に従って  $n = 2, 3, \dots, N$  まで時間発展させた ( $N\Delta t = 10$ )。

図 4 に、真値と 3 種類の刻み幅に対する数値解を示す。縦軸は  $u \in [-200, 200]$  に固定して描画した。いずれの刻み幅でも、初期のうちは真値と同様に減衰しているように見えるが、時間が進むにつれて振幅が増大し、 $t \approx 10$  では  $u$  が  $\pm 10^2 \sim 10^5$  のオーダで発散していることが分かる。特に  $\Delta t$  が大きいほど発散が急激であり、 $\Delta t = 0.10$  の場合には数値解が短時間で縦軸の表示範囲を大きく超えている。

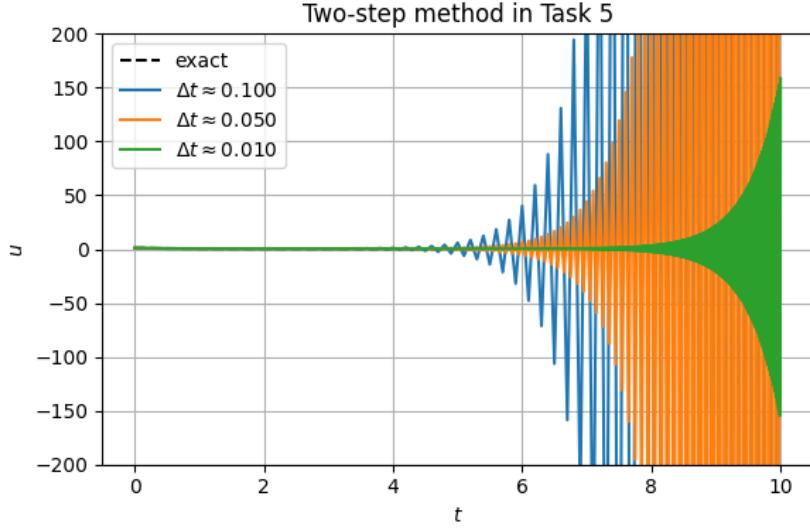


図 4: 課題 5 の 2 段法  $u_n = u_{n-2} + 2\Delta t(-2u_{n-1} + 1)$  による式 (4.29) の数値解 ( $\Delta t = 0.01, 0.05, 0.10$ )。

$t = 10$  における数値解  $u_N$  と真値  $u_{\text{exact}}(10)$  の絶対誤差

$$E = |u_N - u_{\text{exact}}(10)|$$

を表 2 に示す (有効数字 4 衔)。

表 2: 課題 5 の 2 段法における  $t = 10$  での数値解と誤差

$\Delta t$	$u_N$	$E =  u_N - u_{\text{exact}}(10) $
0.01	$1.588 \times 10^2$	$1.583 \times 10^2$
0.05	$1.753 \times 10^4$	$1.753 \times 10^4$
0.10	$1.120 \times 10^5$	$1.120 \times 10^5$

真値  $u_{\text{exact}}(10) \approx 0.5$  が有界であるのに対し、どの刻み幅でも  $u_N$  は時間とともに急速に増大しており、解析で得られた「本 2 段法の特性根の一つは常に  $|\lambda| > 1$  となるため、一般の初期値に対して数値解は必ず発散する」という結果と定性的にも定量的にも一致している。

## 7 課題6 爆発解の数値計算と弧長パラメータによる変数変換

### 7.1 原理と方法

生物個体群の増減を表す簡単なアリー効果モデルとして、次の常微分方程式を考える：

$$u'(t) = (u(t) - 1) u(t), \quad u(0) = u_0 > 0. \quad (139)$$

右辺は個体数  $u$  が多いほど増加率が大きくなること、さらに  $u < 1$  のときは右辺が負となり、個体群が減少に向かうことを表している。

式 (139) の解は変数分離により解析的に求めることができる。 $u'(t) = u(u - 1)$  より

$$\frac{1}{u(u - 1)} du = dt$$

を積分すると

$$\ln \left| \frac{u - 1}{u} \right| = t + C$$

を得る。初期条件  $u(0) = u_0$  を代入して  $C$  を消去すると、解析解は

$$u(t) = \begin{cases} 0, & u_0 = 0, \\ 1, & u_0 = 1, \\ \frac{u_0}{u_0 + (1 - u_0)e^t}, & \text{それ以外} \end{cases} \quad (140)$$

と書ける。 $0 < u_0 < 1$  では  $t \rightarrow \infty$  で  $u(t) \rightarrow 0$ 、 $u_0 = 1$  では  $u(t) \equiv 1$ 、 $u_0 > 1$  では有限時刻で分母が 0 となり、解は有限時間で爆発する。特に  $u_0 > 1$  の場合の爆発時刻  $T_b$  は

$$u_0 + (1 - u_0)e^{T_b} = 0 \implies T_b = \ln \frac{u_0}{u_0 - 1} \quad (141)$$

と求まる。

爆発解の数値計算では、 $t$  のごく短い変化に対して  $u$  が急激に増加するため、固定ステップ幅の手法では安定に追跡することが困難になることが多い。本課題では、解軌道  $(t, u(t))$  に沿った弧長

$$s(t) = \int_0^t \sqrt{1 + (u'(\tau))^2} d\tau$$

を新たな独立変数として用い、弧長方向に一様ステップで積分することで爆発近傍の解を追跡する [1]。

### 7.2 数値解法の実装

教科書の弧長パラメータへの変数変換に従えば、 $f(t, u) = (u - 1)u$  に対して

$$\frac{du}{ds} = \frac{f(t, u)}{\sqrt{1 + f(t, u)^2}} = \frac{(u - 1)u}{\sqrt{1 + (u - 1)^2 u^2}}, \quad (142)$$

$$\frac{dt}{ds} = \frac{1}{\sqrt{1 + f(t, u)^2}} = \frac{1}{\sqrt{1 + (u - 1)^2 u^2}}, \quad (143)$$

となり、 $s$  を独立変数とする 2 次元の常微分方程式系が得られる。

実装では、この系をそのまま解く代わりに、ステップ幅  $\Delta t$  を

$$\Delta s \approx \sqrt{1 + f(t, u)^2} \Delta t = \text{const.} \quad (144)$$

とみなして更新する。具体的には、まず等間隔の基準幅

$$\Delta t_{\text{default}} = \frac{t_{\text{end}} - t_{\text{start}}}{n}$$

を定め、各ステップで

$$\Delta t = \frac{\Delta t_{\text{default}}}{\sqrt{1 + f(t, u)^2}} \quad (145)$$

と更新することで、解軌道に沿った弧長ステップ  $\Delta s$  が概ね一定になるように制御した。

従属変数の更新には 4 次のルンゲ・クッタ法を用いた。数値解が発散したかの判定には  $|u| > 10^5$  を閾値とし、この閾値を超えた時点で  $u = \infty$  とみなして計算を打ち切った。計算は  $t \in [0, 5]$  とし、初期値  $u_0$  を

$$u_0 = 0.0, 0.2, 0.4, \dots, 2.0$$

の 11 通りについて実行した。

爆発解の評価では、 $u$  自体の誤差は爆発近傍で非常に大きくなり比較が困難であるため、値を圧縮するための変換として

$$v(t) = \arctan(u(t))$$

を導入した。これにより  $u \rightarrow \infty$  のときでも  $v \rightarrow \pi/2$  で抑えられ、数値解と解析解の差

$$e_{\text{atan}}(t) = |\arctan u_{\text{num}}(t) - \arctan u_{\text{exact}}(t)|$$

を全時間区間で評価できる。

さらに  $u_0 > 1$  の場合については、最後の有限な 2 点

$$(t_{n-1}, \arctan u_{n-1}), \quad (t_n, \arctan u_n)$$

を通る直線で  $\arctan u$  を近似し、この直線が  $y = \pi/2$  と交わる  $t$  を爆発時刻の数値的推定値  $\hat{T}_b$  として求め、解析的な  $T_b$  (式 (141)) との比較を行った。

### 7.3 結果

図 5 に  $u(t)$  と  $\arctan(u(t))$  の軌道を示す。上段は  $u(t)$  を、下段は  $\arctan(u(t))$  を時間  $t$  の関数として描いたものである。

図より、 $0 < u_0 < 1$  のとき解は単調減少し  $t \rightarrow \infty$  で  $u(t) \rightarrow 0$  となること、 $u_0 = 1$  では  $u(t) \equiv 1$  の平衡解であること、 $u_0 > 1$  では有限時間で爆発することが確認できる。 $\arctan(u(t))$  をプロットした下段の図では、 $u_0 > 1$  の場合に  $\arctan(u(t))$  が  $\pi/2$  に向かって滑らかに増加し、爆発時刻近傍の挙動を有限スケールで観察できている。

数値解の精度評価として、 $\arctan(u)$  空間での平均絶対誤差  $\text{mean}|e_{\text{atan}}|$  と最大絶対誤差  $\max|e_{\text{atan}}|$  を求めた結果の一部を表 3 に示す。 $u_0 \leq 1$  の範囲では

$$\text{mean}|e_{\text{atan}}| \lesssim 1.3 \times 10^{-6}, \quad \max|e_{\text{atan}}| \lesssim 3.1 \times 10^{-6}$$

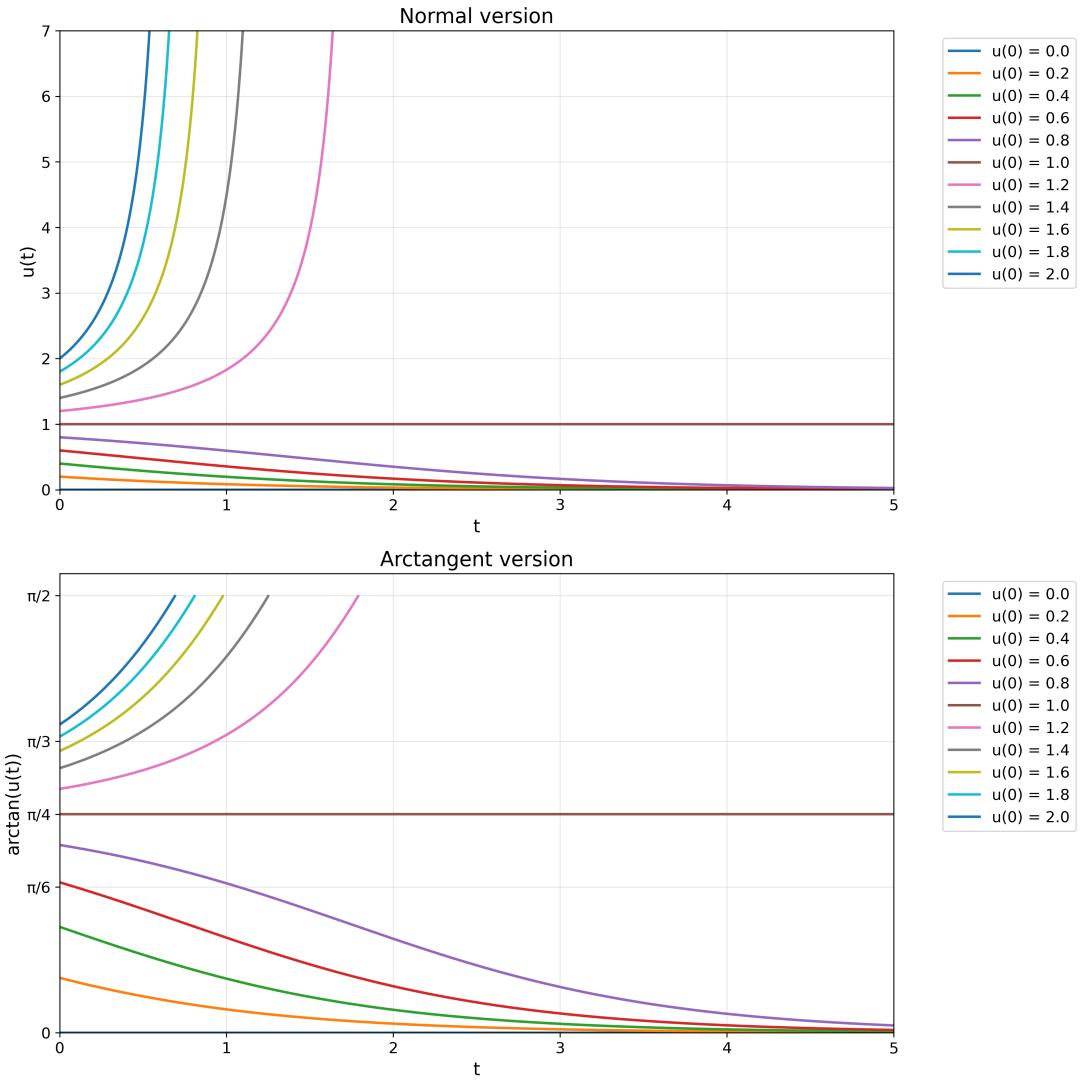


図 5: 式 (139) の数値解 (上:  $u(t)$ , 下:  $\arctan u(t)$ )。初期値  $u_0 = 0.0, 0.2, \dots, 2.0$  の結果を重ねて表示した。

表 3:  $\arctan(u)$  空間での絶対誤差と爆発時刻推定値 (抜粋)。

$u_0$	mean $ e_{\text{atan}} $	max $ e_{\text{atan}} $	$T_b$	$\hat{T}_b$	$\hat{T}_b - T_b$
0.2	$9.38 \times 10^{-8}$	$9.65 \times 10^{-7}$	—	—	—
0.4	$4.80 \times 10^{-7}$	$2.86 \times 10^{-6}$	—	—	—
0.6	$9.52 \times 10^{-7}$	$3.03 \times 10^{-6}$	—	—	—
0.8	$1.23 \times 10^{-6}$	$3.03 \times 10^{-6}$	—	—	—
1.0	0	0	—	—	—
1.2	$4.92 \times 10^{-4}$	$4.99 \times 10^{-4}$	1.791759	1.791261	$-4.98 \times 10^{-4}$
1.4	$4.93 \times 10^{-4}$	$4.99 \times 10^{-4}$	1.252763	1.252265	$-4.98 \times 10^{-4}$
1.6	$4.93 \times 10^{-4}$	$4.99 \times 10^{-4}$	0.980829	0.980331	$-4.98 \times 10^{-4}$
1.8	$4.93 \times 10^{-4}$	$4.99 \times 10^{-4}$	0.810930	0.810432	$-4.98 \times 10^{-4}$
2.0	$4.94 \times 10^{-4}$	$4.99 \times 10^{-4}$	0.693147	0.692649	$-4.98 \times 10^{-4}$

となっており、弧長パラメータを用いたルンゲ・クッタ法が高い精度で解析解を再現している。

$u_0 > 1$  の場合、 $\arctan(u)$  空間での平均・最大誤差はいずれも約  $5 \times 10^{-4}$  程度であり、角度に換算するとおよそ  $0.03^\circ$  に相当する。また、全ての  $u_0 > 1$  について、爆発時刻の数値的推定値  $\hat{T}_b$  は解析的な  $T_b$  より約  $4.98 \times 10^{-4}$  だけ早い（負の差）というほぼ一定のバイアスを示し、相対誤差は最大でも  $|\hat{T}_b - T_b|/T_b \lesssim 7.2 \times 10^{-4}$  に抑えられている。

## 考察

式 (139) の右辺  $(u-1)u$  は  $u=0$  および  $u=1$  を平衡点として持ち、 $0 < u < 1$  のとき負、 $u > 1$  のとき正である。このため、 $0 < u_0 < 1$  の解は単調減少して  $u=0$  に収束し、 $u_0 = 1$  は不動点として保たれる。一方  $u_0 > 1$  では  $u(t)$  が単調増加し、有限時間で爆発することが解析的にも数値的にも確認された。

固定ステップ幅の方法で爆発解を扱うと、 $u'(t)$  が非常に大きくなる領域で解の変化を捉えるために極端に小さな  $\Delta t$  が必要になる。本課題で用いた弧長に基づくステップ幅制御では、 $|u'|$  が大きくなるにつれて  $\Delta t$  が自動的に小さくなり、弧長方向にはほぼ一定の刻み幅で積分が進む。その結果、 $u_0 > 1$  の場合にも爆発直前まで解の軌道を比較的少ないステップ数で追跡することができた。

また、今回の変換の仕様上、値が無限に発散する爆発解であっても、1ステップで縦幅にて高々  $\Delta t$  の増分で更新されるため、数値解が十分大きくなるのに途方もない時間がかかるので、ループの回数の上限を設定し、閾値を超えた時点で計算を打ち切るようにした。このため、爆発直前の非常に大きな  $u$  の値は数値的に得られなかったが、 $\arctan(u)$  空間に圧縮することにより、爆発近傍の挙動が有限スケールで観察でき、これによって関数  $u$  が無限大に相当する  $v = \pi/2$  に到達する時刻を推定できた。

## 8 課題 7 ローレンツ方程式の数値解法とカオス現象の観察

### 8.1 原理, 方法

ローレンツ方程式は、レイリー・ベナール対流の単純化モデルとして提案された 3 変数の非線形常微分方程式であり、パラメータの取り方によってカオス的挙動を示すことが知られている。本課題では

$$\frac{dx}{dt} = \sigma(y - x), \quad (146)$$

$$\frac{dy}{dt} = rx - y - xz, \quad (147)$$

$$\frac{dz}{dt} = xy - bz \quad (148)$$

で与えられるローレンツ方程式を対象とし、パラメータ

$$\sigma = 10, \quad b = \frac{8}{3}, \quad r = 28$$

に固定した。初期条件は

$$x(0) = 1 + \varepsilon, \quad y(0) = 0, \quad z(0) = 0$$

とし、 $\varepsilon$  およびステップ幅  $\Delta t$  を変化させて数値解の挙動を比較することで、(1) 非周期的な長時間挙動（ローレンツ・アトラクタ）、(2) 数値誤差の增幅、(3) 初期値に対する鋭敏な依存性を確認する

ことを目的とする。

時間区間は  $t \in [0, 100]$  とした。数値積分には前進オイラー法と 4 次のルンゲ・クッタ法 (RK4) を用いた。各手法の実装は以下の通りである。課題 (1) では  $\varepsilon = 0$ , ステップ幅  $\Delta t = 1.000 \times 10^{-2}$  (ステップ数  $n = 1.000 \times 10^4$ ) として  $t \in [0, 100]$  を積分し,  $x(t)$  の時間波形と  $(x(t), y(t), z(t))$  の軌道を 3 次元空間で描画した。前進オイラー法と RK4 法それぞれについて軌道を可視化し, 長時間挙動を比較した。

課題 (2) では,  $\varepsilon = 0$  のままステップ幅を

$$\Delta t_i = 1.000 \times 10^{-2} \times 2^{-i}, \quad i = 0, 1, \dots, 13 \quad (149)$$

と変化させた (最小ステップ幅は  $\Delta t_{13} = 1.221 \times 10^{-6}$ )。各  $\Delta t_i$  について  $t \in [0, 100]$  を積分し,  $t = 15, 30, 60$  における  $x(t)$  の値を前進オイラー法と RK4 法それぞれで取得した。 $t^* \in \{15, 30, 60\}$  ごとに, 横軸を  $\Delta t$ , 縦軸を  $x(t^*)$  としてプロットすることで, ステップ幅に対する数値解の依存性を調べた。

課題 (3) では, 初期値を

$$\varepsilon \in \{0, 0.1, 0.01, 0.001\}$$

の 4 通りに変化させ, いずれも RK4 法を用いて

$$\Delta t = 1.000 \times 10^{-3}, \quad n = 1.000 \times 10^5$$

で  $t \in [0, 100]$  を積分した。まず  $t \in [0, 50]$  および  $t \in [50, 100]$  について  $x(t)$  を重ね書きし, 初期値の微小な違いが時間とともにどのように增幅されるかを視覚的に確認した。さらに基準として  $\varepsilon = 0$  の軌道  $u^{(0)}(t)$  を用意し, 他の  $\varepsilon$  に対して

$$e_\varepsilon(t_k) = \|u^{(\varepsilon)}(t_k) - u^{(0)}(t_k)\|_2$$

を全時刻ステップで計算した。各  $\varepsilon$  について, 時刻  $t$  までの最大誤差

$$E_\varepsilon(t_k) = \max_{j \leq k} e_\varepsilon(t_j)$$

をプロットし, 誤差の時間発展を評価した。

## 8.2 結果

**■(1) ローレンツ・アトラクタの観測** RK4 法による  $(x(t), y(t), z(t))$  の軌道を 3 次元空間にプロットした結果を図 6 に示す。軌道は有限の領域内にとどまり, 左右 2 つの「羽根」が重なったような特徴的なローレンツ・アトラクタが得られた。 $x(t)$  の時間波形は, ほぼ有界な振動を繰り返すが, 周期性は見られないことを確認した。

一方, 前進オイラー法による軌道を図 7 に示す。短時間では RK4 法と類似した形状をとるが,  $t$  が大きくなるにつれて軌道が外側にずれていき, アトラクタの形が RK4 法と比べて徐々に歪む様子が見られた。

Lorenz Attractor - Runge-Kutta Method

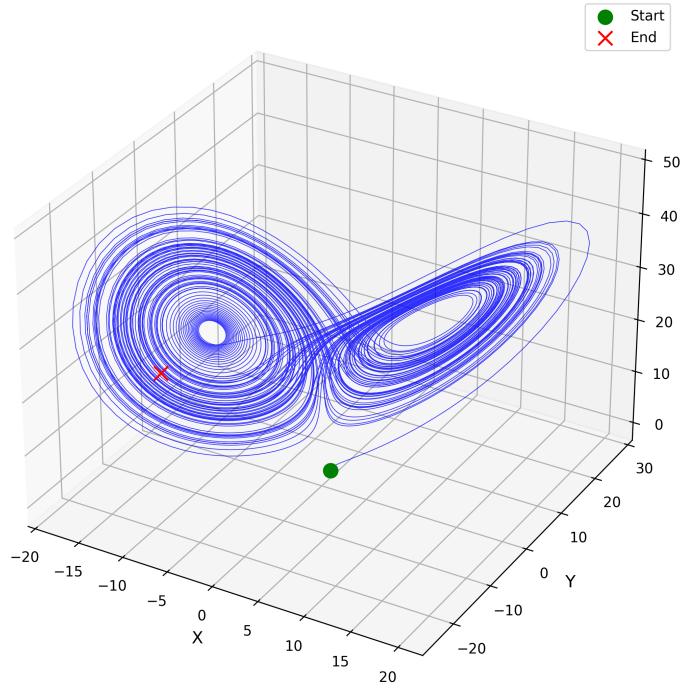


図 6: RK4 法によるローレンツ方程式の軌道  $(x(t), y(t), z(t))$ . パラメータは  $\sigma = 10$ ,  $b = 8/3$ ,  $r = 28$ , 初期値は  $(1, 0, 0)$ ,  $\Delta t = 1.000 \times 10^{-2}$ ,  $t \in [0, 100]$ .

Lorenz Attractor - Forward Euler Method

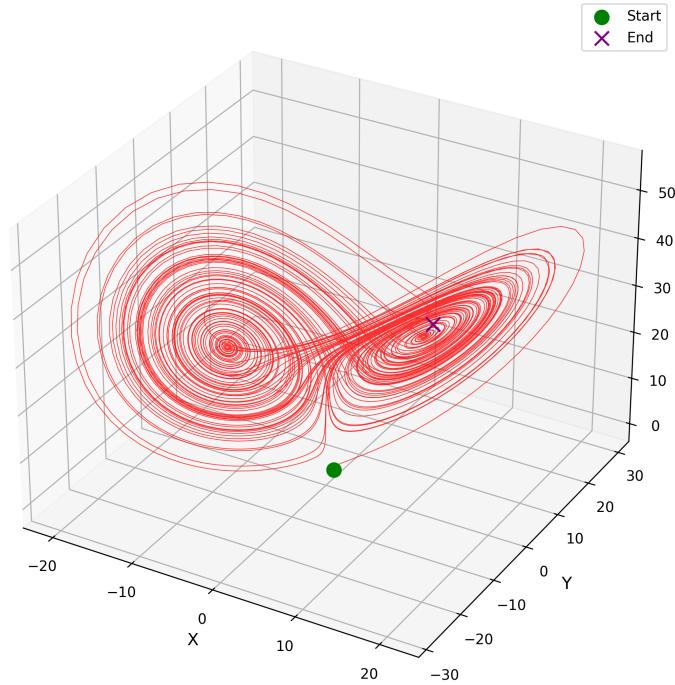


図 7: 前進オイラー法によるローレンツ方程式の軌道. 条件は図 6 と同一. 長時間積分ではアトラクタの形状が歪んでいくことが分かる.

■(2) ステップ幅と数値誤差の関係  $t = 15, 30, 60$  における  $x(t)$  とステップ幅  $\Delta t$  の関係をプロットした結果を図 8 に示す。横軸は  $\Delta t$  を対数スケールでとっている。

$t = 15$  の場合、RK4 法では今回の範囲の  $\Delta t$  で十分に収束しており、短期間であれば数値解が安定しているので、4 次精度の手法として妥当な振る舞いが確認できる。一方、前進オイラー法では  $\Delta t$  を小さくすると  $x(t)$  の値は単調ではないものある値に近づくが、RK4 法と比べて収束が遅く、粗いステップ幅では誤差が顕著である。

$t = 30$  の場合、RK4 法では十分小さい  $\Delta t$  では十分に収束していたものの  $\Delta t \lesssim 10^{-2.5}$  までステップ幅を大きくすると、徐々に値がぶれ始めた。これは RK 法の正確に計算できる条件の限界と考えられることがある。一方前進オイラー法では  $\Delta t$  の値が収束しているとは言い難く、最小の  $\Delta t$ において RK4 法とほぼ同じ値なのも偶然の一致なのかどうか判断できない。

$t = 60$  になると、 $\Delta t$  を十分小さくしても  $x(t)$  の値が一意に収束しているとは言い難く、わずかなステップ幅の違いによって  $x(t)$  が大きく変化する領域が現れる。RK4 法であっても  $\Delta t$  の減少に伴って  $x(t)$  が一定値に近づくというより、ある範囲内ではばらつく様子が見られた。これはローレンツ方程式がカオス的であり、数値誤差が指数的に増幅されるため、十分に長時間になるとステップ幅の違いが軌道の違いとして顕在化することを示している。

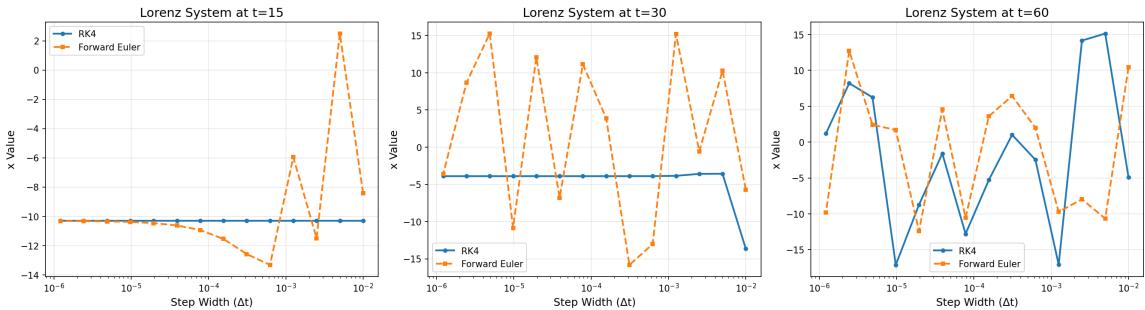


図 8:  $t = 15, 30, 60$  における  $x(t)$  のステップ幅依存性。左から順に  $t = 15, 30, 60$ 。実線が RK4 法、破線が前進オイラー法。横軸は  $\Delta t$  (対数スケール)。

■(3) 初期値に対する鋭敏な依存性 初期値の違い  $\varepsilon \in \{0, 0.1, 0.01, 0.001\}$  を与えたときの  $x(t)$  の時間波形を  $t \in [0, 50]$  および  $t \in [50, 100]$  について描画した結果を図 9、図 10 に示す。

$t \lesssim 10$  の短時間では、 $\varepsilon$  の違いにかかわらず  $x(t)$  の波形はほとんど重なっており、初期値の違いが顕著には現れない。しかし、 $t$  が増加するにつれて位相が徐々にずれ始め、 $t \approx 30$  以降では  $\varepsilon = 0.001$  のような非常に小さな擾乱でも、 $x(t)$  の波形が基準解 ( $\varepsilon = 0$ ) とほとんど無関係な挙動を示すようになる。

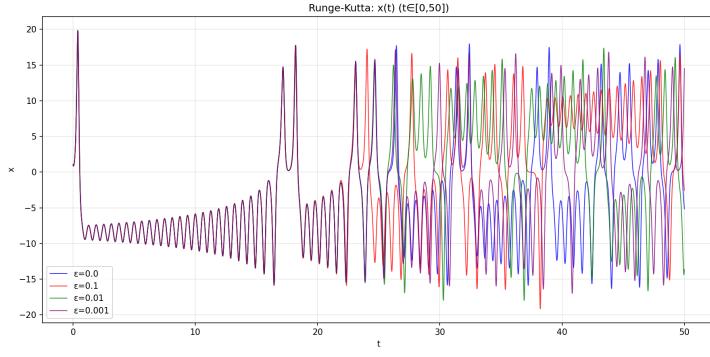


図 9: RK4 法による  $x(t)$  の時間波形 ( $t \in [0, 50]$ ).  $\varepsilon = 0, 0.1, 0.01, 0.001$  の 4 通りを重ね書きしている.

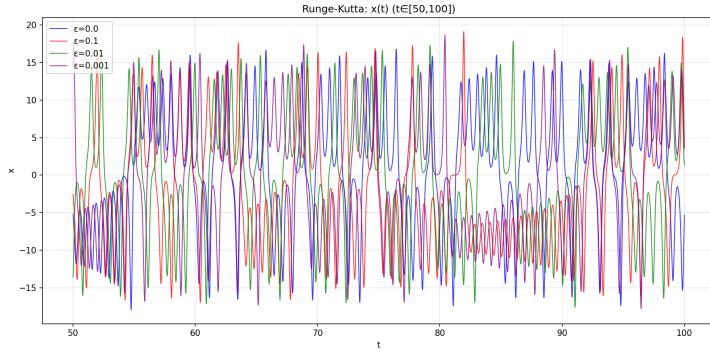


図 10: RK4 法による  $x(t)$  の時間波形 ( $t \in [50, 100]$ ). 初期値のわずかな違いが長時間後には完全に異なる軌道として現れている.

図 11 に,  $\varepsilon = 0$  を基準としたときの誤差の最大値  $E_\varepsilon(t)$  の時間発展を示す. いずれの  $\varepsilon$  についても, 初期段階では  $E_\varepsilon(t)$  が指数関数的に増大し, その後はアトラクタの大きさに対応する  $\mathcal{O}(10)$  程度の値で飽和する挙動が見られる. 初期誤差が小さいほど  $E_\varepsilon(t)$  が飽和値に達するまでに時間がかかるが, 十分長時間ではどの  $\varepsilon$  に対しても同程度の誤差レベルに達している.

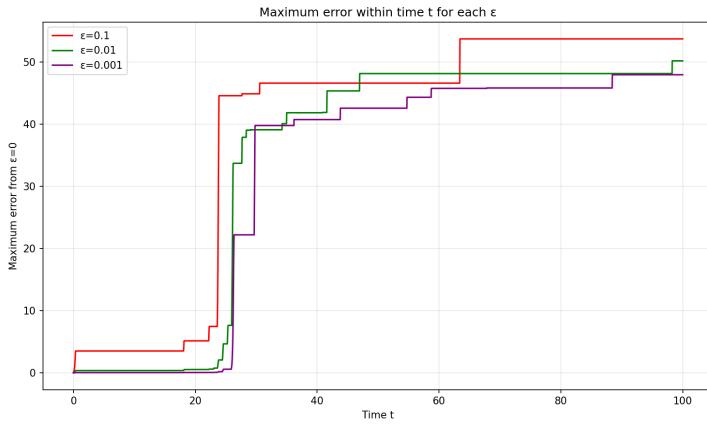


図 11: 基準解 ( $\varepsilon = 0$ ) からの誤差の最大値  $E_\varepsilon(t)$  の時間発展.  $\varepsilon = 0.1, 0.01, 0.001$  についてプロットしている.

### 8.3 考察

全体を通して,RK4 法は前進オイラー法に比べて数値解の精度が高く、ローレンツ方程式のようなカオス的系に対してもある程度安定に振る舞うことが確認できた。長時間が経過したのちには、RK4 法であっても数値誤差の増幅が顕著となり、文字通り「カオス的」な挙動を示すことが分かった。課題 7-2 で示したように、 $t = 15$ あたりまでは RK4 法で安定的に収束していたことは、課題 7-3 のグラフからも確認できる。

## 9 結論

本実験を通じて、常微分方程式の数値解法における各手法の特性を理解し、刻み幅や安定性が解の精度に与える影響を確認した。特にカオス現象の数値計算では、初期値の微小な差異が結果に大きな影響を及ぼすことを観察し、数値解法の慎重な選択と検証の重要性を認識した。

## 付録 A 使用コード一覧

レポート作成、Python の実装の一部に GitHub Copilot の補完を使用した。以下に各課題で使用した Python コードを示す。

### A.1 課題 3 のコード

Listing 1: task3.py

```
1 import numpy as np
2
3 def f (u,t =0):
4     return u
5
6 def analytical_solution (t, u0 = 1):
7     return u0 * np.exp(t)
8
9 def update_foward_euler (delta_t, u1, t1 ,f= f):
10
11     return u1 + delta_t * f(u1, t1)
12
13 def update_adam_bashforth2 (delta_t, u1,u2, t1 ,t2,f= f):
14     f1 = f(u1, t1)
15     f2 = f(u2, t2)
16     return u1 + delta_t * ( (3/2) * f1 - (1/2) * f2 )
17
18 def update_adam_bashforth3 (delta_t, u1,u2,u3, t1 ,t2,t3,f= f):
19     f1 = f(u1, t1)
20     f2 = f(u2, t2)
21     f3 = f(u3, t3)
22     return u1 + delta_t * ( (23/12) * f1 - (16/12) * f2 + (5/12) * f3 )
23
24 def update_heun (delta_t, u1, t1 ,f= f):
25     f1 = f(u1, t1)
26     u_predictor = u1 + delta_t * f1
```

```

27     t = t1 + delta_t
28     f2 = f(u_predictor, t )
29     return u1 + (delta_t / 2) * (f1 + f2)
30
31 def update_runge_kutta4 (delta_t, u1, t1 ,f= f):
32     k1 = f(u1, t1)
33     k2 = f(u1 + (delta_t / 2) * k1, t1 + (delta_t / 2))
34     k3 = f(u1 + (delta_t / 2) * k2, t1 + (delta_t / 2))
35     k4 = f(u1 + delta_t * k3, t1 + delta_t)
36     return u1 + (delta_t / 6) * (k1 + 2 * k2 + 2 * k3 + k4)
37
38 def foward_euler (n,t_start = 0, t_end = 1, u_start = 1):
39     delta_t = np.float64((t_end - t_start) / n)
40     u = np.float64(u_start)
41     t = np.float64(t_start)
42     for i in range (n):
43         u = update_foward_euler (delta_t, u, t, f)
44         t += delta_t
45     return u
46
47 def adam_bashforth2 (n,t_start = 0, t_end = 1, u_start = 1):
48     delta_t = np.float64((t_end - t_start) / n)
49     # 最初の1ステップは真値を使う
50     u2 = np.float64(u_start)
51     t2 = np.float64(t_start)
52     t1 = t2 + delta_t
53     u1 = analytical_solution (t1, u_start)
54     for i in range (1, n):
55         u_new = update_adam_bashforth2 (delta_t, u1, u2, t1, t2, f)
56         u2 = u1
57         t2 = t1
58         u1 = u_new
59         t1 += delta_t
60     return u1
61
62 def adam_bashforth3 (n,t_start = 0, t_end = 1, u_start = 1):
63     delta_t = np.float64((t_end - t_start) / n)
64     # 最初の2ステップは真値を使う
65     u3 = np.float64(u_start)
66     t3 = np.float64(t_start)
67     t2 = t3 + delta_t
68     u2 = analytical_solution (t2, u_start)
69     t1 = t2 + delta_t
70     u1 = analytical_solution (t1, u_start)
71     for i in range (2, n):
72         u_new = update_adam_bashforth3 (delta_t, u1, u2, u3, t1, t2, t3, f)
73         u3 = u2
74         t3 = t2
75         u2 = u1
76         t2 = t1
77         u1 = u_new
78         t1 += delta_t
79     return u1
80
81 def heun (n,t_start = 0, t_end = 1, u_start = 1):
82     delta_t = np.float64((t_end - t_start) / n)
83     u = np.float64(u_start)
84     t = np.float64(t_start)

```

```

85     for i in range (n):
86         u = update_heun (delta_t, u, t, f)
87         t += delta_t
88     return u
89
90 def runge_kutta4 (n,t_start = 0, t_end = 1, u_start = 1):
91     delta_t = np.float64((t_end - t_start) / n)
92     u = np.float64(u_start)
93     t = np.float64(t_start)
94     for i in range (n):
95         u = update_runge_kutta4 (delta_t, u, t, f)
96         t += delta_t
97     return u
98
99 def main (n):
100    # 更新方法を変えて誤差を確認
101
102    print("Forward Euler:", abs ( foward_euler (n) - analytical_solution (1) ) )
103    print("Adams-Bashforth 2nd order:", abs ( adam_bashforth2 (n) -
104        analytical_solution (1) ) )
105    print("Adams-Bashforth 3rd order:", abs ( adam_bashforth3 (n) -
106        analytical_solution (1) ) )
107    print("Heun method:", abs ( heun (n) - analytical_solution (1) ) )
108    print("Runge-Kutta 4th order:", abs ( runge_kutta4 (n) - analytical_solution (1) ) )
109
110    print("真の解:", analytical_solution (1) )
111    print("真の解との誤差:")
112    for pow in range(2, 10):
113        n = 2 ** pow
114        print("n =", n)
115        main (n)

```

## A.2 課題 4 のコード

**Listing 2:** task4.py

```

1 import os
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 # パラメータ
6 alpha = 10.0    # ここは元に戻す
7 beta   = 0.0
8 u0     = 1.0
9 t0     = 0.0
10 t_end = 10.0
11
12 # 右辺と真値
13 def f(u, t, alpha=alpha, beta=beta):
14     return -alpha * u + beta
15
16 def u_exact(t, alpha=alpha, beta=beta, u0=u0):
17     if alpha == 0.0:

```

```

18     return u0 + beta * t
19     return (u0 - beta/alpha) * np.exp(-alpha * t) + beta/alpha
20
21 # --- 1ステップ法 -----
22
23 def step_crank_nicolson(dt, u, t, alpha=alpha, beta=beta):
24     num = u + 0.5 * dt * (-alpha * u + beta)
25     den = 1.0 + 0.5 * dt * alpha
26     return num / den
27
28 def step_predictor_corrector(dt, u, t, alpha=alpha, beta=beta):
29     # 予測: 前進オイラー
30     u_pred = u + dt * f(u, t, alpha, beta)
31     # 修正: 後退オイラーを予測値で陽化
32     return u + dt * f(u_pred, t + dt, alpha, beta)
33
34 def step_heun(dt, u, t, alpha=alpha, beta=beta):
35     k1 = f(u, t, alpha, beta)
36     u_pred = u + dt * k1
37     k2 = f(u_pred, t + dt, alpha, beta)
38     return u + 0.5 * dt * (k1 + k2)
39
40 # --- 汎用ソルバ -----
41
42 def solve(step_func, dt_target, alpha=alpha, beta=beta):
43     N = int(round((t_end - t0) / dt_target))
44     dt = (t_end - t0) / N
45     t = np.linspace(t0, t_end, N + 1)
46     u = np.empty_like(t)
47     u[0] = u0
48     for n in range(N):
49         u[n+1] = step_func(dt, u[n], t[n], alpha, beta)
50     return t, u, dt
51
52 # --- グラフ描画 & 保存 -----
53
54 methods = [
55     ("crank_nicolson", "Crank - Nicolson", step_crank_nicolson),
56     ("predictor_corrector", "Predictor - Corrector", step_predictor_corrector),
57     ("heun", "Heun method", step_heun),
58 ]
59
60 # 手法ごとの Δt
61 dt_lists = {
62     "crank_nicolson": [0.01, 0.19, 0.205],
63     "predictor_corrector": [0.01, 0.095, 0.105],
64     "heun": [0.01, 0.19, 0.205],
65 }
66
67 save_dir = "."
68 os.makedirs(save_dir, exist_ok=True)
69
70 for fname_prefix, title, step_func in methods:
71     plt.figure(figsize=(6, 4))
72
73     # 真値
74     t_exact = np.linspace(t0, t_end, 2000)
75     plt.plot(t_exact, u_exact(t_exact), "k--", label="exact")

```

```

76
77     # 各  $\Delta t$  での数値解
78     for dt_target in dt_lists[fname_prefix]:
79         t, u, dt_actual = solve(step_func, dt_target)
80         plt.plot(t, u, label=f"$\Delta t \approx {dt_actual:.3f}$")
81
82     plt.xlabel("$t$")
83     plt.ylabel("$u$")
84     plt.title(title)
85     plt.legend()
86     plt.grid(True)
87
88     # 縦軸の範囲
89     if fname_prefix == "crank_nicolson":
90         plt.ylim(-1.0, 1.0)
91     else:
92         plt.ylim(-1.0, 10.0)
93
94     plt.tight_layout()
95     out_path = f"{save_dir}/task4_{fname_prefix}.png"
96     plt.savefig(out_path)
97     plt.close()
98
99 print("saved:", [f"task4_{m[0]}.png" for m in methods])

```

### A.3 課題 5 のコード

**Listing 3:** task5.py

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # ODE:  $u' = -2u + 1$ ,  $u(0) = 1$ 
5 def f(u, t):
6     return -2.0 * u + 1.0
7
8 def exact_solution(t, u0=1.0):
9     #  $u(t) = 1/2 + (u_0 - 1/2) e^{-2t}$ 
10    return 0.5 + (u0 - 0.5) * np.exp(-2.0 * t)
11
12 def update_two_step(delta_t, u_nm2, u_nm1):
13     """
14     2 段法:
15          $u_n = u_{n-2} + 2 \Delta t f(t_{n-1}, u_{n-1})$ 
16         ここでは  $f$  は  $t$  に依存しないので  $t$  は使わない。
17     """
18    return u_nm2 + 2.0 * delta_t * f(u_nm1, 0.0)
19
20 def solve_two_step(delta_t, t_start=0.0, t_end=10.0, u_start=1.0):
21     """
22     2 段法で  $t_{start}$  から  $t_{end}$  まで解く。
23      $u_0 = u_{start}$ ,
24      $u_1$  は真値  $u_{exact}(\Delta t)$  を用いて与える。
25     """
26     # ステップ数を整数に丸めて  $dt$  を取り直す
27     n_steps = int(round((t_end - t_start) / delta_t))

```

```

28     dt = (t_end - t_start) / n_steps
29
30     t = np.linspace(t_start, t_end, n_steps + 1)
31     u = np.zeros_like(t)
32
33     u[0] = float(u_start)
34     u[1] = float(exact_solution(dt, u_start))
35
36     for n in range(2, n_steps + 1):
37         u[n] = update_two_step(dt, u[n-2], u[n-1])
38
39     return t, u, dt
40
41 if __name__ == "__main__":
42     T = 10.0
43     dt_list = [0.10, 0.05, 0.01]
44
45     # 図の作成
46     plt.figure(figsize=(6, 4))
47
48     # 真値
49     t_exact = np.linspace(0.0, T, 2000)
50     plt.plot(t_exact, exact_solution(t_exact), "k--", label="exact")
51
52     # 各  $\Delta t$  での数値解
53     for dt_target in dt_list:
54         t_num, u_num, dt_actual = solve_two_step(dt_target, 0.0, T, 1.0)
55         plt.plot(t_num, u_num, label=f"$\Delta t \approx {dt_actual:.3f}$")
56
57     plt.xlabel("$t$")
58     plt.ylabel("$u$")
59     plt.title("Two-step method in Task 5")
60     plt.grid(True)
61     plt.legend()
62
63     # 発散の様子が見えるよう縦軸を固定
64     plt.ylim(-200.0, 200.0)
65
66     plt.tight_layout()
67     plt.savefig("task5_two_step.png")
68     plt.show()
69
70     #  $t = 10$  における値と誤差も出力（表用）
71     uT_exact = exact_solution(T)
72     print(f"t = 10 の真値:", uT_exact)
73     for dt_target in dt_list:
74         t_num, u_num, dt_actual = solve_two_step(dt_target, 0.0, T, 1.0)
75         err = abs(u_num[-1] - uT_exact)
76         print(f"Δt ≈ {dt_actual:.3f} → u_N = {u_num[-1]:.6e}, error = {err:.6e}")
    )

```

#### A.4 課題 6 のコード

Listing 4: task6.py

```

1 import numpy as np

```

```

2 import matplotlib.pyplot as plt
3
4 def f(u, t=0):
5     return (u - 1) * u
6
7 def analytic_solution(t, u0):
8     if u0 == 0:
9         return 0.0
10    if u0 == 1:
11        return 1.0
12    return u0 / (u0 + (1 - u0) * np.exp(t))
13
14 def update_runge_kutta4(delta_t, u1, t1, f=f):
15     k1 = f(u1, t1)
16     k2 = f(u1 + (delta_t / 2) * k1, t1 + (delta_t / 2))
17     k3 = f(u1 + (delta_t / 2) * k2, t1 + (delta_t / 2))
18     k4 = f(u1 + delta_t * k3, t1 + delta_t)
19     return u1 + (delta_t / 6) * (k1 + 2 * k2 + 2 * k3 + k4)
20
21 def runge_kutta4_arc_length(n, t_start=0, t_end=1, u_start=1, return_trajectory=False):
22     delta_t_default = np.float64((t_end - t_start) / n)
23     delta_t = delta_t_default
24     u = np.float64(u_start)
25     t = np.float64(t_start)
26
27     if return_trajectory:
28         t_values = [t]
29         u_values = [u]
30
31     for i in range(1000000):
32         u = update_runge_kutta4(delta_t, u, t, f)
33         if abs(u) > 1e5:
34             u = np.float64('inf')
35             if return_trajectory:
36                 t_values.append(t)
37                 u_values.append(u)
38             break
39
40         delta_t = delta_t_default / np.sqrt(1 + (f(u, t))**2)
41         t += delta_t
42
43         if return_trajectory:
44             t_values.append(t)
45             u_values.append(u)
46
47         if t > t_end:
48             break
49
50     if return_trajectory:
51         return u, t, np.array(t_values), np.array(u_values)
52     return u
53
54 def estimate_blowup_time_from_atan(t_vals, u_vals):
55     """
56     arctan(u) 空間で、最後の有限な2点から
57     blow-up time の推定値 t_hat を返す。
58     """

```

```

59     finite_mask = np.isfinite(u_vals)
60     t_finite = t_vals[finite_mask]
61     u_finite = u_vals[finite_mask]
62
63     if t_finite.size < 2:
64         return None
65
66     # 最後の2点
67     t1, t2 = t_finite[-2], t_finite[-1]
68     y1, y2 = np.arctan(u_finite[-2]), np.arctan(u_finite[-1])
69
70     if t2 == t1:
71         return None
72
73     # 2点を通る直線  $y = m t + b$ 
74     m = (y2 - y1) / (t2 - t1)
75     if m == 0.0:
76         return None
77
78     b = y2 - m * t2
79
80     #  $y = \pi/2$  と交わる  $t$  を発散時刻の推定値とする
81     t_hat = (0.5 * np.pi - b) / m
82
83     return t_hat
84
85
86 # プロット用のデータを収集
87 fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 12))
88
89 # ノーマルバージョン（上）
90 for u_start in np.arange(0, 2.1, 0.2):
91     u_final, t_final, t_vals, u_vals = runge_kutta4_arc_length(
92         10000, 0, 5, u_start, return_trajectory=True
93     )
94     ax1.plot(t_vals, u_vals, label=f'u(0) = {u_start:.1f}', linewidth=2)
95     approx = u_vals[-1] if len(u_vals) > 0 else np.nan
96
97     # 解析解
98     analytic_vals = analytic_solution(t_vals, u_start)
99
100    # arctan(u) 空間での誤差
101    u_vals_atan = np.arctan(u_vals)
102    analytic_vals_atan = np.arctan(analytic_vals)
103
104    abs_errors_atan = np.abs(u_vals_atan - analytic_vals_atan)
105    mean_abs_error_atan = np.mean(abs_errors_atan)
106    max_abs_error_atan = np.max(abs_errors_atan)
107
108    msg = (
109        f"u_start: {u_start:.1f}, final t: {t_final:.4f}, approx: {approx:.6f}, "
110        f"mean |atan err|: {mean_abs_error_atan:.6e}, "
111        f"max |atan err|: {max_abs_error_atan:.6e}"
112    )
113
114    # u_start > 1 の場合は blow-up 時刻の推定値と解析解を比較
115    if u_start > 1.0:
116        t_hat = estimate_blowup_time_from_atan(t_vals, u_vals)

```

```

117     if t_hat is not None:
118         # 解析的 blow-up 時刻
119         t_blow = np.log(u_start / (u_start - 1.0))
120
121         diff      = t_hat - t_blow          # 推定値 - 真値
122         rel_error = diff / t_blow        # 相対誤差
123
124         msg += (
125             f", t_blow_exact: {t_blow:.6f}, "
126             f"t_hat: {t_hat:.6f}, "
127             f"diff: {diff:.3e}, "
128             f"rel_error: {rel_error:.3e}"
129         )
130     else:
131         msg += ", t_hat: N/A"
132
133     print(msg)
134
135 ax1.set_xlabel('t', fontsize=14)
136 ax1.set_ylabel('u(t)', fontsize=14)
137 ax1.set_title('Normal version', fontsize=16)
138 ax1.set_xlim(0, 5)
139 ax1.set_ylim(0, 7)
140 ax1.tick_params(axis='both', labelsize=12)
141 ax1.legend(bbox_to_anchor=(1.05, 1), loc='upper left', fontsize=12)
142 ax1.grid(True, alpha=0.3)
143
144 # アークタンジェントバージョン（下）
145 for u_start in np.arange(0, 2.1, 0.2):
146     _, t_final, t_vals, u_vals = runge_kutta4_arc_length(
147         10000, 0, 5, u_start, return_trajectory=True
148     )
149     u_vals_atan = np.arctan(u_vals)
150     ax2.plot(t_vals, u_vals_atan, label=f'u(0) = {u_start:.1f}', linewidth=2)
151
152 ax2.set_xlabel('t', fontsize=14)
153 ax2.set_ylabel('arctan(u(t))', fontsize=14)
154 ax2.set_title('Arctangent version', fontsize=16)
155 ax2.set_xlim(0, 5)
156 ax2.set_ylim(0, np.pi/2 * 1.05)
157 ax2.set_yticks([0, np.pi/6, np.pi/4, np.pi/3, np.pi/2])
158 ax2.set_yticklabels(['0', 'π/6', 'π/4', 'π/3', 'π/2'], fontsize=12)
159 ax2.tick_params(axis='x', labelsize=12)
160 ax2.legend(bbox_to_anchor=(1.05, 1), loc='upper left', fontsize=12)
161 ax2.grid(True, alpha=0.3)
162
163 plt.tight_layout()
164 plt.savefig('task6_trajectory.png', dpi=300, bbox_inches='tight')
165 plt.show()
166
167 print("\nグラフを 'task6_trajectory.png' として保存しました。")

```

## A.5 課題 7 のコード

Listing 5: task7\_1.py

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 sigma = 10.0
5 beta = 8.0 / 3.0
6 r = 28.0
7 def x_dot (x, y, z, t=0):
8     return sigma * (y - x)
9
10 def y_dot (x, y, z, t=0):
11
12     return x * (r - z) - y
13
14 def z_dot (x, y, z, t=0):
15     return x * y - beta * z
16
17
18 def update_runge_kutta4_lorenz (delta_t, u1, t1 ,f_x= x_dot, f_y= y_dot, f_z= z_dot):
19     x1, y1, z1 = u1
20
21     k1x = f_x(x1, y1, z1, t1)
22     k1y = f_y(x1, y1, z1, t1)
23     k1z = f_z(x1, y1, z1, t1)
24
25     k2x = f_x(x1 + (delta_t / 2) * k1x, y1 + (delta_t / 2) * k1y, z1 + (delta_t / 2) *
26                 k1z, t1 + (delta_t / 2))
27     k2y = f_y(x1 + (delta_t / 2) * k1x, y1 + (delta_t / 2) * k1y, z1 + (delta_t / 2) *
28                 k1z, t1 + (delta_t / 2))
29     k2z = f_z(x1 + (delta_t / 2) * k1x, y1 + (delta_t / 2) * k1y, z1 + (delta_t / 2) *
30                 k1z, t1 + (delta_t / 2))
31
32     k3x = f_x(x1 + (delta_t / 2) * k2x, y1 + (delta_t / 2) * k2y, z1 + (delta_t / 2) *
33                 k2z, t1 + (delta_t / 2))
34     k3y = f_y(x1 + (delta_t / 2) * k2x, y1 + (delta_t / 2) * k2y, z1 + (delta_t / 2) *
35                 k2z, t1 + (delta_t / 2))
36     k3z = f_z(x1 + (delta_t / 2) * k2x, y1 + (delta_t / 2) * k2y, z1 + (delta_t / 2) *
37                 k2z, t1 + (delta_t / 2))
38
39     k4x = f_x(x1 + delta_t * k3x, y1 + delta_t * k3y, z1 + delta_t * k3z, t1 + delta_t
40                 )
41     k4y = f_y(x1 + delta_t * k3x, y1 + delta_t * k3y, z1 + delta_t * k3z, t1 + delta_t
42                 )
43     k4z = f_z(x1 + delta_t * k3x, y1 + delta_t * k3y, z1 + delta_t * k3z, t1 + delta_t
44                 )
45
46     x_new = x1 + (delta_t / 6) * (k1x + 2 * k2x + 2 * k3x + k4x)
47     y_new = y1 + (delta_t / 6) * (k1y + 2 * k2y + 2 * k3y + k4y)
48     z_new = z1 + (delta_t / 6) * (k1z + 2 * k2z + 2 * k3z + k4z)
49
50     return np.array([x_new, y_new, z_new])
51
52 def update_forward_euler_lorenz(delta_t, u1, t1, f_x=x_dot, f_y=y_dot, f_z=z_dot):
53     x1, y1, z1 = u1
54
55     x_new = x1 + delta_t * f_x(x1, y1, z1, t1)
56     y_new = y1 + delta_t * f_y(x1, y1, z1, t1)
57     z_new = z1 + delta_t * f_z(x1, y1, z1, t1)
58
59     return np.array([x_new, y_new, z_new])

```

```

48 def forward_euler_lorenz(n, t_start=0, t_end=50, u_start=(1, 0, 0), return_trajectory=False):
49     delta_t = np.float64((t_end - t_start) / n)
50     u = np.array(u_start, dtype=np.float64)
51     t = np.float64(t_start)
52
53     if return_trajectory:
54         t_values = [t]
55         trajectory = [u.copy()]
56
57     for i in range(n):
58         u = update_forward_euler_lorenz(delta_t, u, t, x_dot, y_dot, z_dot)
59         t += delta_t
60
61         if return_trajectory:
62             t_values.append(t)
63             trajectory.append(u.copy())
64
65     if return_trajectory:
66         return u, np.array(t_values), np.array(trajectory)
67     return u
68
69 def runge_kutta4_lorenz (n,t_start = 0, t_end = 50, u_start = (1,0,0),
70 return_trajectory=False):
71     delta_t = np.float64((t_end - t_start) / n)
72     u = np.array(u_start, dtype=np.float64)
73     t = np.float64(t_start)
74
75     if return_trajectory:
76         t_values = [t]
77         trajectory = [u.copy()]
78
79     for i in range (n):
80         u = update_runge_kutta4_lorenz (delta_t, u, t, x_dot, y_dot, z_dot)
81         t += delta_t
82
83         if return_trajectory:
84             t_values.append(t)
85             trajectory.append(u.copy())
86
87     if return_trajectory:
88         return u, np.array(t_values), np.array(trajectory)
89     return u
90
91 # パラメータ設定
92 n = 10000
93 t_start = 0
94 t_end = 100
95 u_start = (1, 0, 0)
96
97 print(f"初期条件: x={u_start[0]}, y={u_start[1]}, z={u_start[2]}")
98 print(f"時間範囲: t={t_start} から t={t_end}")
99 print(f"ステップ数: {n}")
100 print()
101
102 # ルンゲ・クッタ法で軌道を計算
103 print("ルンゲ・クッタ法で計算中...")

```

```

104 u_final_rk, t_values_rk, trajectory_rk = runge_kutta4_lorenz(
105     n, t_start, t_end, u_start, return_trajectory=True
106 )
107 print(f"ルンゲ・クッタ法 最終状態: x={u_final_rk[0]:.4f}, y={u_final_rk[1]:.4f}, z={u_final_rk[2]:.4f}")
108
109 # 前進オイラー法で軌道を計算
110 print("前進オイラー法で計算中...")
111 u_final_euler, t_values_euler, trajectory_euler = forward_euler_lorenz(
112     n, t_start, t_end, u_start, return_trajectory=True
113 )
114 print(f"前進オイラー法 最終状態: x={u_final_euler[0]:.4f}, y={u_final_euler[1]:.4f}, z={u_final_euler[2]:.4f}")
115 print()
116
117 # ルンゲ・クッタ法の3次元プロット
118 fig1 = plt.figure(figsize=(12, 9))
119 ax1 = fig1.add_subplot(111, projection='3d')
120
121 ax1.plot(trajectory_rk[:, 0], trajectory_rk[:, 1], trajectory_rk[:, 2], linewidth=0.5,
122           color='blue', alpha=0.8)
123 ax1.scatter(trajectory_rk[0, 0], trajectory_rk[0, 1], trajectory_rk[0, 2], color='green',
124             s=100, label='Start', marker='o')
125 ax1.scatter(trajectory_rk[-1, 0], trajectory_rk[-1, 1], trajectory_rk[-1, 2], color='red',
126             s=100, label='End', marker='x')
127
128 ax1.set_xlabel('X', fontsize=12)
129 ax1.set_ylabel('Y', fontsize=12)
130 ax1.set_zlabel('Z', fontsize=12)
131 ax1.set_title('Lorenz Attractor - Runge-Kutta Method', fontsize=14)
132 ax1.legend()
133 ax1.grid(True, alpha=0.3)
134
135 # ルンゲ・クッタ法のプロットを保存
136 fig1.savefig('lorenz_runge_kutta.png', dpi=300, bbox_inches='tight')
137 print("ルンゲ・クッタ法のプロットを 'lorenz_runge_kutta.png' に保存しました")
138
139 # 前進オイラー法の3次元プロット
140 fig2 = plt.figure(figsize=(12, 9))
141 ax2 = fig2.add_subplot(111, projection='3d')
142
143 ax2.plot(trajectory_euler[:, 0], trajectory_euler[:, 1], trajectory_euler[:, 2],
144           linewidth=0.5, color='red', alpha=0.8)
145 ax2.scatter(trajectory_euler[0, 0], trajectory_euler[0, 1], trajectory_euler[0, 2],
146             color='green', s=100, label='Start', marker='o')
147 ax2.scatter(trajectory_euler[-1, 0], trajectory_euler[-1, 1], trajectory_euler[-1, 2],
148             color='purple', s=100, label='End', marker='x')
149
150 ax2.set_xlabel('X', fontsize=12)
151 ax2.set_ylabel('Y', fontsize=12)
152 ax2.set_zlabel('Z', fontsize=12)
153 ax2.set_title('Lorenz Attractor - Forward Euler Method', fontsize=14)
154 ax2.legend()
155 ax2.grid(True, alpha=0.3)
156
157 # 前進オイラー法のプロットを保存
158 fig2.savefig('lorenz_forward_euler.png', dpi=300, bbox_inches='tight')
159 print("前進オイラー法のプロットを 'lorenz_forward_euler.png' に保存しました")

```

```

154
155 plt.tight_layout()
156 plt.show()

```

**Listing 6:** task7\_2.py

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 sigma = 10.0
5 beta = 8.0 / 3.0
6 r = 28.0
7
8 def x_dot (x, y, z, t=0):
9     return sigma * (y - x)
10
11 def y_dot (x, y, z, t=0):
12     return x * (r - z) - y
13
14 def z_dot (x, y, z, t=0):
15     return x * y - beta * z
16
17 def update_runge_kutta4_lorenz (delta_t, u1, t1 ,f_x= x_dot, f_y= y_dot, f_z= z_dot):
18     x1, y1, z1 = u1
19
20     k1x = f_x(x1, y1, z1, t1)
21     k1y = f_y(x1, y1, z1, t1)
22     k1z = f_z(x1, y1, z1, t1)
23
24     k2x = f_x(x1 + (delta_t / 2) * k1x, y1 + (delta_t / 2) * k1y, z1 + (delta_t / 2) *
25                 k1z, t1 + (delta_t / 2))
26     k2y = f_y(x1 + (delta_t / 2) * k1x, y1 + (delta_t / 2) * k1y, z1 + (delta_t / 2) *
27                 k1z, t1 + (delta_t / 2))
28     k2z = f_z(x1 + (delta_t / 2) * k1x, y1 + (delta_t / 2) * k1y, z1 + (delta_t / 2) *
29                 k1z, t1 + (delta_t / 2))
30
31     k3x = f_x(x1 + (delta_t / 2) * k2x, y1 + (delta_t / 2) * k2y, z1 + (delta_t / 2) *
32                 k2z, t1 + (delta_t / 2))
33     k3y = f_y(x1 + (delta_t / 2) * k2x, y1 + (delta_t / 2) * k2y, z1 + (delta_t / 2) *
34                 k2z, t1 + (delta_t / 2))
35     k3z = f_z(x1 + (delta_t / 2) * k2x, y1 + (delta_t / 2) * k2y, z1 + (delta_t / 2) *
36                 k2z, t1 + (delta_t / 2))
37
38     k4x = f_x(x1 + delta_t * k3x, y1 + delta_t * k3y, z1 + delta_t * k3z, t1 + delta_t
39 )
40     k4y = f_y(x1 + delta_t * k3x, y1 + delta_t * k3y, z1 + delta_t * k3z, t1 + delta_t
41 )
42     k4z = f_z(x1 + delta_t * k3x, y1 + delta_t * k3y, z1 + delta_t * k3z, t1 + delta_t
43 )
44
45     x_new = x1 + (delta_t / 6) * (k1x + 2 * k2x + 2 * k3x + k4x)
46     y_new = y1 + (delta_t / 6) * (k1y + 2 * k2y + 2 * k3y + k4y)
47     z_new = z1 + (delta_t / 6) * (k1z + 2 * k2z + 2 * k3z + k4z)
48
49     return np.array([x_new, y_new, z_new])
50
51 def update_forward_euler_lorenz(delta_t, u1, t1, f_x=x_dot, f_y=y_dot, f_z=z_dot):
52     x1, y1, z1 = u1
53     x_new = x1 + delta_t * f_x(x1, y1, z1, t1)

```

```

44     y_new = y1 + delta_t * f_y(x1, y1, z1, t1)
45     z_new = z1 + delta_t * f_z(x1, y1, z1, t1)
46     return np.array([x_new, y_new, z_new])
47
48 # 共通のサンプリング関数：軌道は保持せず、指定時刻だけ返す
49 def sample_lorenz(method, n, t_start, t_end, u_start, sample_times):
50     delta_t = np.float64((t_end - t_start) / n)
51     u = np.array(u_start, dtype=np.float64)
52     t = np.float64(t_start)
53
54     sample_times = np.array(sample_times, dtype=np.float64)
55     # サンプルしたい時刻に対応するステップ番号
56     idx_map = {
57         int(round((ts - t_start) / delta_t)): i
58         for i, ts in enumerate(sample_times)
59     }
59
60     samples = np.empty((len(sample_times), 3), dtype=np.float64)
61
62     for k in range(n + 1):
63         # k番目ステップの状態を必要に応じて保存
64         if k in idx_map:
65             samples[idx_map[k]] = u
66
67         if k == n:
68             break
69
70         if method == "rk4":
71             u = update_runge_kutta4_lorenz(delta_t, u, t, x_dot, y_dot, z_dot)
72         elif method == "fe":
73             u = update_forward_euler_lorenz(delta_t, u, t, x_dot, y_dot, z_dot)
74         else:
75             raise ValueError("method must be 'rk4' or 'fe'")
76
77         t += delta_t
78
79     return samples
80
81 # ここからメイン部分
82 t_start = 0
83 t_end = 100
84 u_start = (1, 0, 0)
85 sample_times = [15.0, 30.0, 60.0]
86
87 rk_vals_15 = []
88 fe_vals_15 = []
89 rk_vals_30 = []
90 fe_vals_30 = []
91 rk_vals_60 = []
92 fe_vals_60 = []
93 n_list = []
94
95
96 # 複数のステップ数で比較
97 for pow in range(14):
98     n = 10000 * (2 ** pow)
99     n_list.append(n)
100
101 rk_samples = sample_lorenz("rk4", n, t_start, t_end, u_start, sample_times)

```

```

102     fe_samples = sample_lorenz("fe", n, t_start, t_end, u_start, sample_times)
103
104     rk_vals_15.append(rk_samples[0])
105     rk_vals_30.append(rk_samples[1])
106     rk_vals_60.append(rk_samples[2])
107
108     fe_vals_15.append(fe_samples[0])
109     fe_vals_30.append(fe_samples[1])
110     fe_vals_60.append(fe_samples[2])
111
112     rk_vals_15 = np.array(rk_vals_15)
113     fe_vals_15 = np.array(fe_vals_15)
114     rk_vals_30 = np.array(rk_vals_30)
115     fe_vals_30 = np.array(fe_vals_30)
116     rk_vals_60 = np.array(rk_vals_60)
117     fe_vals_60 = np.array(fe_vals_60)
118
119 # x軸をΔt（ステップ幅）に設定
120 delta_t_values = [(100 - 0) / n for n in n_list]
121
122 # プロット（3つの時刻を別々のグラフに）
123 fig, axes = plt.subplots(1, 3, figsize=(18, 5))
124
125 # t=15のグラフ
126 axes[0].plot(delta_t_values, rk_vals_15[:, 0], 'o--', label='RK4', linewidth=2,
127                 markersize=4)
127 axes[0].plot(delta_t_values, fe_vals_15[:, 0], 's--', label='Forward Euler', linewidth
128                 =2, markersize=4)
128 axes[0].set_xlabel('Step Width ( $\Delta t$ )', fontsize=12)
129 axes[0].set_ylabel('x Value', fontsize=12)
130 axes[0].set_title('Lorenz System at t=15', fontsize=14)
131 axes[0].set_xscale('log')
132 axes[0].legend()
133 axes[0].grid(True, alpha=0.3)
134
135 # t=30のグラフ
136 axes[1].plot(delta_t_values, rk_vals_30[:, 0], 'o--', label='RK4', linewidth=2,
137                 markersize=4)
137 axes[1].plot(delta_t_values, fe_vals_30[:, 0], 's--', label='Forward Euler', linewidth
138                 =2, markersize=4)
138 axes[1].set_xlabel('Step Width ( $\Delta t$ )', fontsize=12)
139 axes[1].set_ylabel('x Value', fontsize=12)
140 axes[1].set_title('Lorenz System at t=30', fontsize=14)
141 axes[1].set_xscale('log')
142 axes[1].legend()
143 axes[1].grid(True, alpha=0.3)
144
145 # t=60のグラフ
146 axes[2].plot(delta_t_values, rk_vals_60[:, 0], 'o--', label='RK4', linewidth=2,
147                 markersize=4)
147 axes[2].plot(delta_t_values, fe_vals_60[:, 0], 's--', label='Forward Euler', linewidth
148                 =2, markersize=4)
148 axes[2].set_xlabel('Step Width ( $\Delta t$ )', fontsize=12)
149 axes[2].set_ylabel('x Value', fontsize=12)
150 axes[2].set_title('Lorenz System at t=60', fontsize=14)
151 axes[2].set_xscale('log')
152 axes[2].legend()
153 axes[2].grid(True, alpha=0.3)

```

```

154
155 plt.tight_layout()
156 plt.savefig('lorenz_comparison.png', dpi=150, bbox_inches='tight')
157 plt.show()

```

**Listing 7:** task7\_3.py

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 sigma = 10.0
5 beta = 8.0 / 3.0
6 r = 28.0
7 def x_dot (x, y, z, t=0):
8     return sigma * (y - x)
9
10 def y_dot (x, y, z, t=0):
11     return x * (r - z) - y
12
13 def z_dot (x, y, z, t=0):
14     return x * y - beta * z
15
16
17
18 def update_runge_kutta4_lorenz (delta_t, u1, t1 ,f_x= x_dot, f_y= y_dot, f_z= z_dot):
19     x1, y1, z1 = u1
20
21     k1x = f_x(x1, y1, z1, t1)
22     k1y = f_y(x1, y1, z1, t1)
23     k1z = f_z(x1, y1, z1, t1)
24
25     k2x = f_x(x1 + (delta_t / 2) * k1x, y1 + (delta_t / 2) * k1y, z1 + (delta_t / 2) *
26                 k1z, t1 + (delta_t / 2))
27     k2y = f_y(x1 + (delta_t / 2) * k1x, y1 + (delta_t / 2) * k1y, z1 + (delta_t / 2) *
28                 k1z, t1 + (delta_t / 2))
29     k2z = f_z(x1 + (delta_t / 2) * k1x, y1 + (delta_t / 2) * k1y, z1 + (delta_t / 2) *
30                 k1z, t1 + (delta_t / 2))
31
32     k3x = f_x(x1 + (delta_t / 2) * k2x, y1 + (delta_t / 2) * k2y, z1 + (delta_t / 2) *
33                 k2z, t1 + (delta_t / 2))
34     k3y = f_y(x1 + (delta_t / 2) * k2x, y1 + (delta_t / 2) * k2y, z1 + (delta_t / 2) *
35                 k2z, t1 + (delta_t / 2))
36     k3z = f_z(x1 + (delta_t / 2) * k2x, y1 + (delta_t / 2) * k2y, z1 + (delta_t / 2) *
37                 k2z, t1 + (delta_t / 2))
38
39     k4x = f_x(x1 + delta_t * k3x, y1 + delta_t * k3y, z1 + delta_t * k3z, t1 + delta_t
40                 )
41     k4y = f_y(x1 + delta_t * k3x, y1 + delta_t * k3y, z1 + delta_t * k3z, t1 + delta_t
42                 )
43     k4z = f_z(x1 + delta_t * k3x, y1 + delta_t * k3y, z1 + delta_t * k3z, t1 + delta_t
44                 )
45
46     x_new = x1 + (delta_t / 6) * (k1x + 2 * k2x + 2 * k3x + k4x)
47     y_new = y1 + (delta_t / 6) * (k1y + 2 * k2y + 2 * k3y + k4y)
48     z_new = z1 + (delta_t / 6) * (k1z + 2 * k2z + 2 * k3z + k4z)
49     return np.array([x_new, y_new, z_new])
50
51 def update_forward_euler_lorenz(delta_t, u1, t1, f_x=x_dot, f_y=y_dot, f_z=z_dot):
52     x1, y1, z1 = u1

```

```

43     x_new = x1 + delta_t * f_x(x1, y1, z1, t1)
44     y_new = y1 + delta_t * f_y(x1, y1, z1, t1)
45     z_new = z1 + delta_t * f_z(x1, y1, z1, t1)
46     return np.array([x_new, y_new, z_new])
47
48 def forward_euler_lorenz(n, t_start=0, t_end=50, u_start=(1, 0, 0), return_trajectory=False):
49     delta_t = np.float64((t_end - t_start) / n)
50     u = np.array(u_start, dtype=np.float64)
51     t = np.float64(t_start)
52
53     if return_trajectory:
54         t_values = [t]
55         trajectory = [u.copy()]
56
57     for i in range(n):
58         u = update_forward_euler_lorenz(delta_t, u, t, x_dot, y_dot, z_dot)
59         t += delta_t
60
61         if return_trajectory:
62             t_values.append(t)
63             trajectory.append(u.copy())
64
65     if return_trajectory:
66         return u, np.array(t_values), np.array(trajectory)
67     return u
68
69 def runge_kutta4_lorenz (n,t_start = 0, t_end = 50, u_start = (1,0,0),
70 return_trajectory=False):
71     delta_t = np.float64((t_end - t_start) / n)
72     u = np.array(u_start, dtype=np.float64)
73     t = np.float64(t_start)
74
75     if return_trajectory:
76         t_values = [t]
77         trajectory = [u.copy()]
78
79     for i in range (n):
80         u = update_runge_kutta4_lorenz (delta_t, u, t, x_dot, y_dot, z_dot)
81         t += delta_t
82
83         if return_trajectory:
84             t_values.append(t)
85             trajectory.append(u.copy())
86
87     if return_trajectory:
88         return u, np.array(t_values), np.array(trajectory)
89     return u
90
91 # パラメータ設定
92 n=100000
93 t_start = 0
94 t_end = 100
95
96 epsilons = [0.0, 0.1, 0.01, 0.001]
97 colors = ['blue', 'red', 'green', 'purple']
98

```

```

99 # 各 ε に対する軌道を計算
100 trajectories_rk = []
101 trajectories_fe = []
102 t_values_list = []
103
104 for epsilon in epsilons:
105     x_start = 1.0 + epsilon
106     u_start = (x_start, 0, 0)
107     u_final_rk, t_values_rk, trajectory_rk = runge_kutta4_lorenz(n, t_start, t_end,
108                     u_start, return_trajectory=True)
109     u_final_fe, t_values_fe, trajectory_fe = forward_euler_lorenz(n, t_start, t_end,
110                     u_start, return_trajectory=True)
111
112     trajectories_rk.append(trajectory_rk)
113     trajectories_fe.append(trajectory_fe)
114     t_values_list.append(t_values_rk)
115
116 # グラフ作成: t ∈ [0, 50]
117 fig1 = plt.figure(figsize=(12, 6))
118
119 # Runge-Kutta法: x(t)
120 for i, epsilon in enumerate(epsilons):
121     mask = t_values_list[i] <= 50
122     plt.plot(t_values_list[i][mask], trajectories_rk[i][mask, 0],
123             color=colors[i], label=f'ε={epsilon}', alpha=0.8, linewidth=1.0)
124 plt.xlabel('t')
125 plt.ylabel('x')
126 plt.title('Runge-Kutta: x(t) (t ∈ [0,50])')
127 plt.legend()
128 plt.grid(True, alpha=0.3)
129
130 plt.tight_layout()
131 plt.savefig('lorenz_x_t0_50.png', dpi=150)
132 plt.show()
133
134 # グラフ作成: t ∈ [50, 100]
135 fig2 = plt.figure(figsize=(12, 6))
136
137 # Runge-Kutta法: x(t)
138 for i, epsilon in enumerate(epsilons):
139     mask = t_values_list[i] > 50
140     plt.plot(t_values_list[i][mask], trajectories_rk[i][mask, 0],
141             color=colors[i], label=f'ε={epsilon}', alpha=0.8, linewidth=1.0)
142 plt.xlabel('t')
143 plt.ylabel('x')
144 plt.title('Runge-Kutta: x(t) (t ∈ [50,100])')
145 plt.legend()
146 plt.grid(True, alpha=0.3)
147
148 plt.tight_layout()
149 plt.savefig('lorenz_x_t50_100.png', dpi=150)
150 plt.show()
151
152 # ε=0との誤差の最大値を計算してプロット
153 fig3 = plt.figure(figsize=(10, 6))
154
155 # ε=0の軌道(基準)
156 ref_trajectory = trajectories_rk[0] # epsilons[0] = 0.0

```

```

155
156 for i, epsilon in enumerate(epsilon[1:], start=1): # ε = 0 以外
157     # 全時刻の誤差を一度に計算
158     errors = np.linalg.norm(trajectories_rk[i] - ref_trajectory, axis=1)
159
160     # 累積最大値を計算 (各時刻 t までの最大誤差)
161     max_errors = np.maximum.accumulate(errors)
162
163     plt.plot(t_values_list[i], max_errors, color=colors[i], label=f'ε = {epsilon}', linewidth=1.5)
164
165 plt.xlabel('Time t')
166 plt.ylabel('Maximum error from ε = 0')
167 plt.title('Maximum error within time t for each ε')
168 plt.legend()
169 plt.grid(True, alpha=0.3)
170 plt.tight_layout()
171 plt.savefig('lorenz_max_error_vs_time.png', dpi=150)
172 plt.show()

```

## 参考文献

- [1] 数理工学実験（2025 年度配布資料）.