数理工学実験
常微分方程式

滝本亘（学籍番号 1029-33-1175）

2022 年 10 月 31 日

1　課題 3

1.1　はじめに

常微分方程式の初期値問題

$$u' = u, \quad u(0) = 1 \tag{1.1}$$

を考える。ステップ幅を Δt として $t = n\Delta t$ における数値解 $u(t, \Delta t)$ とする。また $t = 1$ における数値解と厳密解 $u_{exact}(t)$ を

$$E(\Delta t) = |u(1, \Delta t) - u_{exact}(1)| \tag{1.2}$$

とする。以下に示す 5 つのアルゴリズムに対して、$E(\Delta t)$ の関数形（指数関数？ベキ関数？）がわかるような図を作り、その指数（e^{at} や t^a の a のこと）を理論的な予測とともに表にまとめよ。アダムス・バッシュフォース法では必要な初期値は厳密解を使うこと。なお n 次のアルゴリズムは、1 ステップ Δt における数値解と厳密解の差は $O(\Delta t^{n+1})$ であることは使ってよい。前進オイラー法は 1 次である。図の説得力が上がるように、Δt の値を適切にサンプリングせよ（等間隔の点が望ましい)。

・前進オイラー (Forward Euler) 法 (FE)
・2 次アダムス・バッシュフォース (Adams-Bashforth) 法 (AB2)
・3 次アダムス・バッシュフォース法 (AB3)
・2 次ルンゲ・クッタ (Runge-Kutta) 法 (RK2)（あるいはホイン法）
・4 次ルンゲ・クッタ法 (RK4)

1.2　原理・方法の詳細

常微分方程式を考える：

$$u' = f(t, u(t)), \quad u(0) = u_0 \tag{1.3}$$

ここで、$n = 0, ..., N$ とし、以下の離散変数を導入する：

$$t_n = n\Delta t, \quad u_n = u(t_n) \tag{1.4}$$

1.2.1　前進オイラー法

式 (1.3) を $t =\in [t_{n-1}, t_n]$ の範囲で積分すると

$$u_n - u_{n-1} = \int_{t_{n-1}}^{t_n} f(\tau, u(\tau))d\tau \tag{1.5}$$

となる。$\Delta t = t_n - t_{n-1}$ が十分小さければ、右辺の被積分関数の値は端点の値やそれらの平均で置き換えられる。そこで、$t \in [t_{n-1}, t_n]$ における f を $f(t_{n-1}, u_{n-1})$ で置き換える。$f(t_{n-1}, u_{n-1})$ は積分変数 τ に依存しないので、得られる結果は

$$u_n \approx u_{n-1} + f(t_{n-1}, u_{n-1})\Delta t \tag{1.6}$$

となる。式 (1.6) が前進オイラー法である。

1.2.2　2 次アダムス・バッシュフォース法、3 次アダムス・バッシュフォース法

式 (1.5) の中の f を、τ に関する N 次多項式 $p_N(\tau)$ で近似する：

$$\int_{t_{n-1}}^{t_n} f(\tau, u(\tau))d\tau \approx \int_{t_{n-1},t_n} p_N(\tau)d\tau \tag{1.7}$$

$p_N(\tau)$ が τ の多項式なので、式 (1.7) の積分は解析的に実行できる。

p_N を Lagrange 補間により構成することを考える。これには $N+1$ 個の点が必要である。陽解法なので、

$$(t_{n-1}, f_{n-1}), (t_{n-2}, f_{n-2}), ..., (t_{n-N-1}, f_{n-N-1})$$

を使う。結果として、$N = 1$ のときは

$$u_n \approx u_{n-1} + \frac{\Delta t}{2}(3f_{n-1} - f_{n-2}) \tag{1.8}$$

また、$N = 2$ のときは

$$u_n \approx u_{n-1} + \frac{\Delta t}{12}(23f_{n-1} - 16f_{n-2} + 5f_{n-3}) \tag{1.9}$$

となる。式 (1.8) および式 (1.9) がそれぞれ 2 次アダムス・バッシュフォース法および 3 次アダムス・バッシュフォース法である。

1.2.3　2 次ルンゲ・クッタ法、4 次ルンゲ・クッタ法

式 (1.6) を少し変形した次の形

$$\begin{cases} u_n = u_{n-1} + \dfrac{\Delta}{2}(f_{n-1} + f(t_n, u_n^*)) \\ u_n^* = u_{n-1} + f_{n-1}\Delta t \end{cases} \tag{1.10}$$

を考える。式 (1.10) が 2 次ルンゲ・クッタ法である。

また、4 次ルンゲ・クッタ法は

$$u_n = u_{n-1} + \tfrac{\Delta t}{6}(F_1 + 2F_2 + 2F_3 + F_4) \tag{1.11}$$

$$\begin{cases} F_1 = f(t_{n-1}, u_{n-1}) \\ F_2 = f(t_{n-1} + \dfrac{\Delta}{2}, u_{n-1} + F_1\dfrac{\Delta}{2}) \\ F_3 = f(t_{n-1} + \dfrac{\Delta}{2}, u_{n-1} + F_2\dfrac{\Delta}{2}) \\ F_4 = f(t_n, u_{n-1} + F_3\Delta) \end{cases} \tag{1.12}$$

と書ける。

1.3 理論予測

$E(\Delta t)$ の関数系の指数を予測する。

u_n をテイラー展開すると、

$$u_n = u_{n-1} + u'_{n-1}\Delta t + \frac{u"_{n-1}}{2}\Delta t^2 + ... \tag{1.13}$$

となり、右辺第 2 項までがオイラー法である。よって、1 ステップ Δt における数値解と厳密解の差は $O(\Delta t^2)$ である。したがって、n 回反復後に求まる u_n の誤差 $E(\Delta t)$ は、$O(\Delta t)$ である。

他の n 次のアルゴリズムでも、同様にして 1 ステップ Δt における数値解と厳密解の差は $O(\Delta t^{n+1})$ であり、n 回反復後に求まる u_n の誤差 $E(\Delta t)$ は、$O(\Delta t^n)$ であることがわかる。よって、各アルゴリズムにおける $O(\Delta t^{n+1})$ は以下の表ようになると予測される。

アルゴリズム	$E(\Delta t)$ の次数
前進オイラー法	1
2 次アダムス・バッシュフォース法	2
3 次アダムス・バッシュフォース法	3
2 次ルンゲ・クッタ法	2
4 次ルンゲ・クッタ法	4

1.4 実験方法

式 (1.1) より、厳密解は

$$u_{exact} = e^t \quad u_{exact(1)} = e \tag{1.14}$$

であり、初期値は

$$u_0 = 1, \quad f_0 = 1, \quad f_{-1} = e^{-\Delta t}, \quad f_{-2} = e^{-2\Delta t} \tag{1.15}$$

である。各アルゴリズムにおいて、

$$\Delta t = 2^0, 2^{-1}, ..., 2^{-11} \tag{1.16}$$

とし、各 Δt における $u(1)$ と $u_{exact}(1)$ との差の絶対値を求める。

1.4.1 前進オイラー法

```
1  #include <stdio.h>
2  #include <math.h>
3
4  double FE(double dt){
5      double u = 1.0;
6
7      for(double t=0; t<1; t+=dt){
8          u = u + u*dt;
9      }
10
11     return fabs(u-exp(1));
12 }
13
14 int main(void){
15     FILE *fp;
```

```
16      fp = fopen("3-1","w");
17
18      for(double dt=1; dt>=pow(2,-11); dt/=2){
19          printf("%.16f␣%.16f\n", dt, FE(dt));
20          fprintf(fp,"%.16f␣%.16f\n", dt, FE(dt));
21      }
22
23      fclose(fp);
24  }
```

1.4.2 2 次アダムス・バッシュフォース法

```
1   #include <stdio.h>
2   #include <math.h>
3
4   double AB2(double dt){
5       double u = 1.0;
6       double u1 = u;
7       double u2 = exp(-dt);
8
9       for(double t=0; t<1; t+=dt){
10          u = u1 + (3*u1-u2)*dt/2;
11          u2 = u1;
12          u1 = u;
13      }
14
15      return fabs(u-exp(1));
16  }
17
18  int main(void){
19      FILE *fp;
20      fp = fopen("3-2","w");
21
22      for(double dt=1; dt>=pow(2,-11); dt/=2){
23          printf("%.16f␣%.16f\n", dt, AB2(dt));
24          fprintf(fp,"%.16f␣%.16f\n", dt, AB2(dt));
25      }
26
27      fclose(fp);
28  }
```

1.4.3 3 次アダムス・バッシュフォース法

```
1   #include <stdio.h>
2   #include <math.h>
3
4   double AB3(double dt){
5       double u = 1.0;
6       double u1 = u;
7       double u2 = exp(-dt);
8       double u3 = exp(-2*dt);
9
10      for(double t=0; t<1; t+=dt){
11          u = u1 + (23*u1-16*u2+5*u3)*dt/12;
12          u3 = u2;
13          u2 = u1;
14          u1 = u;
15      }
16
17      return fabs(u-exp(1));
18  }
19
```

```
20  int main(void){
21      FILE *fp;
22      fp = fopen("3-3","w");
23
24      for(double dt=1; dt>=pow(2,-11); dt/=2){
25          printf("%.16f␣%.16f\n", dt, AB3(dt));
26          fprintf(fp,"%.16f␣%.16f\n", dt, AB3(dt));
27      }
28
29      fclose(fp);
30  }
```

1.4.4 2 次ルンゲ・クッタ法

```
1   #include <stdio.h>
2   #include <math.h>
3
4   double RK2(double dt){
5       double u = 1.0;
6       double u0;
7
8       for(double t=0; t<1; t+=dt){
9           u0 = u + u*dt;
10          u = u + (u+u0)*dt/2;
11      }
12
13      return fabs(u-exp(1));
14  }
15
16  int main(void){
17      FILE *fp;
18      fp = fopen("3-4","w");
19
20      for(double dt=1; dt>=pow(2,-11); dt/=2){
21          printf("%.16f␣%.16f\n", dt, RK2(dt));
22          fprintf(fp,"%.16f␣%.16f\n", dt, RK2(dt));
23      }
24
25      fclose(fp);
26  }
```

1.4.5 4 次ルンゲ・クッタ法

```
1   #include <stdio.h>
2   #include <math.h>
3
4   double RK4(double dt){
5       double u = 1.0;
6       double f1;
7       double f2;
8       double f3;
9       double f4;
10
11      for(double t=0; t<1; t+=dt){
12          f1 = u;
13          f2 = u + f1*dt/2;
14          f3 = u + f2*dt/2;
15          f4 = u + f3*dt;
16          u = u + (f1+2*f2+2*f3+f4)*dt/6;
17      }
18
19      return fabs(u-exp(1));
```

```
20  }
21
22  int main(void){
23      FILE *fp;
24      fp = fopen("3-5","w");
25
26      for(double dt=1; dt>=pow(2,-11); dt/=2){
27          printf("%.16f␣%.16f\n", dt, RK4(dt));
28          fprintf(fp,"%.16f␣%.16f\n", dt, RK4(dt));
29      }
30
31      fclose(fp);
32  }
```

1.5 結果

各アルゴリズムの結果を図 1 に示す。

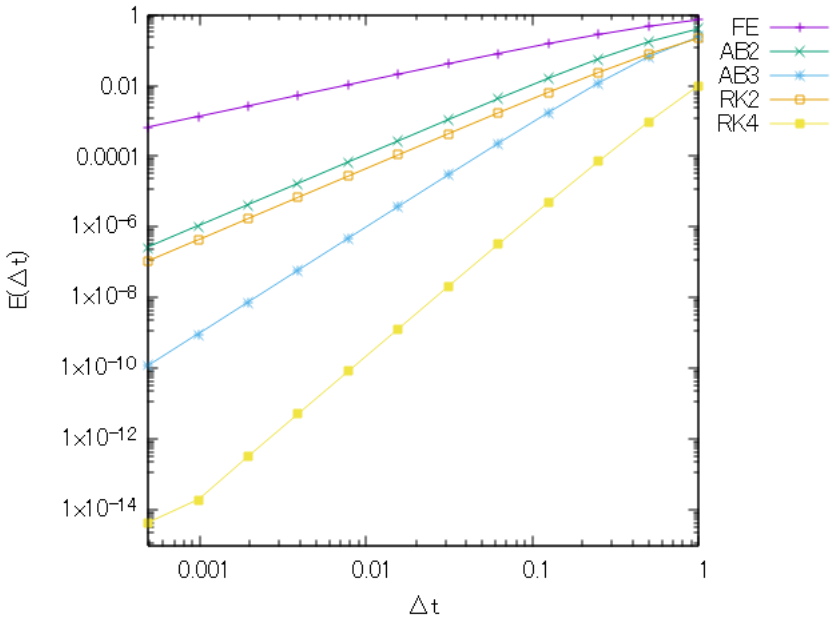

図 1: 数値解と厳密解の差

1.6 考察

図 1 に小節 (1.3) の予測を重ねた様子を図 2 に示す。

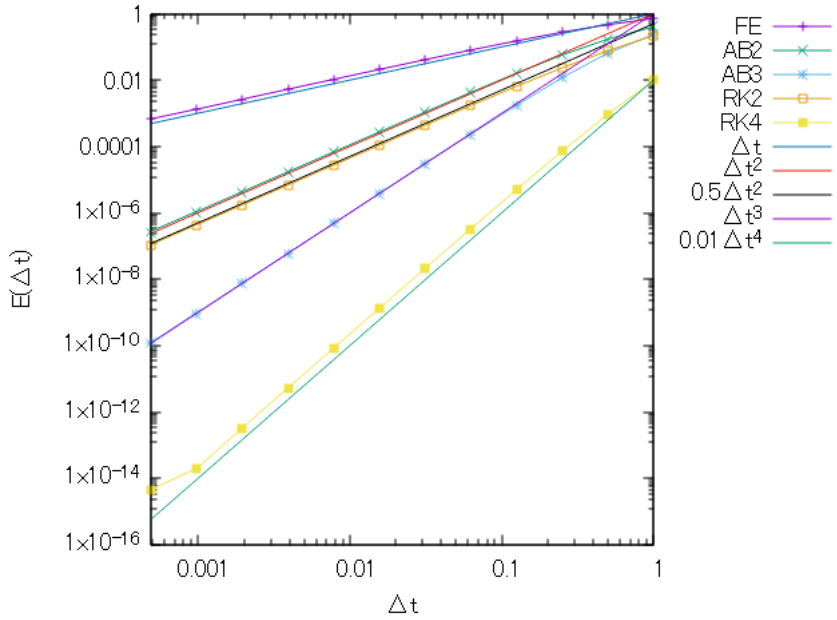

図 2: 数値解と厳密解の差とその予測

図 2 より、実験から得られた結果と理論値が一致することが分かる。

2　課題 4

2.1　はじめに

常微分方程式

$$u' = -\alpha u + \beta, \quad u(0) = u_0, \quad \alpha > 0, \beta \in \mathbb{R} \tag{2.1}$$

を考える。2 つのアルゴリズム

・クランク・ニコルソン (Crank-Nicolson) 法 (CN)

・2 次ルンゲ・クッタ法 (RK2)（あるいはホイン法）

において、ステップ幅 Δt に対するアルゴリズムの安定性を理論的に調べよ。さらに、$u_0 = 1$, $\alpha = 10$, $\beta = 1$ として数値計算により理論結果を確認せよ。安定・不安定の双方につき数値実験せよ。

2.2　原理・方法の詳細

式 (1.3) の常微分方程式を、式 (1.4) の離散変数を用いて考える。

2.2.1　クランク・ニコルソン法

式 (1.5) の中の f を、積分範囲の二つの端点

$$(t_{n-1}, f_{n-1}), \quad (t_n, f_n)$$

を通る 1 次関数で近似する。このとき f は

$$f(\tau, u(\tau)) \approx \frac{\tau - t_{n-1}}{t_n - t_{n-1}} f_n + \frac{\tau - t_n}{t_{n-1} - t_n} f_{n-1} \tag{2.2}$$

と表すことができる。式 (2.2) は τ の 1 次関数なので、式 (1.5) の積分は容易に実行できて、

$$\int_{t_{n-1}}^{t_n} f(\tau, u(\tau)) \approx \frac{\Delta t}{2}(f_{n-1} + f_n) \tag{2.3}$$

を得る。つまり、u_n は、

$$u_n \approx u_{n-1} + \frac{\Delta t}{2}(f_{n-1} + f_n) \tag{2.4}$$

であり、式 (2.4) がクランク・ニコルソン法である。

2.2.2　2 次ルンゲ・クッタ法

小小節 (1.2.3) を参照。

2.2.3 安定性

アルゴリズムの安定・不安定を明確にするために、以下では $\lim_{t\to\infty}|u(t)|=\infty$ とならないような問題を考える。また、議論を簡単にするため、差分化が以下の形で書けるとする。

$$u_n = a_1 u_{n-1} + a_0 \tag{2.5}$$

数値計算のアルゴリズムが不安定であるとき、$\lim_{t\to\infty}|u(t)|=\infty$ となるはずである。そこで、式 (2.5) を満たす数列 u_n を、$u_n=\lambda^n$ の形で探してみる。$u_n=\lambda^n$ を代入して整理すると、式 (2.5) は

$$\lambda^n - a_1\lambda^{n-1} - a_0 = 0 \tag{2.6}$$

と書ける。まず、同次解を求める。$\lambda\neq 0$ として、式 (2.6) の同次方程式

$$\lambda^n - a_1\lambda^{n-1} = 0 \tag{2.7}$$

を解くと、$\lambda=a_1$ を得る。一方、式 (2.5) の非同次解として $u_n=a_0/(1-a_1)$ が見つかる。すなわち、式 (2.5) の解は、c_1 を未定係数として、

$$u_n = c_1 a_1^n + a_0/(1-a_1) \tag{2.8}$$

である。u_0 が与えられているので、$c_1=u_0-a_0/(1-a_1)$ と決まる。結局、式 (2.5) の解は、

$$u_n = \left(u_0 - \frac{a_0}{1-a_1}\right)a_1^n + \frac{a_0}{1-a_1} \tag{2.9}$$

である。もし $|a_1|>1$ だとすると、式 (2.9) は $n\to\infty$ で発散してしまうことが分かる。

2.3 理論予測

ステップ幅 Δt に対する各アルゴリズムの安定性を調べる。

2.3.1 クランク・ニコルソン法

式 (2.1) をクランク・ニコルソン法で離散化すると、

$$\begin{aligned} u_n &= u_{n-1} + \frac{\Delta t}{2}(-\alpha u_{n-1}+\beta) + (-\alpha u_n + \beta) \\ &= \frac{1-\frac{\Delta t}{2}\alpha}{1+\frac{\Delta t}{2}\alpha}u_{n-1} + \frac{\beta\Delta t}{1+\frac{\Delta t}{2}\alpha} \end{aligned} \tag{2.10}$$

と変形できるので、

$$a_1 = \frac{1-\frac{\Delta t}{2}\alpha}{1+\frac{\Delta t}{2}\alpha},\quad a_0 = \frac{\beta\Delta t}{1+\frac{\Delta t}{2}\alpha} \tag{2.11}$$

である。$\alpha>0$ に対して、クランク・ニコルソン法では常に $|a_1|<1$ が満たされるため、クランク・ニコルソン法は任意の $\Delta t>0$ に対して安定である。

2.3.2 2 次ルンゲ・クッタ法

式 (2.1) を 2 次ルンゲ・クッタ法で離散化すると、

$$\begin{aligned} u_n &= u_{n-1} + \frac{\Delta t}{2}\langle[-\alpha u_{n-1} + \beta] + [-\alpha\{u_{n-1} + (-\alpha u_{n-1} + \beta)\Delta t\} + \beta]\rangle \\ &= \{1 - \alpha\Delta t + \frac{(\alpha\Delta t)^2}{2}\}u_{n-1} + \beta\Delta t - \frac{\alpha\beta(\Delta t)^2}{2} \end{aligned} \tag{2.12}$$

と変形できるので、

$$a_1 = 1 - \alpha\Delta t + \frac{(\alpha\Delta t)^2}{2}, \quad a_0 = \beta\Delta t - \frac{\alpha\beta(\Delta t)^2}{2} \tag{2.13}$$

である。つまり、$|a_1| \leq 1$ となるためには、

$$0 \leq \Delta t \leq \frac{2}{\alpha} \tag{2.14}$$

が要求される。

2.4 実験方法

式 (2.1) に、$u_0 = 1$,$\alpha = 10$,$\beta = 1$ を代入した

$$u' = -10u + 1, \quad u(0) = 1, \tag{2.15}$$

を考える。各アルゴリズムにおいて、

$$\Delta t = 0.01, 0.19, 0.205 \tag{2.16}$$

とし、各 Δt に対するアルゴリズムの安定性を調べる。

2.4.1 クランク・ニコルソン法

式 (2.4) に $f = -10u + 1$ を代入して整理すると、

$$u_n = \frac{1 - 5\Delta t}{1 + 5\Delta t}u_{n-1} + \frac{\Delta t}{1 + 5\Delta t} \tag{2.17}$$

となる。

```
1  #include <stdio.h>
2  #include <math.h>
3
4  double CN(double dt){
5      FILE *fp;
6      fp = fopen("4-1-001","w");
7
8      double u = 1.0;
9
10     for(double t=0; t<10; t+=dt){
11         u = (1-5*dt)/(1+5*dt)*u + dt/(1+5*dt);
12
13         printf("%.16f %.16f\n",t,u);
14         fprintf(fp,"%.16f %.16f\n",t,u);
15     }
16
17     fclose(fp);
18 }
```

```
19
20  int main(void){
21      double dt[] = {0.01, 0.19, 0.205};
22      CN(dt[0]);
23  }
```

2.4.2 2 次ルンゲ・クッタ法

```
1   #include <stdio.h>
2   #include <math.h>
3
4   double RK2(double dt){
5       FILE *fp;
6       fp = fopen("4-2-001","w");
7
8       double u = 1.0;
9       double u0;
10
11      for(double t=0; t<10; t+=dt){
12          u0 = u + (-10*u+1)*dt;
13          u = u + (-10*u+1-10*u0+1)*dt/2;
14
15          printf("%.16f␣%.16f\n",t,u);
16          fprintf(fp,"%.16f␣%.16f\n",t,u);
17      }
18
19      fclose(fp);
20  }
21
22  int main(void){
23      double dt[] = {0.01, 0.19, 0.205};
24      RK2(dt[0]);
25  }
```

2.5 結果

2.5.1 クランク・ニコルソン法

結果を図 3 に示す。

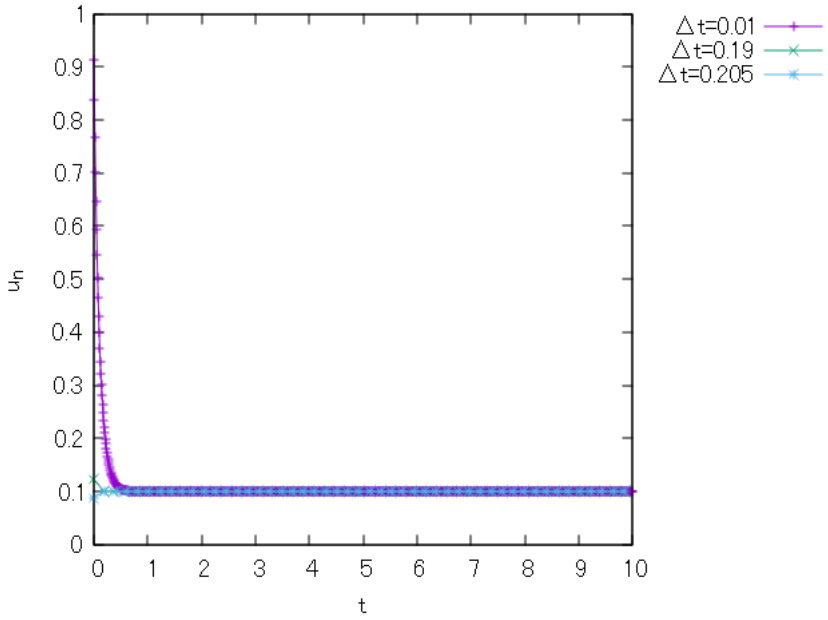

図 3: クランク・ニコルソン法

2.5.2　2 次ルンゲ・クッタ法

結果を図 4 に示す。

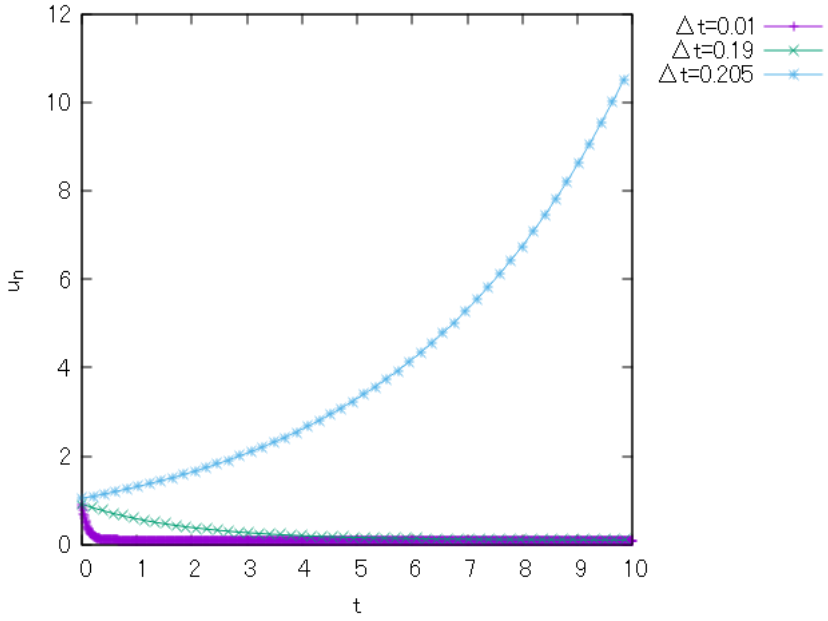

図 4: 2 次ルンゲ・クッタ法

2.6 考察

2.6.1 厳密解

式 (2.15) を解く。

$$
\begin{aligned}
u' &= -10u + 1 \\
\frac{du}{-10u+1} &= dt && (2.18)\\
\int \frac{du}{-10u+1} &= \int dt && (2.19)\\
-\frac{\log|-10u+1|}{10} &= t + C && (2.20)\\
-10u + 1 &= C_1 e^{-10t} && (2.21)\\
u &= C_1 e^{-10t} + \frac{1}{10} && (2.22)
\end{aligned}
$$

となり、$u(0) = 1$ より、$C_1 = 9/10$ である。したがって、

$$u = \frac{9}{10} e^{-10t} + \frac{1}{10} \tag{2.23}$$

よって、$t \to \infty$ としたとき、$u \to 1/10$ となる。

2.6.2 クランク・ニコルソン法

図 3 より、いずれの Δt に対しても安定である。これは小小節 (2.3.1) の理論予測と一致する。

2.6.3 2 次ルンゲ・クッタ法

図 4 より、$\Delta t = 0.01, 0.19$ に対しては安定であるが、$\Delta t = 0.205$ に対しては不安定である。小小節 (2.3.2) より、$\alpha = 10$ のとき、$0 \leq \Delta t \leq 0.2$ を満たす Δt に対して安定である。よって、実験結果は理論予測と一致している。

3 課題 5

3.1 はじめに

微分方程式

$$u' = -2u + 1 \tag{3.1}$$

の厳密解を求め、任意の初期値に対して $t \to \infty$ で有限値に収束することを確認せよ。アルゴリズム

$$u_n = u_{n-2} + 2\Delta t f(t_{n-1}, u_{n-1}) \tag{3.2}$$

では数値解を得ることが不可能であることを安定性の観点から理論的に示せ。また $u(0) = 1$ として数値計算を実行し、理論予測を確認せよ。

3.2 理論予測

3.2.1 収束性

式 (3.1) を解く。

$$\begin{aligned}
u' &= -2u + 1 && (3.3)\\
\frac{du}{-2u+1} &= dt && (3.4)\\
\int \frac{du}{-2u+1} &= \int dt && (3.5)\\
-\frac{\log|-2u+1|}{2} &= t + C && (3.6)\\
-2u + 1 &= C_1 e^{-2t} && (3.7)\\
u &= C_1 e^{-2t} + \frac{1}{2} && (3.8)
\end{aligned}$$

これより、任意の初期値に対して $t \to \infty$ としたとき、$u \to 1/2$ となる。

3.2.2 安定性

式 (3.1) を式 (3.2) のアルゴリズムで離散化すると、

$$u_n = -4u_{n-1}\Delta t + u_{n-2} + 2\Delta t \tag{3.9}$$

$u_n = \lambda^n$ を代入して整理すると、

$$\lambda^n + 4\lambda^{n-1}\Delta t - \lambda^{n-2} - 2\Delta t = 0 \tag{3.10}$$

と書ける。$\lambda \neq 0$ として、式 (3.10) の同次方程式

$$\lambda^n + 4\lambda^{n-1}\Delta t - \lambda^{n-2} = 0 \tag{3.11}$$

を解くと、

$$\begin{cases} \lambda_1 = -2\Delta t + \sqrt{4(\Delta t)^2 + 1} \\ \lambda_2 = -2\Delta t - \sqrt{4(\Delta t)^2 + 1} \end{cases} \tag{3.12}$$

を得る。一方、式 (3.9) の非同次解として $u_n = 1/2$ が見つかる。すなわち、式 (3.9) の解は、c_1,c_2 を未定係数として、

$$u_n = c_1\lambda_1^n + c_2\lambda_2^n + \frac{1}{2} \tag{3.13}$$

である。ここで、

$$|\lambda_1| \quad = |-2\Delta t + \sqrt{4(\Delta t)^2 + 1}| < |-2\Delta t + \sqrt{(2\Delta t + 1)^2}| = 1 \tag{3.14}$$

$$|\lambda_2|^2 \quad = 8(\Delta t)^2 + 4\Delta\sqrt{4(\Delta t)^2 + 1} + 1 > 1 \tag{3.15}$$

より、$n \to \infty$ としたとき、$\lambda_1^n \to 0$, $\lambda_2^n \to \infty$ となり、u_n は特定の値に収束しない。

したがって、式 (3.2) のアルゴリズムは任意の $\Delta t > 0$ に対して不安定である。

3.3 実験方法

初期値は $u(0) = 1$ である。

また、$\Delta t = 0.01, 0.19, 0.205$ とし、各 Δt に対するアルゴリズムの安定性を調べる。

```
1  #include <stdio.h>
2  #include <math.h>
3
4  double fu(double dt){
5      FILE *fp;
6      fp = fopen("5-001","w");
7
8      double u = 1.0;
9      double u1 = u;
10     double u2 = exp(-dt)/2 + 0.5;
11
12     for(double t=0; t<10; t+=dt){
13         u = u2 + 2*(-2*u1+1)*dt;
14         u2 = u1;
15         u1 = u;
16
17         printf("%.16f␣%.16f\n",t,u);
18         fprintf(fp,"%.16f␣%.16f\n",t,u);
19     }
20
21     fclose(fp);
22 }
23
24 int main(void){
25     double dt[] = {0.01, 0.19, 0.205};
26     fu(dt[0]);
27 }
```

3.4 結果

結果を図 5 に示す。

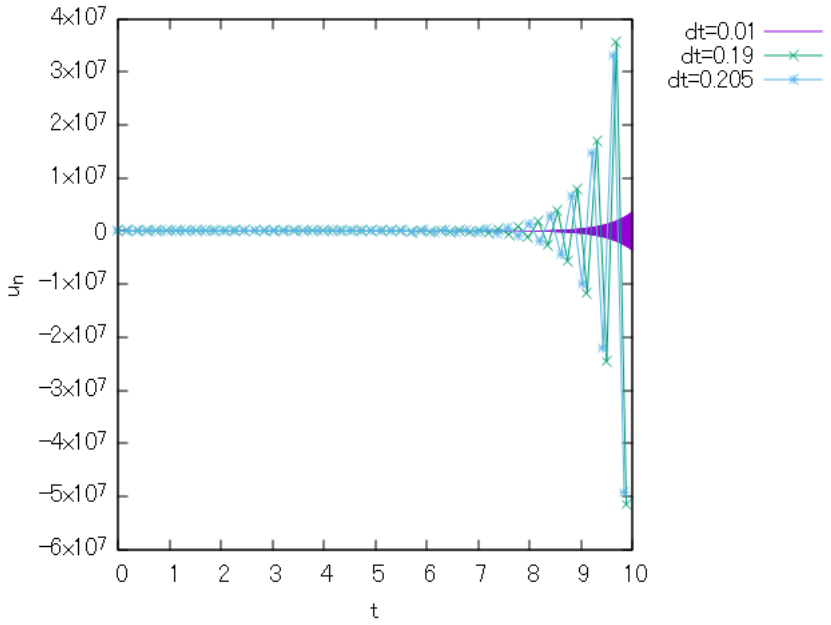

図 5: 数値解

3.5 考察

図 5 より、いずれの Δt に対しても不安定である。これは小節 (3.2) の理論予測と一致する。

また、各 Δt に対して $^{\exists}t_0$ が決まり、$t > t_0$ の範囲で数値解が発散していく様子が観察できる。t_0 の値は Δt が小さいほど大きくなっている。

4　課題 7

4.1　はじめに

ローレンツ方程式

$$\begin{cases} x' = \sigma(y - x) \\ y' = rx - y - xz \\ z' = xy - bz \end{cases} \tag{4.1}$$

を考える。初期値を $(x(0), y(0), z(0)) = (1 + \epsilon, 0, 0)$ とし、定数を $\sigma = 10, b = 8/3, r = 28$ 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、時間区間 $t \in [0, 100]$ における軌道を求めるものとする。

4.1.1

$\epsilon = 0$ とする。$x(t)$ を t の関数として図示せよ。また軌道 $(x(t), y(t), z(t))$ を xyz 空間に図示せよ。

4.1.2

ϵ を変化させたときの解をそれぞれ

$$\epsilon = 0 \text{ の解：} \quad x_0(t) = (x_0(t), y_0(t), z_0(t)) \tag{4.2}$$

$$\epsilon = 10^{-n} \text{の解：} \quad x_n(t) = (x_n(t), y_n(t), z_n(t))(n = 1, 2, ...) \tag{4.3}$$

とする。x の初期値が ϵ だけ異なる $x_0(t)$ と $x_n(t)$ の差

$$\Delta_n(t) = \|x_n(t) - x_0(t)\|(n = 1, 2, ...) \tag{4.4}$$

を考える。$\|\cdot\|$ はユークリッド距離である。$n = 2, 4, 6$ を取った時、$\Delta_n(t)$ について以下のことが分かっているとする；

ー $\log_{10} \Delta_n(t)$ を t の関数としてプロットしたとき、十分大きい t を取ると、ある値（漸近値）の近くで変動する。

ー漸近値に至る少し前の時間帯（初期値付近ではない）では $\Delta_n(t)$ の関数系（指数関数かべキ関数）と指数は $n = (2, 4, 6)$ に依存しない。

このとき、片対数表示の図と両対数表示の図を作成・比較しながら、漸近値に至る少し前の時間帯における $\Delta_n(t)$ の関数系と指数を定めよ。指数はおおよその値で構わない。また観測結果から分かることを議論せよ。

4.2　実験方法

4.2.1

```c
#include <stdio.h>
#include <math.h>

double dx(double x, double y, double z){
    return 10*(y-x);
}

double dy(double x, double y, double z){
    return 28*x-y-x*z;
}

double dz(double x, double y, double z){
    return x*y-z*8/3;
}

double RKx(double x, double y, double z){
    double dt = 0.01;
    double f1 = dx(x,y,z);
    double f2 = dx(x,y,z) + f1*dt/2;
    double f3 = dx(x,y,z) + f2*dt/2;
    double f4 = dx(x,y,z) + f3*dt;

    return x + (f1+2*f2+2*f3+f4)*dt/6;
}

double RKy(double x, double y, double z){
    double dt = 0.01;
    double f1 = dy(x,y,z);
    double f2 = dy(x,y,z) + f1*dt/2;
    double f3 = dy(x,y,z) + f2*dt/2;
    double f4 = dy(x,y,z) + f3*dt;

    return y + (f1+2*f2+2*f3+f4)*dt/6;
}

double RKz(double x, double y, double z){
    double dt = 0.01;
    double f1 = dz(x,y,z);
    double f2 = dz(x,y,z) + f1*dt/2;
    double f3 = dz(x,y,z) + f2*dt/2;
    double f4 = dz(x,y,z) + f3*dt;

    return z + (f1+2*f2+2*f3+f4)*dt/6;
}

int main(void){
    double x = 1.0;
    double y = 0;
    double z = 0;
    double dt = 0.01;

    FILE *fp;
    fp = fopen("7-1","w");

    for(double t=0; t<100; t+=dt){
        x = RKx(x,y,z);
        y = RKy(x,y,z);
        z = RKz(x,y,z);

        printf("%.16f␣%.16f␣%.16f␣%.16f\n",t,x,y,z);
        fprintf(fp,"%.16f␣%.16f␣%.16f␣%.16f\n",t,x,y,z);
    }

    fclose(fp);
}
```

4.2.2

```
#include <stdio.h>
#include <math.h>
#include <complex.h>

double dx(double x, double y, double z){
    return 10*(y-x);
}

double dy(double x, double y, double z){
    return 28*x-y-x*z;
}

double dz(double x, double y, double z){
    return x*y-z*8/3;
}

double RKx(double x, double y, double z){
    double dt = 0.01;
    double f1 = dx(x,y,z);
    double f2 = dx(x,y,z) + f1*dt/2;
    double f3 = dx(x,y,z) + f2*dt/2;
    double f4 = dx(x,y,z) + f3*dt;

    return x + (f1+2*f2+2*f3+f4)*dt/6;
}

double RKy(double x, double y, double z){
    double dt = 0.01;
    double f1 = dy(x,y,z);
    double f2 = dy(x,y,z) + f1*dt/2;
    double f3 = dy(x,y,z) + f2*dt/2;
    double f4 = dy(x,y,z) + f3*dt;

    return y + (f1+2*f2+2*f3+f4)*dt/6;
}

double RKz(double x, double y, double z){
    double dt = 0.01;
    double f1 = dz(x,y,z);
    double f2 = dz(x,y,z) + f1*dt/2;
    double f3 = dz(x,y,z) + f2*dt/2;
    double f4 = dz(x,y,z) + f3*dt;

    return z + (f1+2*f2+2*f3+f4)*dt/6;
}

double norm(double x, double y, double z, double x0, double y0, double z0){
    double u = (x-x0)*(x-x0) + (y-y0)*(y-y0) + (z-z0)*(z-z0);

    return sqrt(u);
}

int main(void){
    double dt = 0.01;
    double x0 = 1.0;
    double y0 = 0;
    double z0 = 0;

    double n = 2;
    double x = 1 + pow(10,-n);
    double y = 0;
    double z = 0;

    FILE *fp;
    fp = fopen("7-2-2","w");

```

```
67          for(double t=0; t<100; t+=dt){
68              x0 = RKx(x0,y0,z0);
69              y0 = RKy(x0,y0,z0);
70              z0 = RKz(x0,y0,z0);
71
72              x = RKx(x,y,z);
73              y = RKy(x,y,z);
74              z = RKz(x,y,z);
75
76              double u = norm(x,y,z,x0,y0,z0);
77
78              printf("%.16f␣%.16f\n",t,u);
79              fprintf(fp, "%.16f␣%.16f\n",t,u);
80          }
81
82          fclose(fp);
83      }
```

4.3 結果

4.3.1

結果を以下の図に示す。

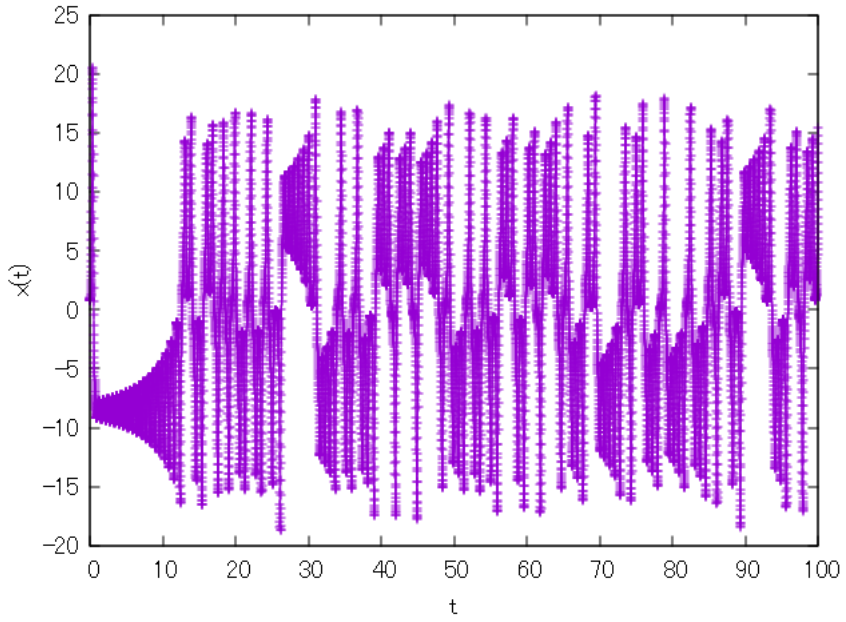

図 6: x(t)

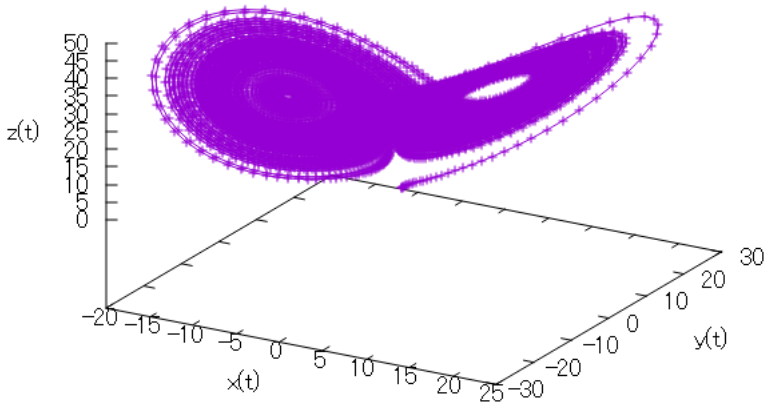

図 7: (x(t),y(t),z(t))

4.3.2

結果を以下の図に示す。

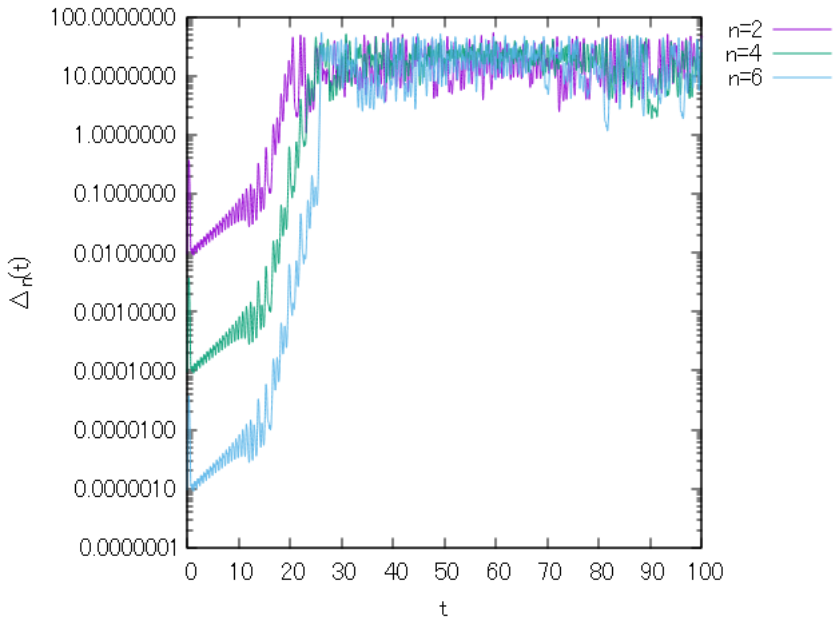

図 8: 片対数表示

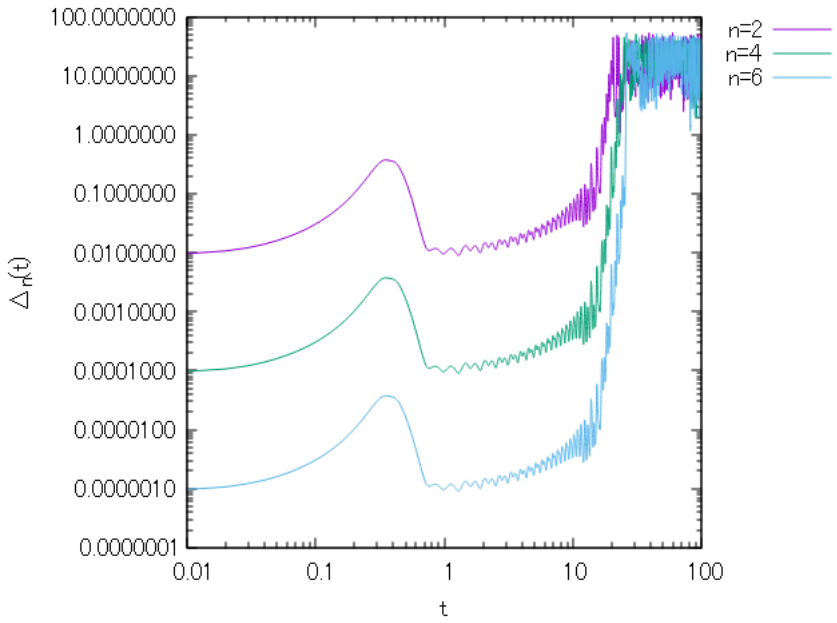

図 9: 両対数表示

4.4 考察

4.4.1

図 6 より、$x(t)$ は特定の値には収束せずに振動していることが分かる。
また、図 7 より、$y(t)$,$z(t)$ についても同様のことが言える。

4.4.2

図 8,9 より、$t \leq 10$ のとき、$\Delta_n(t)$ はおおよそ 10 から 40 の間で変動することが分かる。

漸近値に至る少し前の時間帯では $\Delta_n(t)$ の関数系と指数は n に依存していない。下図より、その関数系は指数関数であり、指数はおおよそ 7 である。

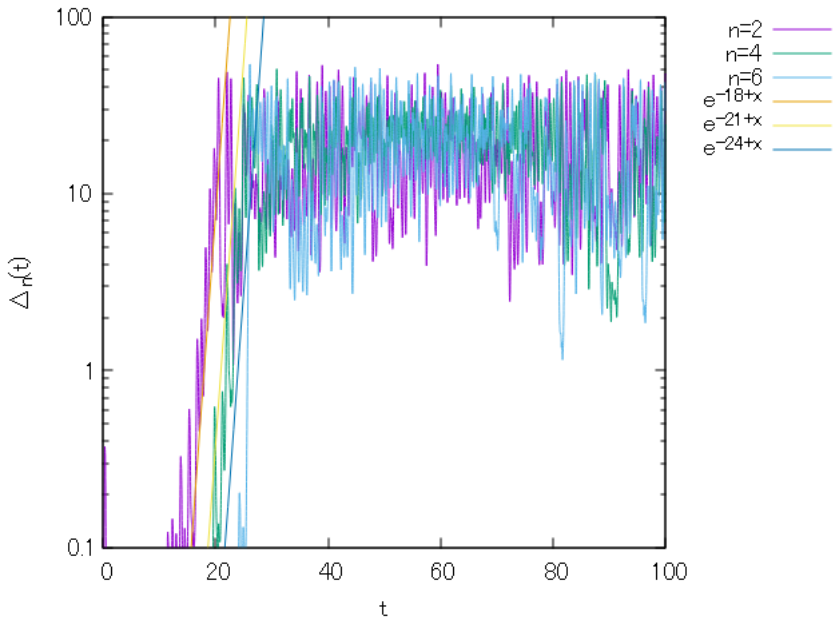

図 10: 片対数表示

5　課題 8

5.1　はじめに

次の連立微分方程式を考える。

$$\frac{dx_i}{dt} = \omega - \frac{K}{N}\sum_{j=1} N\sin(x_i - x_j), \quad (i = 1, ..., N) \tag{5.1}$$

x_i は単位円上の位置を表す。K は定数である。ω_i は定数であり次で与える：

$$\omega_i = \tan\left[\pi\left(\frac{i}{N+1} - \frac{1}{2}\right)\right], \quad (i = 1, ..., N) \tag{5.2}$$

また x_i の初期値 $x_i(0)$ は次で与える：

$$x_i(0) = y_i + 0.01\sin y_i, \qquad y_i := \frac{2\pi}{N}(i-1) \quad (i = 1, ..., N) \tag{5.3}$$

$x_i(t)$ が求まれば、時刻 t におけるオーダーパラメータと呼ばれる量

$$R = \sqrt{R_x^2 + R_y^2} \quad (0 \le R \le 1) \qquad R_x = \frac{1}{N}\sum_{j=1} N\cos x_j, \quad R_y = \frac{1}{N}\sum_{j=1} N\sin x_j \tag{5.4}$$

の値 $R(t)$ が求まる。

5.1.1

微分方程式 (5.1) をそのままコーディングすると、各 i に対して右辺第 2 項を計算するのに $O(N)$ かかり、i が N 個あるから計算量が $O(N^2)$ になる。しかし実際には $O(N)$ で計算可能であることと、その手順を議論せよ。
ヒント：式 (5.1) の sin 項を加法定理で分解し、$\sum_{j=1}^{N}$ を先に計算することを考えてみよ。すべての i に対して共通部分があれば計算は 1 回で済む。ただし 4 次ルンゲ・クッタ法の 1 ステップ中には「仮の x」が複数存在するので、「共通部分」も「仮の x」に対して計算しなければならないことに注意せよ。

5.1.2

以下ではステップ幅 $\Delta t = 0.01$ の 4 次ルンゲ・クッタ法を用いる。$N = 10^2, 10^3$ に対して、区間 $t \in [50, 100]$ における $R(t)$ の時間平均

$$\overline{R} := \frac{1}{50}\int_{50}^{100} R(t)dt \tag{5.5}$$

を求めよ。K の値は区間 $K \in [1, 3]$ において 0.1 刻みで動かし、$_overline R$ を K の関数としてプロットせよ。

5.1.3

$N \to \infty$ としたとき、$K = 2$ は臨界点 K_c と呼ばれている。なぜそのように呼ばれるのか、オーダーパラメータ R の意味（$R = 0$ と $R > 0$ の違いは？）に注意しながら $N = 10^2$ と $N = 10^3$ で結果を比較しながら議論せよ。

5.2 理論予測

5.2.1

式 (5.1) の sin 項を加法定理で分解すると、

$$\sin(x_i - x_j) = \sin x_i \cos x_j - \cos x_j \sin x = i \tag{5.6}$$

となるため、

$$\sum_{j=1}^{N} \sin(x_i - x_j) = \sin x_i \sum_{j=1}^{N} \cos x_j - \cos x_i \sum_{j=1}^{N} \sin x_j \tag{5.7}$$

となる。ここで、各 i に対して右辺を計算するのに $O(N)$ かかる。i を 1 つ計算するときに、任意の j に対して $\sin x_j, \cos x_j$ を求めている。そのため、その値を記憶しておけば、2 つ目以降の i を計算するのは $O(1)$ で済む。したがって、全体の計算量は

$$O(N) + O(1) + ... + O(1) = O(N) \tag{5.8}$$

ただし、式 (5.8) の左辺の $O(1)$ は $n-1$ 個続く。